DREBIN Dataset Classification: A Machine Learning Empirical Evaluation

***Abstract*—**
***Index Terms*—**

I. Introdução

II. Metodologia

A. Dataset

O banco de dados DREBIN [1], derivado do método de mesmo nome, foi desenvolvido para pesquisadores interessados em estudar detecção de *malware* e comparar diferentes abordagens. O *dataset* é composto por um total de 131.611 instâncias, que representam uma variedade de aplicativos reais coletados de diferentes mercados e fontes. Dos aplicativos coletados, 5.560 são classificados como *malware*, enquanto os restantes não representam perigo para o usuário, ou seja, são considerados benignos.

Cada instância do banco de dados é composta pelas características extraídas dos aplicativos durante a análise realizada pelo método DREBIN. Esse método extrai uma variedade de informações do código e manifesto do aplicativo para criar um conjunto abrangente de *features*, são elas:

- **Componentes de Hardware:** Refere-se aos componentes de hardware que o aplicativo utiliza ou interage.
- **Permissões Solicitadas:** Indica as permissões que o aplicativo solicita ao usuário para acessar recursos do dispositivo.
- **Componentes do Aplicativo:** Referem-se aos diferentes componentes do aplicativo, como atividades, serviços, receptores de transmissão e provedores de conteúdo.
- **Intenções Filtradas:** Representa as intenções que são filtradas ou manipuladas pelo aplicativo.
- **Chamadas de API Restritas:** Indica as chamadas de API que estão restritas pelo sistema Android devido a questões de segurança e privacidade.
- **Permissões Utilizadas:** Refere-se às permissões que o aplicativo realmente utiliza durante sua execução.
- **Chamadas de API Suspeitas:** Representa chamadas de API que são consideradas suspeitas com base em padrões de comportamento de *malware* conhecidos.
- **Endereços de Rede:** Indica os endereços de rede com os quais o aplicativo se comunica durante sua execução.

Essas características são organizadas em oito conjuntos principais de *features* e, em seguida, mapeadas para um espaço vetorial, no qual padrões e combinações podem ser analisados geometricamente. Cada grupo de características possui uma quantidade diferente de *features*, que depende do propósito da aplicação. Por exemplo, uma aplicação de música não solicita ou pelo menos não deveria solicitar acesso à câmera. As *features* são representadas de forma quantitativa, identificando a frequência com que cada característica aparece no aplicativo. O vetor de características de uma instância contém contagens que indicam quantas vezes cada conjunto de características foi identificado na instância do aplicativo analisado, permitindo a detecção de padrões associados a comportamento malicioso.

Os dados, amostras de aplicativos e todos os conjuntos de características extraídas, podem ser acessados através do link fornecido pelos autores do estudo [3].

TABLE I
Parâmetros dos Classificadores

Classificadores		
Decision Tree	Altura maxima = Unlimited	Critério = Gini Impurity
Naive Bayes	Algoritmo: Gaussiano	
KNN	k = 7	
MLP	Hidden layers size = (100)	Max iterations: 1000

B. Modelos

Neste estudo, conduziremos uma análise empírica na área de classificação binária para a detecção de *malwares*. Faremos uso de algoritmos de classificação monolíticos, incluindo Decision Tree, Naive Bayes, KNN e MLP, conforme detalhado na Tabela 1. Além disso, para explorar estratégias de *ensemble* e aumentar a robustez do estudo, aplicaremos modelos de *Bagging*. Esses modelos criam múltiplos conjuntos de dados de treinamento por meio de amostragem aleatória com substituição e combinam as previsões dos modelos individuais para melhorar a estabilidade e precisão das previsões.

C. Métricas de Avaliação

Como estamos lidando com classificadores monolíticos, que tratam todo o conjunto de dados de uma vez, devemos escolher métricas que forneçam uma visão abrangente do desempenho do modelo nas duas classes, especialmente quando há um desequilíbrio significativo entre elas [7]. Por isso, utilizaremos acurácia, precisão, recall, F1 score e área sob a curva ROC para avaliação.

1) Acurácia:

$$\text{Acurácia} = \frac{\text{Número de previsões corretas}}{\text{Número total de previsões}} \tag{1}$$

A acurácia é uma métrica básica que mede a proporção de previsões corretas em relação ao total de previsões feitas pelo modelo. No entanto, em conjuntos de dados desbalanceados, a acurácia pode ser enganosa, pois um modelo pode ser

altamente preciso para a classe majoritária, mas ignorar completamente as classes minoritárias.Para mitigar esse problema, adotaremos também as outras métricas.

2) Precisão:

$$\text{Precisão} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos positivos}} \quad (2)$$

A precisão mede a proporção de verdadeiros positivos (previsões corretas da classe positiva) em relação a todas as previsões positivas feitas pelo modelo.

3) Recall:

$$\text{Recall} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos negativos}} \quad (3)$$

O recall mede a proporção de verdadeiros positivos em relação a todas as instâncias que realmente pertencem à classe positiva no conjunto de dados.

4) F1 Score:

$$\text{F1 Score} = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}} \quad (4)$$

O F1 Score é a média harmônica entre precisão e recall e fornece uma medida única do desempenho do modelo, levando em consideração tanto falsos positivos quanto falsos negativos.

5) Área sob a curva ROC (AUC ROC): A área sob a curva ROC (Receiver Operating Characteristic) é uma métrica que avalia a capacidade do modelo de distinguir entre as classes positiva e negativa, independentemente do limiar de classificação. A ROC é particularmente útil em situações de desbalanceamento de classes, pois não é afetada pela distribuição de classes no conjunto de dados.

D. Balanceamento dos Dados

Dado o desbalanceamento das classes no conjunto de dados DREBIN e a propensão dos modelos tradicionais de aprendizado de máquina a apresentarem baixa capacidade de generalização devido ao favorecimento da predição da classe majoritária, os modelos mencionados serão avaliados tanto antes quanto após o balanceamento dos dados. Após analisar o banco de dados, percebemos que a quantidade de *features* benignas é de aproximadamente 96%, enquanto as *features* malignas ocupam os 4% restantes. Devido a essa discrepância no balanceamento das classes, decidimos utilizar a técnica SMOTE (*Synthetic Minority Over-sampling Technique*) para o balanceamento. Esta técnica foi utilizada para criar artificialmente novos dados para a classe minoritária, a partir daqueles já presentes no conjunto de dados, e adicioná-los ao conjunto original. Esperamos, dessa forma, evitar que a classe majoritária seja favorecida e prevenir um possível *overfitting* da classe minoritária que poderia ocorrer caso apenas duplicássemos esses dados.

E. Ambiente de Experimentação

Nosso experimento foi conduzido com os seguintes recursos:

- **Hardware**:
  - Processador: AMD Ryzen 7-5800H
  - Placa de Video: NVidia GeForce RTX 3050 4GB
  - Memoria: 16GB
- **Software**:
  - Sistema operacional: Windows 11
  - Bibliotecas:
    - Numpy
    - Sklearn
    - Pandas

TABLE II
PARÂMETROS DO BALANCEAMENTO

Algoritmo		
SMOTE	Sampling Strategy = Auto	K-Neighbors = 5

III. RESULTADOS

- **Protocolo Experimental 1**: Para aprimorar a avaliação dos classificadores monolíticos, optamos por adotar uma abordagem estratificada de hold-out, repetida 30 vezes. Neste procedimento, o conjunto de dados foi dividido na proporção de 80/20, garantindo a representatividade de cada classe em cada iteração. Além disso, implementamos um sistema de *ensemble* para cada classificador, utilizando 30 estimadores em cada técnica de *Bagging*, para garantir uma estimativa mais estável que pode ser comparável, em termos de variabilidade, aos resultados obtidos pela estratégia monolítica. Assim, poderemos realizar uma análise estatística abrangente e robusta dos resultados obtidos.

Ao término da primeira etapa da experimentação, feita com dados desbalanceados, obtivemos os resultados apresentados na Table 2 para os classificadores monolíticos e os resultados da Table 3 para os classificadores de *ensemble*.

- **Protocolo Experimental 2**: Após os experimentos descritos no Protocolo Experimental 1, balanceamos os dados utilizando a tecnica SMOTE e seguimos com a mesma abordagem estratificada de hold-out, repetida por 30 vezes. Onde o conjunto de dados foi dividido numa proporção de 80/20, e continuamos implementando um sistema de *ensemble* para cada classificador, utilizando 30 estimadores em cada técnica de *Bagging*.

Ao término da segunda etapa da experimentação, feita com dados balanceados, obtivemos os resultados apresentados na Table 4 para os classificadores monolíticos e os resultados da Table 5 para os classificadores de *ensemble*.

IV. COMPARAÇÃO COM TRABALHOS RELACIONADOS

V. CONCLUSÕES E DISCUSSÕES

REFERENCES

[1] Daniel Arp et Al. "Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket", 21th Annual Network and Distributed System Security Symposium (NDSS), February 2014

[2] Michael Spreitzenbarth et Al. "MobileSandbox: Looking Deeper into Android Applications", 28th International ACM Symposium on Applied Computing (SAC), March 2013

[3] https://drebin.mlsec.org/
[4] Luo Shi-qi et Al. "Deep Learning in Drebin: Android malware Image Texture Median Filter Analysis and Detection," KSII Transactions on Internet and Information Systems, vol. 13, no. 7. Korean Society for Internet Information (KSII), 31-Jul-2019.
[5] Chenglin Li et Al. "Android Malware Detection Based on Factorization Machine" in IEEE Access, vol. 7, pp. 184008-184019, 2019
[6] V. Kouliaridis et Al. "Feature Importance in Android Malware Detection," 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 2020, pp. 1449-1454
[7] De Diego et al. General Performance Score for classification problems. Appl Intell 52, 12049–12063 (2022)

TABLE III
RESULTADOS DO EXPERIMENTO 1 - DADOS DESBALANCEADOS

Classificadores	Acurácia	Precisão	Recall	F1	AUC ROC
Decision Tree	**0.9817 ± (0.0007)**	0.7713 ± (0.0106)	**0.8194 ± (0.0128)**	**0.7945 ± (0.0077)**	**0.9042 ± (0.0063)**
Naive Bayes	0.9163 ± (0.0030)	0.2501 ± (0.0098)	0.4704 ± (0.0149)	0.3264 ± (0.0097)	0.7034 ± 0.(0070)
MLP	0.9769 ± (0.0011)	**0.8174 ± (0.0327)**	0.6016 ± (0.0282)	0.6920 ± (0.0149)	0.7977 ± (0.0136)
KNN	0.9814 ± (0.0007)	0.8136 ± (0.0114)	0.7383 ± (0.0134)	0.7740 ± (0.0089)	0.8653 ± (0.0066)

TABLE IV
RESULTADOS DO EXPERIMENTO 1 - DADOS DESBALANCEADOS E BAGGING

Classificadores	Acurácia	Precisão	Recall	F1	AUC ROC
Decision Tree Bagging	**0.9891**	**0.9037**	**0.8354**	**0.8682**	**0.9157**
Naive Bayes Bagging	0.9178	0.2486	0.4487	0.3200	0.6938
MLP Bagging	0.9795	0.8527	0.6349	0.7278	0.8150
KNN Bagging	0.9823	0.8183	0.7572	0.7865	0.8748

TABLE V
RESULTADOS DO EXPERIMENTO 1 - DADOS BALANCEADOS

Classificadores	Acurácia	Precisão	Recall	F1	AUC ROC
Decision Tree	**0.9812 ± (0.0006)**	**0.9737 ± (0.0009)**	0.9891 ± (0.0011)	**0.9813 ± (0.0006)**	**0.9812 ± (0.0006)**
Naive Bayes	0.7299 ± (0.0046)	0.7255 ± (0.0047)	0.7397 ± (0.0141)	0.7325 ± (0.0067)	0.7299 ± (0.0046)
MLP	0.9337 ± (0.0025)	0.9319 ± (0.0119)	0.9361 ± (0.0147)	0.9338 ± (0.0028)	0.9337 ± (0.0025)
KNN	0.9707 ± (0.0009)	0.9516 ± (0.0016)	**0.9919 ± (0.0013)**	0.9713 ± (0.0009)	0.9707 ± (0.0009)

TABLE VI
RESULTADOS DO EXPERIMENTO 1 - DADOS BALANCEADOS E BAGGING

Classificadores	Acurácia	Precisão	Recall	F1	AUC ROC
Decision Tree Bagging	**0.9853**	**0.9806**	0.9903	**0.9854**	**0.9853**
Naive Bayes Bagging	0.7308	0.7254	0.7428	0.7340	0.7308
MLP Bagging	0.9437	0.9393	0.9488	0.9440	0.9437
KNN Bagging	0.9720	0.9544	**0.9913**	0.9725	0.9720