Análise Comparativa de Algoritmos de Machine Learning na Detecção de Malwares em Dispositivos Android

Lucas P. Ferreira[1]

[1]Centro de Ciências Sociais e Tecnológicas – Universidade Católica de Pelotas (UCPel)
R. Gonçalves Chaves, 373 – 96015-560 – Pelotas – RS – Brasil

lucas.ferreira@sou.ucpel.edu.br

Abstract. *With the advent and ubiquity of mobile devices capable of running increasingly robust and complex software, along with the popularization of the Android OS, these devices became primary targets for data security threats. In this way, malware detection becomes necessary, since its dissemination is strongly linked to the way applications are distributed. Machine Learning algorithms are being explored by researchers in the detection of these malicious codes, using static and/or dynamic data extracted from these applications. In this paper, four classification algorithms are evaluated for malware detection. Combining two distinct datasets containing the same attributes, the correlation and training of each classifier was done by adjusting hyperparameters for the training data and applying cross-validation on the test data. From this data, the evaluation on the test set was computed using the accuracy, F1 Score, and the area over the ROC curve. The Random Forest algorithm obtained the best results, with an accuracy of 96,5%, F1 Score of 95,1%, and the area on the ROC curve of 95,5%.*

Resumo. *Com o advento e a onipresença dos dispositivos móveis capazes de executar softwares cada vez mais robustos e complexos, em conjunto com a popularização do Sistema Operacional Android, estes dispositivos tornaram-se alvos primários para ameaças à segurança de dados. Neste sentido, a detecção de malwares se faz necessária, uma vez que sua disseminação está fortemente ligada a forma como os aplicativos são distribuídos. Algoritmos de Machine Learning estão sendo explorados por pesquisadores na detecção destes códigos maliciosos, utilizando para tal dados estáticos e/ou dinâmicos extraídos destes aplicativos. Neste trabalho, são avaliados quatro algoritmos de classificação na detecção de malwares. A partir da junção de dois datasets distintos contendo os mesmos atributos, foi feita a correlação e o treinamento de cada classificador ajustando hiper parâmetros para os dados de treino e aplicada a validação cruzada nos dados de teste. A partir destes dados, foi computada a avaliação no conjunto de testes, utilizando a acurácia, o Score F1 e a curva ROC. O algoritmo Random Forest obteve os melhores resultados, com acurácia de 96,5%, Score F1 de 95,1% e área da curva ROC de 95,5%.*

1. Introdução

Na contemporaneidade, os *smartphones* se tornaram uma parte inseparável de nossas vidas. Segundo a GSMA Mobile Economy em seu relatório de 2020, a tendência é

que tenhamos cerca de 6 bilhões de dispositivos conectados à internet em 2020 e que esse número cresça ainda mais até 2023. Entre os diferentes sistemas operacionais para dispositivos móveis, o Android está no topo de todo o mercado mundial com 78% de participação no mercado [12]. O sistema operacional de código aberto abriu o caminho para fornecer *smartphones* acessíveis e de uso facilitado para todos os públicos. No entanto, a popularidade do Android também fez dele um alvo principal para hackers violarem sua segurança e obterem acesso a informações pessoais dos usuários. Um relatório publicado pela McAfee [13] descobriu que mais de 35 milhões de aplicações Android estavam contaminadas com algum tipo de *malware* apenas em 2019. A detecção de novas variantes de *malwares* se tornou um problema difícil para os métodos existentes, que se baseiam no modelo de assinatura de código. Como possível solução, algoritmos de Machine Learning estão sendo amplamente explorados pelos pesquisadores como alternativa para a detecção destas pragas virtuais.

Neste artigo, será avaliado o uso de quatro algoritmos de Machine Learning — Decision Tree, K-Nearest Neighbour, Linear SVM e Random Forest — como alternativa para a detecção de malwares com características estáticas. A partir da junção de dois datasets distintos, denominados Drebin-215 [3] e Malgenome-215 [15], foi gerado um dataset único que foi aplicado no teste destes algoritmos. Esses datasets foram pré-processados antes de sua junção para uma melhor distribuição destes dados. Assim, cada algoritmo foi avaliado depois de aplicar uma seleção de features, uma hiper parametrização e uma validação cruzada do tipo k-fold utilizando como métricas a acurácia, o score F1 e a curva ROC.

O artigo está estruturado em cinco seções. A Seção I (esta seção) apresenta a introdução do trabalho. A Seção II apresenta a metodologia aplicada. A Seção III descreve a fundamentação teórica dos algoritmos utilizados e como eles funcionam. A Seção IV apresenta, portanto, os resultados obtidos da execução desses algoritmos com suas métricas de avaliação. Finalmente, a seção V apresenta as considerações finais deste trabalho, bem como deixa sugestões para possíveis trabalhos futuros.

2. Metodologia

Nesta seção será descrito o processo que foi utilizado para o desenvolvimento deste trabalho, como mostrado na figura 1.

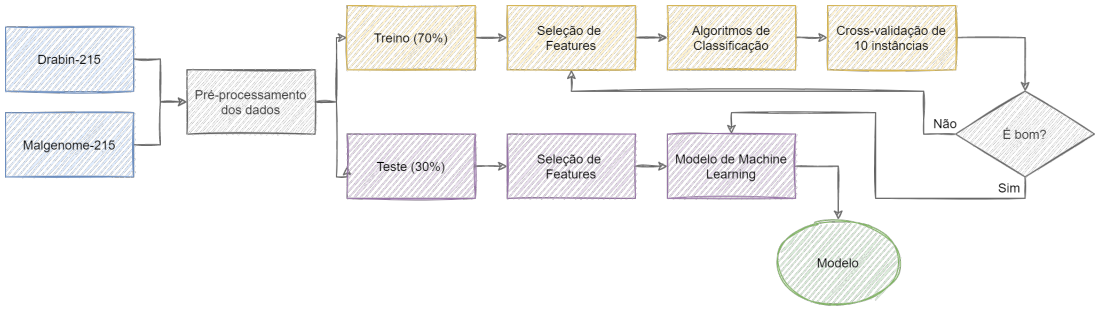

Figura 1: Fluxograma da metodologia aplicada

Em primeiro lugar, os dados foram pré-processados e unidos com o objetivo de se tornar um único dataset. Então este dataset foi dividido em porções de treino e teste

e, na porção de teste, foi aplicada uma seleção de features para que os modelos fossem treinados com ele. Ainda utilizando os dados de treino, uma hiper parametrização foi feita também, do tipo k-fold com o valor de k sendo 10, para cada um dos classificadores. Finalmente os modelos de classificação são escolhidos ao passo que os parâmetros escolhidos são avaliados no conjunto de dados de teste.

Como forma de avaliar tais mudanças, foram utilizadas as métricas de acurácia e o Score F1. Para julgar a eficiência de um classificador, a curva ROC também foi considerada na avaliação.

2.1. Aquisição do Dataset

Os dois datasets utilizados neste trabalho foram escolhidos levando em conta a continuação de [14], no qual foi provido como material suplementar junto com o artigo. As amostras dos dois datasets foram originadas de [3] e [15] respectivamente. Ambos os datasets contêm 215 features, consistindo de quatro sets distintos de dados, sendo eles:

- **Chamadas de API:** nomes de métodos extraídos de Android Application Packages (APKs) com a extensão .dex;
- **Permissões de Manifesto:** permissões declaradas no arquivo de manifesto AndroidManifest.xml;
- **Intenções:** usadas para comunicações entre outros aplicativos/serviços declados também no arquivo de manifesto AndroidManifest.xml; e
- **Assinaturas de Comando:** comandos Linux encontrados nos arquivos APK analisados.

Estes dados são utilizados pois, de acordo com [1], foi demonstrada a eficácia da utilização destes parâmetros aplicados juntamente a algoritmos de Machine Learning. Neste sentido, as features utilizadas neste trabalho, oriundas de [14], foram coletadas a partir de diversos APKs de aplicativos com uma ferramenta de análise e extração de dados desses mesmos arquivos. Todas as features combinam valores binários indicando a existência ou a não-existência de uma feature respectivamente. A classe de desfecho contém valores 'B', denotando que aquele arquivo é benigno ou 'S', denominando infecção por malware. Todavia, estes parâmetros também foram alterados de modo a manter a consistência dos dados de forma binária. A tabela 1 mostra detalhes dos dois datasets.

Dataset	# de Instâncias	# de Infectados	# de Benignos	# de Features
Drebin-215	15,036	5,560	9,476	215
Malgenome-215	3,799	1,260	2,539	215

Tabela 1: Características dos datasets selecionados.

2.2. Pré-processamento de Dados

Dos datasets coletados de [14], foram feitos alguns ajustes nos dados com o propósito de melhorar os resultados de classificação pelos algoritmos. Em primeiro lugar, foi verificado se o dataset continha dados em branco. 5 instâncias do dataset Drebin-215 continham valores em branco para uma feature em específico e foram removidas. O dataset Mangenome-215 não continha dados em branco.

Antes de realizar a união dos dois datasets, foram verificadas as features em comum entre os dois. Da verificação, 208 features eram comuns entre eles. O resto das features foram descartadas. As features de desfecho – 'B' para benigno e 'S' para maligno – foram convertidas para valores binários respectivamente. Então, foi feita a união dos dois datasets em um único, o qual foi usado neste trabalho. Os datasets combinados continham um conjunto de 208 features com um total de 18,830 instâncias, sendo 12,015 delas maliciosas (63,8%) e 6,815 (36,2%) limpas.

Finalmente, o dataset oriundo da junção foi dividido em uma proporção de 70:30 para treino e teste respectivamente. A divisão, entretanto, manteve a distribuição de classes entre ambos os conjuntos.

2.3. Seleção de Features

A seleção de features foi aplicada no conjunto de dados de treino utilizando uma correlação baseada em seleção de features (CfsSubsetEval) utilizando o Weka [7]. A função avalia o valor de um subconjunto de atributos, considerando a capacidade individual de cada feature junto com o grau de redundância entre eles [8]. Das 208 features do conjunto de dados de treino, 27 foram escolhidas pelo algoritmo consideradas como importantes para o desfecho final.

A predicabilidade de uma feature pode ser mensurada pela sua correlação em relação a classe desfecho. O valor de correlação perto de 0 significa que a feature não é muito correlata com a classe de desfecho, ou seja, não é uma feature efetiva para a predição. Por outro lado, o valor de correlação perto de -1 ou 1 significa que a feature é fortemente correlata com a classe de desfecho, ou seja, é uma funcionalidade efetiva para a predição. A fígura 2 mostra o diagrama de caixa da correlação absoluta dos valores de cada feature com a classe de desfecho do dataset, antes e depois da seleção das features.

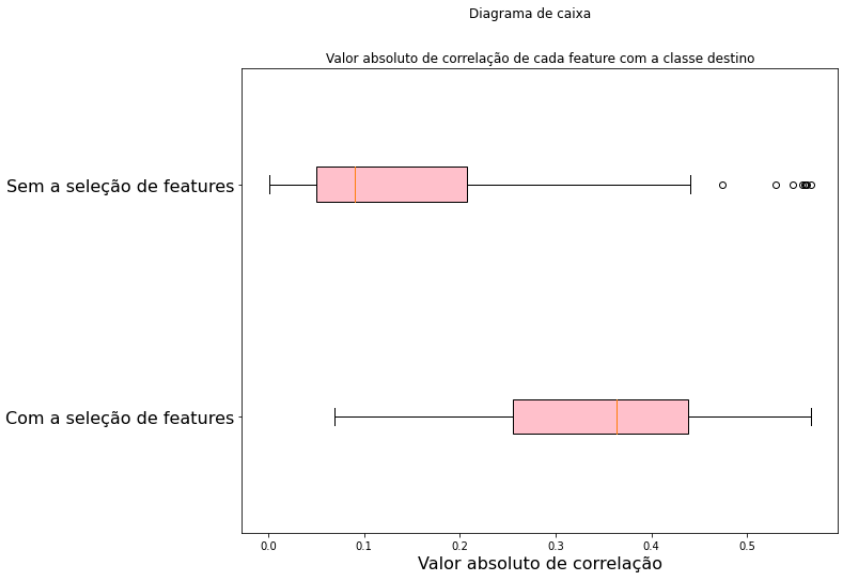

Figura 2: Valor absoluto de correlação de cada feature com a classe destino (desfecho)

A média absoluta de correlação de cada feature com respeito ao classe destino cresceu, portanto, de 0,15($\pm$0,14) sem nenhuma seleção de features para 0,35($\pm$0,14)

depois da seleção de features feita pelo CfsSubsetEval. Ainda da figura 2 podemos observar que, sem a seleção de features, a correlação dos valores se mostram positivamente enviesados (significando uma maior correlação absoluta tendendo a valores perto de 0). Entretanto, depois da seleção de features, a correlação dos valores se mostram negativamente enviesados (significando uma correlação absoluta com valores perto de 1).

A redundância entre as features pode ser medida utilizando a correlação de cada par de features. Um par de features com seu valor de correlação próximo de 0 tem uma redundância baixa, enquanto os valores de correlação perto de 1 ou -1 significam que eles possuem uma alta redundância entre si. A figura 3 mostra o diagrama de caixa da absoluta correlação de cada par de features antes e depois da seleção de features.

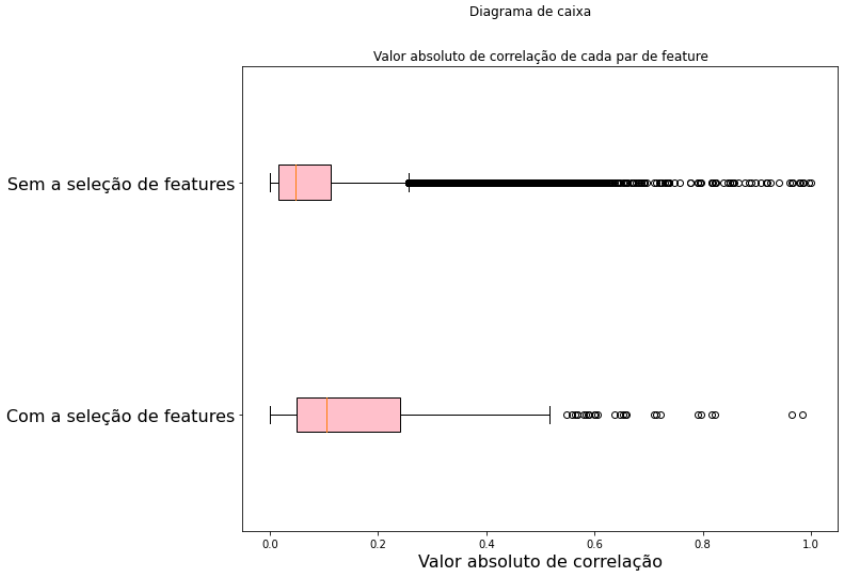

Figura 3: Valor absoluto de correlação de cada par de features

Dele, pode-se perceber que a média da correlação absoluta de cada par de features aumento de de 0,09($\pm$0,11) sem seleção de features para 0,18($\pm$0,19) após a seleção de features feita pelo CfsSubsetEval. Todavia, ainda da figura 3 é possível observar que os dados positivamente enviesados (significando uma correlação absoluta com valores perto de 0) são mantidos em ambos os casos.

2.4. Avaliação

Para cada um dos algoritmos, foram utilizados os parâmetros otimamente selecionados pela função GridSearchCV [11]. Estes parâmetros foram utilizados apenas no conjunto de dados de treino. No conjunto de dados de teste, não foi utilizado intencionalmente com o propósito de refletir um melhor uso em situações do mundo real, onde as variáveis podem alterar. Isso reflete em uma avaliação semelhante a de uma situação em produção com dados do mundo real.

3. Algoritmos

Existem diversos algoritmos desenvolvidos para as mais diversas variantes de um problema. Aqui, serão discutidos quatro destes algoritmos de Machine Learning utilizados

na resolução de problemas de aprendizado supervisionado. Segundo [10], o aprendizado supervisionado é utilizado quando se conhece o modelo que o programa deve seguir e a partir dele generaliza-se de modo a conseguir aplicá-lo a qualquer entrada.

Neste trabalho, serão aplicados os seguintes algoritmos: Decision Tree, K-Nearest Neighbour, Linear SVM e Random Forest.

3.1. Decision Tree

A técnica de árvore de decisão (Decision Tree), é um método usado em problemas de classificação e regressão, que particiona recursivamente um conjunto de treinamento, até que cada subconjunto resultante apresente casos de uma única classe. Para isso, na construção da árvore o algoritmo escolhido examina e compara a distribuição de classes.

A construção da árvore de decisão é fundamentada no modelo top-down, isto é, a sua estrutura inicia do nó raiz e vai até as suas folhas. Apesar dos algoritmos desse grupo conterem diferenças importantes em sua lógica de execução, todos utilizam da técnica de dividir para conquistar. Que diz que o problema deve ser dividido sucessivamente em vários problemas menores, até uma solução para cada um dos problemas mais simples seja encontrada. Nesse fundamento, os algoritmos tentam descobrir maneiras de dividir sucessivamente o conjunto em vários subconjuntos, até que cada um apresente apenas uma classe ou que uma das classes seja a maioria, não sendo necessário novas divisões [4].

Uma árvore de decisão geralmente possui em sua estrutura os seguintes componentes:

- **Nó:** São todos os elementos que aparecem na árvore;
- **Nó Raiz:** É o primeiro nó do topo da árvore;
- **Nó pai e nó filho:** São os nós logo abaixo do nó raiz que se dividem em sub nós. Um nó dividido em sub nós é chamado nó pai de sub nós, enquanto os sub nós são filhos do nó pai;
- **Nó de Decisão:** Quando um sub nó se divide em outros sub nós;
- **Folhas ou Terminal:** São os nós que não têm filhos, os últimos elementos da árvore.

Em sua interpretação cada nó de decisão realiza um teste para algum atributo, cada ramo descendente representa um valor desse atributo, o conjunto de ramificações são diferentes, as folhas definem uma classe, e cada percurso da árvore iniciando do nó raiz até a folha, constitui uma regra de classificação. Analisando essa estrutura, é possível extrair regras do tipo ”se-então”para melhor compreensão dos resultados.

Para definir as partições da árvore, a regra utilizada é o quão útil o atributo é para a classificação. A partir dessa regra, um determinado ganho de informação é aplicado a cada atributo. Dessa forma, o atributo escolhido como atributo teste para o nó corrente, é o que contém o maior ganho de informação. Por meio desta aplicação, um novo processo de partição é iniciado. Quando a árvore é usada para classificação, as regras de partição mais conhecidas são a entropia e o índice de Gini.

A entropia é o cálculo do ganho de informação que mede a impureza dos dados. Em um conjunto de dados, mede a falta de homogeneidade dos dados de entrada em

relação a sua classificação. Já o índice Gini, desenvolvido por Conrado Gini em 1912, calcula o grau de heterogeneidade dos dados, sendo usado para calcular a impureza de um nó.

Um problema encontrado no uso de árvores de decisão, é o *Overfitting*, que significa o ajuste exagerado dos dados de treinamento. Na execução do algoritmo de partição recursiva, o mesmo estende a sua profundidade até o ponto de classificar corretamente os elementos do conjunto de treinamento. Nesse processo, quando o conjunto de treinamento não possui ruído, o número de erros no treinamento pode ser zero. Porém, quando o conjunto possui ruído, ou quando o conjunto de treinamento não é representativo, este algoritmo pode produzir o *overfitting*.

Segundo o criador do método, uma maneira de impedir o problema do overfitting e melhorar a classificação é fazer a poda da árvore. A podagem da árvore tende a ser feita em duas situações. Para parar o crescimento da árvore mais cedo, conhecido como pré-podagem ou poda descendente. Ou com a árvore já completa, conhecido como pós-podagem ou poda ascendente.

Para determinar o número ideal de nós, utiliza-se uma representação gráfica que mostra o percentual de erro no conjunto de treinamento, versus o número de nós da árvore. Quando o erro no conjunto de teste começa a crescer, o número de nós nesse ponto é considerado ideal.

Algoritmos que usam a ideia das árvores de decisão são bastante aplicados por retornarem modelos preditivos de fácil interpretação, estabilidade e alta precisão. O método Top-Down Induction of Decision Tree (TDIDT) é um dos mais utilizados na aplicação de árvore de decisão e é refêrencia para outros algoritmos como o Classification and Regression Tree (CART) e o C5.0.

Em sua execução o TDIDT faz uma busca recursiva por atributos que melhor dividem o conjunto de observações em sub-conjuntos. O CART é caracterizado por ter a capacidade de assimilar relações entre dados, mesmo não havendo visíveis relações. Em seu uso faz a construção de árvores binárias, onde cada nó interno possui como saída dois nós filhos. Já no C5.0 o algoritmo transforma árvores treinadas em conjuntos de regras if-then, e avalia qual ordem essas regras devem ser aplicadas.

No seu uso prático, as árvores de decisão foram aplicadas em vários domínios da ciência e engenharia como pesquisas de produtos farmacêuticos, saúde pública, biologia celular, consumo de energia elétrica, estudos de transporte, entre outros.

3.2. K-Nearest Neighbour (KNN)

A técnica K-Vizinhos mais próximos (K-Nearest Neighbors) é um método classificador onde o aprendizado é fundamentado na similaridade que um dado (vetor) tem de outro. O treinamento utiliza um conjunto de dados composto por vetores n-dimensionais e cada elemento desse conjunto define um ponto no espaço n-dimensional.

No seu funcionamento, para descobrir a classe de um elemento que não pertença ao conjunto de dados de treinamento, o KNN busca K elementos do conjunto que estejam mais próximos do elemento desconhecido, isto é, que tenham a menor distância. Estes K elementos são chamados de K-vizinhos mais próximos, e verificando a qual classes esses K vizinhos pertencem, a classe com maior número de elementos será atribuída à classe

do elemento desconhecido.

Para o cálculo da distância entre dois pontos, as métricas mais comuns são a distância Euclidiana, Manhattan e Minkowski, sendo a primeira a mais utilizada.

Na utilização do algoritmo KNN, a sua execução ocorre por meio das seguintes etapas:

1. Recebe um dado não classificado;
2. Mede a distância (Euclidiana, Manhattan, Minkowski ou Ponderada) do dado desconhecido com todos os outros dados que já estão classificados;
3. Obtém-se as K menores distâncias;
4. Verifica-se a classe de cada um dos dados que tiveram a menor distância e se conta a quantidade de cada classe que aparece;
5. Se escolhe como resultado a classe que mais apareceu dentre os dados que tiveram as menores distâncias; e
6. Classifica-se o dado desconhecido com a classe escolhida do resultado da classificação que mais apareceu.

O classificador KNN possui o número de K-vizinhos como único parâmetro livre, o qual é controlado pelo usuário para obter uma melhor classificação. Esta classificação pode ser exaustiva computacionalmente se utilizado um conjunto com muitos dados. Entretanto, em aplicações específicas seu uso é recomendado.

3.3. Linear SVM

A técnica de máquina de vetores de suporte (Support Vector Machine) é um método de aprendizagem para problemas de reconhecimento de padrões. Foi introduzida por Vapnik em 1995 através da teoria estatística de aprendizagem, no qual utiliza do fundamento de separação ótima entre classes, onde se às classes são separáveis, então o resultado é obtido de modo a separar o máximo as classes [9].

Sua proposta está em resolver problemas de classificação e análise de regressão. No entanto é principalmente usado em problemas de classificação de duas classes, isto é, atuando como um classificador binário, onde é aplicado em tarefas de classificação de dados lineares e não lineares [2].

O funcionamento básico de uma SVM ocorre através do fornecimento de duas classes e um conjunto de pontos que pertencem a essas classes. Com essas informações um hiperplano é determinado separando os pontos de forma a colocar o maior número de pontos da mesma classe do mesmo lado, enquanto atribui o valor mais alto a distância de cada classe a esse hiperplano. A distância de uma classe a um hiperplano é a menor distância entre ele e os pontos dessa classe, no qual é denominado margem de separação. Já a construção do hiperplano é feita por um subconjunto dos pontos das duas classes, chamados de vetor de suporte [6].

Quando não é possível separar os dados linearmente, é utilizado um classificador não linear que usa funções do kernel para estimar as margens. O objetivo dessas funções é maximizar as margens entre os hiperplanos, sendo as funções do kernel mais empregadas a linear, polinomial, base radial e sigmóide [2]. No funcionamento dessas funções os dados de entrada são mapeados em um espaço de dimensão maior, onde os dados são separados por meio de um hiperplano, tornando-os linearmente separáveis [6].

3.4. Random Forest

A técnica floresta aleatória (Random Forest) é um método desenvolvido por Breiman em 2001 para resolver problemas de classificação e regressão de métodos de aprendizagem em árvores, por meio do bootstrap dos dados de treinamento, aumentando o uso do algoritmo conhecido como CART (classification e Regression Trees).

Na grande maioria, os métodos de decisão em árvore possuem viés baixo, mas alta variância, resultando na produção de um sobre ajuste (overfitting). Utilizando RF o problema da variância é tratado usando um número n de árvores de modo que ao produzir n observações independentes $f_1, f_2, ..., f_n$, uma para cada nó final de árvores, a variância diminui através da média dessas n observações. Isso porque a variância individual de cada árvore é maior do que a variância da média delas.

O termo floresta é utilizado, pois a técnica cria uma combinação (ensemble) de árvores de decisão, que na maioria das vezes são treinadas com o método de bagging. Este método possui como fundamento a ideia que a combinação dos modelos de aprendizado aumenta o resultado geral. Em uma definição resumida, o algoritmo de florestas aleatórias gera várias árvores de decisão e as combina para conseguir uma predição com maior acurácia e mais estabilidade.

A precisão do classificador de RF é muito boa, possui forte capacidade de generalização, tem alta velocidade computacional e os parâmetros configuráveis são muito poucos. Em particular, o RF é apropriado para calcular a função não linear de variáveis e pode refletir a interação entre variáveis.

O processo de treinamento do algoritmo RF é baseado no método de bagging. No seu funcionamento, Considerando um conjunto de dados D, com d tuplas. A cada iteração i, onde $(i = 1, 2, ..., k)$ um conjunto de treinamento Di de d tuplas é amostrado com elementos do conjunto de dados D. Isto é, do conjunto de dados iniciais, k amostras são formadas com instâncias e atributos escolhidos de forma aleatória. Assim, cada conjunto D_i passa pelo processo de treinamento de uma árvore de decisão. Através da média do resultado de todas as árvores o resultado final é obtido.

4. Resultados

Nesta seção serão exibidos os resultados obtidos a partir da aplicação dos algoritmos de Machine Learning, desde o seu treino até seu teste.

4.1. Aplicação dos Algoritmos de Machine Learning

Conforme visto na seção anterior, quatro algoritmos de Machine Learning foram aplicados. Uma hiper parametrização foi feita para cada um dos classificadores com uma validação cruzada (tipo k-fold com k valendo 10) nos dados de treino. Para avaliar a hiper parametrização, são utilizados os valores de acurácia, o score F1 e a curva ROC. O score F1 foi utilizado para avaliar a distrição de classes entre os conjuntos e a curva ROC para avaliar a apropriabilidade do classificador. Neste sentido, a função GridSearchCV [11] foi utilizada para exaustivamente procurar dentro de um range definido valores de hiper parametrização com o propósito de encontrar o melhor conjunto de dados para cada algoritmo. Por fim, os classificadores com os parâmetros escolhidos são avaliados com o conjunto de testes.

4.1.1. Decision Tree

Para mensurar a impuridade de um nodo em uma árvore, geralmente são utilizados os parâmetros Gini Index ou a sua entropia. O max_depth, ou seja, parâmetro que confere o maior caminho da raiz até um nodo na árvore foi escolhido para hiper parametrização. Utilizando a função GridSearchCV, foi avaliado em um range de 1 a 30 em conjunto com os parâmetros Gini e entropia os melhores parâmetros, utilizando a acurácia, o score F1 e a curva ROC como critério de escolha. É possível ver na figura 4 o melhor critério escolhido pelo GridSearchCV.

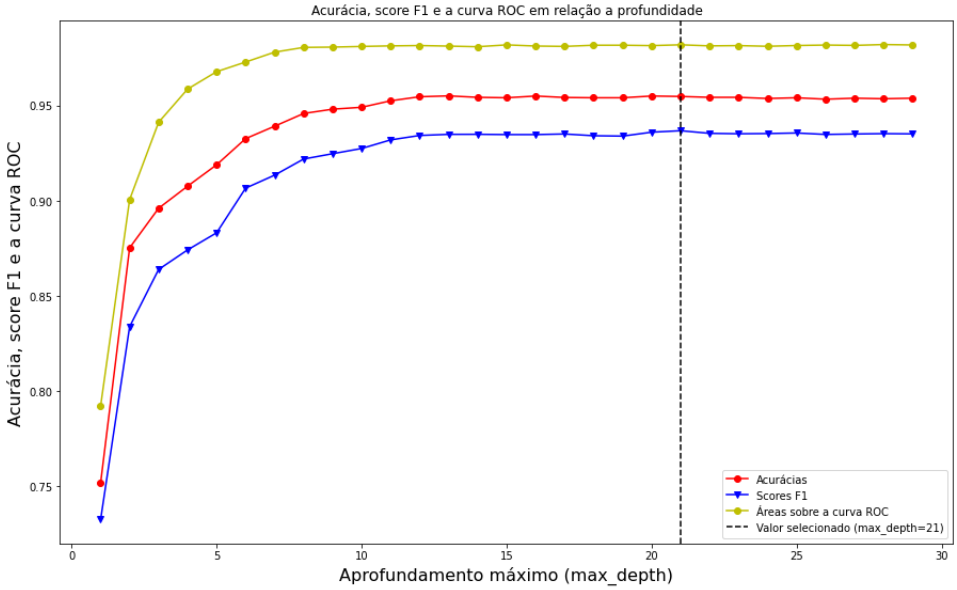

Figura 4: Valores escolhidos pelo GridSearchCV para a Decision Tree

4.1.2. K-Nearest Neighbour (KNN)

Para os parâmetros do KNN, foi utilizada a distância Minkowski com p=2 e o número de vizinhos k. A função GridSearchCV também foi utilizada para medir a hiper parametrização do algoritmo. Os resultados podem ser conferidos na figura 5.

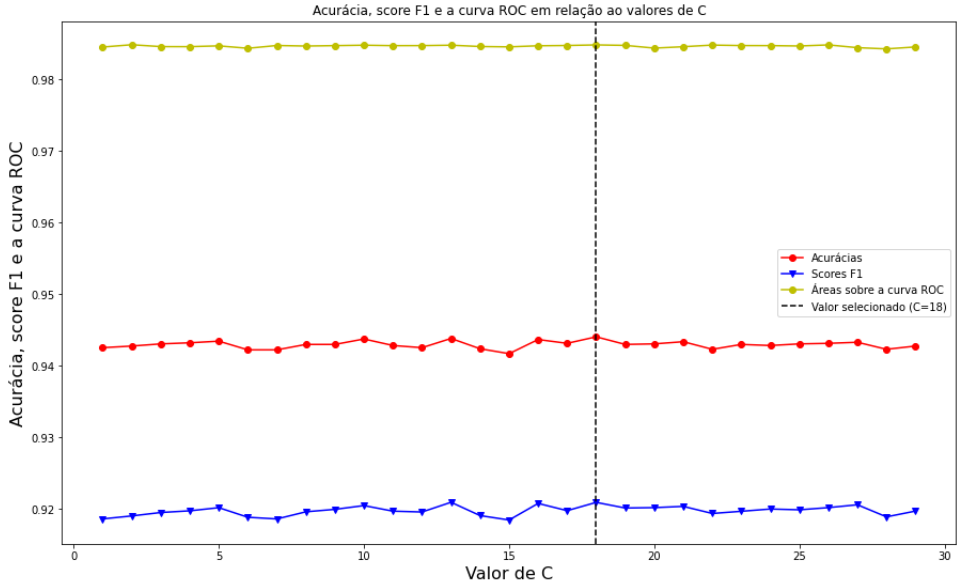

Figura 5: Valores escolhidos pelo GridSearchCV para o KNN

4.1.3. Support Vector Machine (SVM)

A hiper parametrização utilizada para o SVM é o valor de C. Um valor alto de C significa menores violações de margem com uma violação pequena. Todavia, um pequeno valor de C significa uma maior margem (significando que o modelo é mais propenso a ser generalizado) com uma violação maior. Os resultados obtidos da função podem ser conferidos na figura 6.

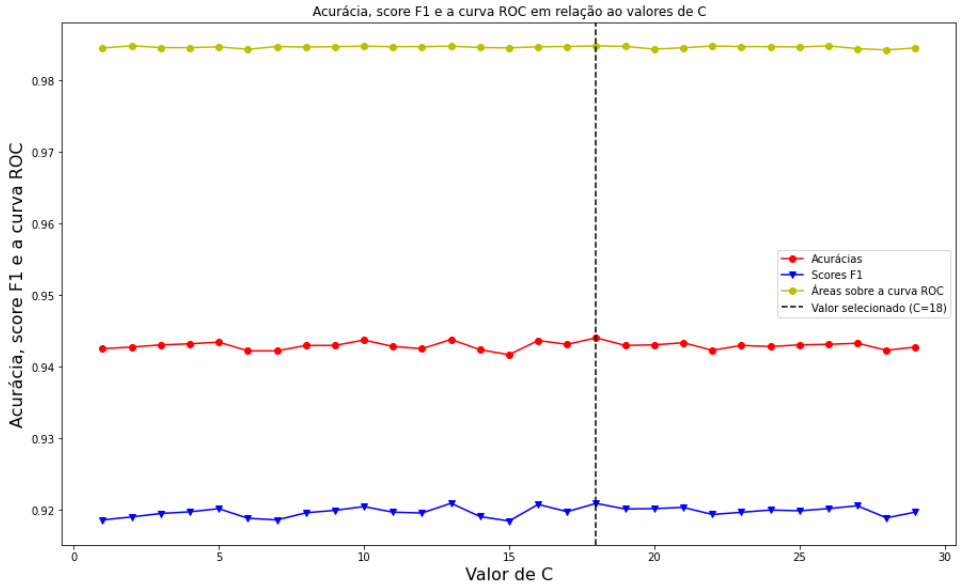

Figura 6: Valores escolhidos pelo GridSearchCV para o SVM

4.1.4. Random Forest

A hiper parametrização para este algoritmo é o n_estimators, ou seja, o número de árvores na floresta, e o max_features, ou seja, o número de features considerando a melhor divisão. Os resultados obtidos da função podem ser conferidos na figura 7.

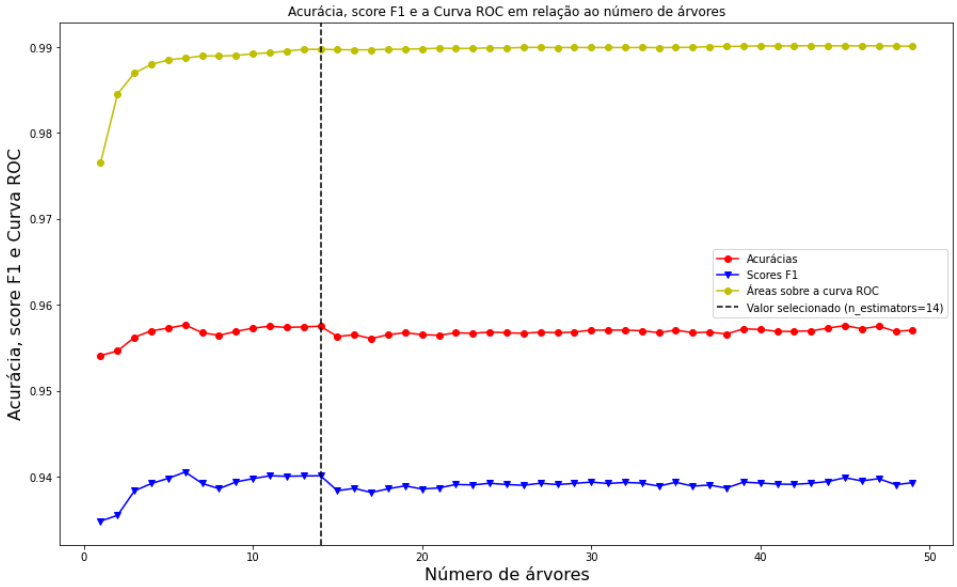

Figura 7: Valores escolhidos pelo GridSearchCV para o Random Forest

5. Comparação dos resultados

As tabelas 2 e 3 abaixo mostram os resultados computados comparando cada resultado dos algoritmos, utilizando para tal primeiramente o dataset com as 207 features e depois o dataset com as 27 features selecionadas como mais importantes.

	Acurácia	**Score F1**	**Curva ROC**
Decision Tree	**0,763**	0,758	0,795
KNN	0,742	0,741	0,786
SVM	0,751	0,743	0,777
Random Forest	0,762	**0,767**	**0,789**

Tabela 2: Resultados de treino de cada algoritmo sem pré-processamento

	Acurácia	**Score F1**	**Curva ROC**
Decision Tree	0,744	0,768	0,713
KNN	0,798	0,791	0,794
SVM	0,781	0,738	0,801
Random Forest	**0,803**	**0,793**	**0,812**

Tabela 3: Resultados de teste de cada algoritmo sem pré-processamento

Percebe-se que, deste conjunto de dados, o algoritmo Random Forest se mostrou a melhor opção em duas métricas, porém a acurácia foi maior durante o treino dos algoritmos. Quando aplicados os dados de teste, o algoritmo Random Forest se mostrou a melhor opção nas três métricas utilizadas.

As tabelas 4 e 5 abaixo mostram os resultados com a seleção de features.

	Acurácia	**Score F1**	**Curva ROC**
Decision Tree	**0,965**	**0,951**	**0,995**
KNN	0,956	0,938	0,986
SVM	0,946	0,923	0,985
Random Forest	**0,965**	**0,951**	0,994

Tabela 4: Resultados de treino de cada algoritmo com pré-processamento

	Acurácia	**Score F1**	**Curva ROC**
Decision Tree	**0,965**	**0,951**	**0,994**
KNN	0,956	0,938	0,986
SVM	0,945	0,924	0,984
Random Forest	0,962	0,948	0,991

Tabela 5: Resultados de teste de cada algoritmo com pré-processamento

Pode-se constatar, portanto que, quando selecionados alguns parâmetros com maior importância, a acurácia do modelo cresce de modo a tornar a classificação mais precisa. O algoritmo Decision Tree se mostrou a melhor opção em ambos os casos de treino e teste. Todavia, o algoritmo Random Forest se mostrou semelhante em dois dos scores, sendo uma alternativa ao Decision Tree.

Além destes dados, outros gráficos foram estudados com o propósito de demonstrar os resultados aqui explorados. Estes gráficos e o código fonte do desenvolvimento deste trabalho pode ser visto no Jupyter Notebook contido no apêndice A deste trabalho. A explicação do código também pode ser encontrada nele.

A realização destes testes implica qual modelo foi o que possuiu a menor acurácia e explica o seu porque, enquanto a aplicação do pré-processamento dos dados aumentou a acurácia. Percebe-se que cada pequeno refinamento reflete diretamente nos resultados finais do classificador. Por vezes é necessário escolher a técnica adequada para o aperfeiçoamento destes dados. Isso pode ser constatado realizando testes comparativos a cada mudança ou aperfeiçoamento.

6. Considerações Finais

O grande problema de toda aplicação de algoritmos de Machine Learning é saber escolher o algoritmo adequado para a resolução de problema difíceis. Neste sentido, o estudo comparativo de algoritmos nos problemas, além do conhecimento de como esses algoritmos funcionam fazem com que o senso crítico do desenvolvedor fique mais apurado e permite

que ele faça a escolha de um algoritmo para um determinado problema de maneira mais fácil.

Ademais, o uso do algoritmo correto para o problema pode fazer com que o desempenho em termos de poder computacional, uso de memória e tempo sejam reduzidos, tornando a aplicação mais robusta e estável.

Outro ponto constatado é que, a depender do número de dados contidos no dataset, o treino e teste destes dados pode levar um tempo maior do que deveria. Neste caso, o algoritmo não se mostra a melhor solução para o problema abordado.

Neste artigo, foi feito um estudo avaliando algoritmos de Machine Learning no processo completo de um treinamento, desde o pré-processamento dos dados até a avaliação dos modelos gerados. Os resultados mostraram que, para um conjunto de dados específico e pré-processado, os resultados foram melhores e alguns algoritmos se mostraram melhores opções para a resolução do problema. Todavia, apesar de terem resultados semelhantes, os tempos para computar estes algoritmos variam e podem ser levados em conta na decisão.

6.1. Trabalhos Futuros

Como trabalhos futuros, pode-se salientar a comparação com outros parâmetros que não os selecionados pela função e seus respectivos resultados.

Outro ponto a ser considerado é uma busca por trabalhos semelhantes na área a título de comparação, além de conhecer novas abordagens para a resolução do problema.

Por fim, a construção de um analisador estático que possa analisar todas as chamadas feitas por um APK sem que seja necessário o uso de um poder computacional grande, ou mesmo utilize muitos recursos energéticos do dispositivo, se fazem uma excelente opção para testar tais classificadores.

Referências

[1] Y. Aafer, W. Du, and H. Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *International conference on security and privacy in communication systems*, pages 86–103. Springer, 2013.

[2] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. *IEEE Access*, 6:33789–33795, 2018.

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

[4] L. G. Castanheira. Aplicação de técnicas de mineração de dados em problemas de classificação de padrões. Sept. 2008.

[5] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, and C. STEIN. *Algoritmos: Teoria e Prática 3ª Edição*. Elsevier Editora Ltda, Rio de Janeiro, RJ, 2012.

[6] V. Gevert, A. L. Silva, F. Gevert, and V. T. Ales. Modelos de regressão logística, redes neurais e support vector machine (svm´s) na análise de crédito a pessoas jurídicas. 2010.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[8] M. A. Hall. Correlation-based feature selection for machine learning. 1999.

[9] A. Moreira, F. Hiago Souza Fernandes, R. Almeida, and C. Nery. O algoritmo support vector machine aplicado ao mapeamento do uso e ocupação do solo (the support vector machine algorithm applied to mapping and land use). *Revista Brasileira de Geografia Física*, 07:291–303, 02 2014.

[10] P. NORVIG and S. RUSSEL. *Inteligência Artificial 3ª Edição*. Elsevier Editora Ltda, Rio de Janeiro, RJ, 2013.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[12] N. POPAL and R. REITH. IDC - Smartphone Market Share - Market Share.

[13] R. Samani. Mcafee mobile threat report q1, 2020. Technical report, 2821 Mission College Blvd., Santa Clara, CA 95054, 2020.

[14] S. Y. Yerima and S. Sezer. Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE transactions on cybernetics*, 49(2):453–466, 2018.

[15] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*, pages 95–109. IEEE, 2012.

Apêndice A Código Fonte

O código fonte desenvolvido durante a concepção deste estudo está disponível em *www.github.com/lucaspeferreira*.