DREBIN Dataset Classification: A Machine Learning Empirical Evaluation

***Abstract*—**
***Index Terms*—**

I. Introdução

Nos últimos anos, a segurança da informação tornou-se uma preocupação central em diversas esferas, abrangendo empresas, governos e indivíduos. O crescente número de ataques cibernéticos e violações de dados ressalta a necessidade de desenvolver métodos eficazes para a detecção e prevenção de ameaças. Nesse contexto, técnicas de aprendizagem de máquina têm se mostrado promissoras, ao proporcionar soluções capazes de identificar padrões e anomalias em grandes volumes de dados de forma automática e eficiente.

Este estudo tem como objetivo apresentar uma análise empírica de algoritmos de aprendizagem de máquina aplicados à detecção de malware em smartphones, com foco específico em dispositivos Android. Malware, ou software malicioso, refere-se a qualquer programa ou código desenvolvido com a intenção de causar danos a um dispositivo, servidor ou rede. No contexto de smartphones, malwares podem roubar informações pessoais, rastrear a localização do usuário, exibir anúncios intrusivos ou até mesmo controlar o dispositivo remotamente.

As abordagens para análise de malware podem ser classificadas em três categorias principais: estática, dinâmica e híbrida. A análise estática envolve a inspeção do código-fonte ou do binário do malware sem a necessidade de sua execução, buscando assinaturas conhecidas, padrões de código e outras características que possam indicar sua natureza maliciosa. A análise dinâmica, por outro lado, observa o comportamento do malware em execução, monitorando sua interação com o sistema e a rede para identificar atividades suspeitas. Já a análise híbrida combina ambas as abordagens, proporcionando uma visão mais completa e detalhada do malware.

Para conduzir esta pesquisa, utilizamos o *Drebin dataset*, um conjunto de dados robusto e amplamente utilizado para a detecção de malware em dispositivos Android. Este conjunto de dados contém diversas variáveis relacionadas à segurança, como logs de acesso, registros de atividades de rede e indicadores de intrusão, permitindo uma análise abrangente do comportamento de diferentes tipos de malware. Para garantir uma análise completa, utilizamos todas as instâncias do *Drebin dataset*. Adicionalmente, a fim de lidar com o desbalanceamento das classes no conjunto de dados, empregamos a técnica SMOTE (*Synthetic Minority Over-sampling Technique*) na segunda fase dos experimentos, com o intuito de balancear as classes e melhorar o desempenho dos algoritmos de aprendizagem de máquina.

Nossa análise inclui a aplicação de algoritmos de aprendizagem monolíticos, como *Multi-Layer Perceptron* (MLP), *k-Nearest Neighbors* (KNN), *Decision Tree* e *Naive Bayes*. Além disso, exploramos algoritmos de *bagging*, utilizando os mesmos algoritmos monolíticos, e algoritmos de seleção dinâmica, incluindo *Gradient Boosted Decision Trees* (GBDT), *Random Forest*, *KNOP*, *METADES*, *OLA*, *Single Best* e *Static Selection*. Ao comparar esses métodos, pretendemos identificar quais abordagens se mostram mais eficazes na detecção de malware em dispositivos Android. Também realizaremos uma comparação do impacto do balanceamento no desempenho dos algoritmos, e verificaremos quais métricas são mais relevantes para avaliar o problema, como acurácia, precisão, *recall*, e a área sob a curva ROC.

II. Comparação com Trabalhos Relacionados

Nos últimos anos, o *Drebin dataset* [1] tem sido amplamente utilizado na pesquisa de segurança em dispositivos móveis, especialmente no contexto da detecção de *malware* em sistemas Android. Diversos estudos exploraram diferentes abordagens para classificar aplicativos maliciosos e benignos, utilizando este conjunto de dados como referência, devido à sua ampla variedade de características e ao número significativo de amostras de *malware*.

A. Abordagens Baseadas em Machine Learning

Um dos estudos pioneiros que utilizaram o *Drebin dataset* foi realizado por Arp et al. [1], os criadores do próprio conjunto de dados. Neste trabalho, os autores propuseram um método que combina análise estática com aprendizado de máquina para detectar *malware* em Android. Eles utilizaram um classificador baseado em *Support Vector Machines* (SVM) e alcançaram uma taxa de detecção elevada, demonstrando a eficácia de técnicas de aprendizado supervisionado na identificação de *malware*.

Mais recentemente, Wang et al. [9] exploraram o uso de redes neurais profundas para a detecção de *malware* em dispositivos Android. O estudo demonstrou que, ao combinar técnicas de *deep learning* com características extraídas do *Drebin dataset*, é possível melhorar significativamente a precisão da detecção em comparação com abordagens mais tradicionais.

B. Técnicas de Ensemble e Métodos Híbridos

Outros estudos têm focado na aplicação de métodos de *ensemble* e abordagens híbridas para melhorar o desempenho

dos classificadores. Por exemplo, Suarez-Tangil et al. [10] propuseram um sistema de detecção que combina técnicas de *bagging* e *boosting*, aplicando esses métodos sobre classificadores monolíticos, como *Decision Trees* e *SVM*. Utilizando o *Drebin dataset*, os autores demonstraram que técnicas de *ensemble* podem aumentar a robustez do modelo contra variabilidades no conjunto de dados.

Além disso, Zhou et al. [11] investigaram a aplicação de métodos híbridos que combinam análise estática e dinâmica para a detecção de *malware*. Eles mostraram que a combinação de diferentes técnicas de análise pode resultar em uma maior taxa de detecção, especialmente quando aplicada a grandes conjuntos de dados, como o *Drebin*.

C. Desafios e Limitações

Embora o *Drebin dataset* tenha sido crucial para avanços na detecção de *malware* em Android, alguns estudos apontam suas limitações. Por exemplo,Suarez-Tangil et al. [10] destacam que o conjunto de dados, embora extenso, pode não representar adequadamente a diversidade das ameaças atuais, especialmente considerando o rápido desenvolvimento de novas variantes de *malware*, o que pode limitar a generalização dos modelos treinados. Além disso, é necessario discutir a importância de balancear o conjunto de dados para evitar o viés nos modelos de aprendizado de máquina, uma vez que o *Drebin* contém uma quantidade significativamente menor de amostras de *malware* em comparação com amostras benignas.

D. Contribuições do Presente Estudo

O presente estudo busca expandir o conhecimento existente utilizando o *Drebin dataset*, mas com um foco em técnicas de balanceamento de dados e seleção dinâmica de modelos, que têm sido subexploradas na literatura. Ao comparar os resultados obtidos com diferentes técnicas de aprendizado de máquina, este trabalho pretende oferecer uma análise mais aprofundada sobre como o balanceamento do conjunto de dados pode impactar o desempenho de modelos de detecção de *malware*.

III. 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. Essas características capturam aspectos específicos do comportamento e da estrutura dos aplicativos e são representadas de forma quantitativa, indicando a frequência com que cada característica aparece. O método DREBIN retira essas informações do código e do manifesto do aplicativo, organizando-as nas oito seguintes categorias:

- **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.

Cada uma das oito posições no vetor de características corresponde a uma dessas categorias e o valor nessa posição indica a quantidade de características presentes pertencentes a esse grupo.

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

B. Modelos

Neste estudo, conduzimos uma análise empírica na área de classificação binária para a detecção de malwares. Utilizamos algoritmos de classificação monolíticos, incluindo Decision Tree, Naive Bayes, KNN e MLP, conforme detalhado na Tabela 1. Também empregamos algoritmos de seleção dinâmica, como SingleBest, StaticSelection, OLA, KNOP e METADES, com o estimador base sendo a Random Forest.

Para explorar estratégias de ensemble e aumentar a robustez do estudo, aplicamos 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, melhorando a estabilidade e a 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 desbalenceamento significativo entre elas [7]. Por isso, além da acurácia, utilizaremos também precisão, recall, F1 score e área sob a curva ROC para avaliação.

TABLE I
PARÂMETROS DOS CLASSIFICADORES

Classificadores	Parâmetros
Decision Tree	Critério = Gini
MLP	Hidden layers size = (100), Max iterations = 1000
KNN	k = 7
Naive Bayes	Algoritmo = Gaussiano
Bagging Decision Tree	Estimador = DecisionTree, n_estimators = 30
Bagging Naive Bayes	Estimador = NaiveBayes, n_estimators = 30
Bagging KNN	Estimador = KNN, n_estimators = 30
Bagging MLP	Estimador = MLP, n_estimators = 30
Gradient Boosted Decision Tree	n_estimators = 100
Random Forest	n_estimators = 100

TABLE II
PARÂMETROS DO BALANCEAMENTO

Algoritmo		
SMOTE	Sampling Strategy = Auto	K-Neighbors = 5

1) Acurácia:

$$\text{Acurácia} = \frac{\text{Número de previsões corretas}}{\text{Número total de previsões}} \quad (1)$$

A acurácia 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.

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 são avaliados tanto antes quanto após o balanceamento dos dados. Após analisar o banco de dados, percebemos que a quantidade de instâncias da classe benignas é de aproximadamente 96%, enquanto as instâncias da classe malignas representam os 4% restantes. Devido a essa discrepância no balanceamento das classes, decidimos utilizar a técnica SMOTE (*Synthetic Minority Oversampling 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. Espera-se, 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

O 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
    - DesLib

IV. RESULTADOS

- **Protocolo Experimental 1**: Para aprimorar a avaliação dos classificadores, 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. Implementamos um sistema de *ensemble* para cada classificador, utilizando 30 estimadores em cada técnica de *Bagging*, para garantir uma estimativa mais estável, comparável, em termos de variabilidade, aos resultados obtidos pela estratégia monolítica. Além disso, aplicamos técnicas de seleção dinâmica à random forest, repetidas também 30 vezes. Dessa forma, pudemos realizar uma análise estatística abrangente e robusta dos resultados obtidos.

Após a primeira etapa da experimentação com dados desbalanceados, observou-se que os modelos METADES, RandomForest e StaticSelection apresentaram as melhores performances globais. Modelos baseados em *bagging*, como BaggingDecisionTree e BaggingKnn, demonstraram melhorias

em relação aos seus equivalentes não-*bagging*. Em contraste, os modelos NaiveBayes e GradientBoostedDecisionTree apresentaram desempenhos relativamente inferiores.

A superioridade dos modelos METADES, RandomForest e StaticSelection pode ser atribuída à capacidade desses algoritmos de combinar decisões de múltiplos classificadores, aumentando, assim, a robustez e a precisão dos resultados. Especificamente, os modelos RandomForest e StaticSelection se beneficiam da agregação de árvores de decisão, o que contribui para a redução do *overfitting* e para a melhoria da generalização dos modelos. Por sua vez, o modelo METADES utiliza estratégias de seleção dinâmica, ajustando-se de forma mais eficaz aos padrões dos dados.

Os modelos BaggingDecisionTree e BaggingKnn demonstraram que a técnica de *bagging* pode melhorar o desempenho de modelos individuais ao reduzir a variância, característica que é especialmente relevante em conjuntos de dados desbalanceados, nos quais a variância pode resultar em classificações errôneas.

A baixa performance do modelo GradientBoostedDecisionTree na métrica de *recall* sugere que esse modelo pode estar excessivamente focado na identificação da classe majoritária, o que prejudica a correta identificação das classes minoritárias.

- **Protocolo Experimental 2**: Após os experimentos descritos no Protocolo Experimental 1, balanceamos o conjunto de treinamento utilizando a técnica SMOTE e seguimos com a mesma abordagem estratificada de hold-out, repetida 30 vezes. O conjunto de dados foi dividido na proporção de 80/20, garantindo a representatividade de cada classe em cada iteração. Continuamos implementando um sistema de *ensemble* para cada classificador, utilizando 30 estimadores em cada técnica de *Bagging*. Além disso, os classificadores dinâmicos, foram também executados 30 vezes com dados balanceados utilizando SMOTE.

Ao término da segunda etapa da experimentação, realizada com dados balanceados, observou-se que os modelos de *ensemble*, que haviam se destacado na primeira fase da experimentação e consistentemente figuravam entre os três melhores em praticamente todas as métricas, não mantiveram esse desempenho na métrica de *recall* na segunda fase da análise.

A acurácia, métrica que quantifica a proporção de previsões corretas em relação ao total de previsões realizadas, apresentou uma redução geral após o balanceamento dos dados utilizando a técnica SMOTE. Em um cenário desbalanceado, uma alta acurácia pode ser alcançada simplesmente prevendo-se sempre a classe majoritária. No entanto, com a distribuição mais equilibrada imposta pelo SMOTE, os modelos enfrentaram uma maior complexidade na tarefa de classificação, o que resultou em uma diminuição da acurácia.

Em relação à precisão, que avalia a proporção de verdadeiros positivos em relação ao total de positivos preditos, também foi observada uma redução geral. O balanceamento dos dados incentivou os modelos a preverem mais exemplos da classe minoritária, o que aumentou o número de falsos positivos e, consequentemente, reduziu a precisão. Esse efeito foi mais acentuado no modelo *Gradient Boosted Decision Tree*, que apresentou a maior perda nesse aspecto. A redução na precisão, combinada com o aumento no *recall*, impactou o *F1 Score*, uma vez que essa métrica é sensível às variações entre precisão e *recall*. Mesmo com a melhora no *recall*, a queda significativa na precisão resultou em uma diminuição no *F1 Score*, sendo o modelo *Gradient Boosted Decision Tree* o mais afetado.

Por outro lado, a métrica de *recall*, que mede a capacidade do modelo em identificar verdadeiros positivos, apresentou uma melhora global. O balanceamento dos dados permitiu que os modelos identificassem mais exemplos da classe minoritária, elevando o número de verdadeiros positivos e, consequentemente, o *recall*. Adicionalmente, a métrica AUC-ROC, que avalia a capacidade do modelo em distinguir entre as classes, também apresentou melhorias. A maior sensibilidade aos positivos reais, proporcionada pelo balanceamento dos dados, levou a um desempenho superior dos modelos na distinção entre as classes, resultando em um aumento na AUC-ROC.

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", ACM Symposium on Applied Computing (SAC), 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), 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," 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)

[8] Chawla et al. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321-357.

[9] Wang et al. (2019). DroidEnsemble: Detecting Android malicious applications with ensemble of string-based features. Journal of Computer Virology and Hacking Techniques, 15(1), 29-38.

[10] Suarez-Tangil et al. (2017). DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware. In Proceedings of the 7th ACM Conference on Data and Application Security and Privacy (CODASPY) (pp. 309-320). Arizona, USA. DOI: 10.1145/3029806.3029825.

[11] Zhou et al. (2012). Dissecting Android malware: Characterization and evolution. IEEE Symposium on Security and Privacy.

TABLE III
METRICS FOR UNBALANCED MODELS

Modelo	Acurácia	Precisão	Recall	F1_score	Roc_auc
DecisionTree	0.9817 (0.0007)	0.7721 (0.0089)	0.8168 (0.0135)	0.7937 (0.0077)	0.9030 (0.0067)
Knn	0.9811 (0.0006)	0.8106 (0.0094)	0.7339 (0.0107)	0.7703 (0.0080)	0.8631 (0.0053)
MLP	0.9766 (0.0009)	0.8178 (0.0321)	0.5922 (0.0321)	0.6857 (0.0151)	0.7931 (0.0154)
NaiveBayes	0.9175 (0.0034)	0.2515 (0.0111)	0.4612 (0.0115)	0.3253 (0.0095)	0.6996 (0.0053)
BaggingDecisionTree	0.9877 (0.0005)	0.8970 (0.0099)	0.8082 (0.0101)	0.8502 (0.0064)	0.9020 (0.0050)
BaggingKnn	0.9818 (0.0009)	0.8244 (0.0158)	0.7343 (0.0132)	0.7766 (0.0107)	0.8636 (0.0066)
BaggingMLP	0.9786 (0.0008)	0.8530 (0.0093)	0.6074 (0.0195)	0.7093 (0.0135)	0.8013 (0.0097)
BaggingNaiveBayes	0.9178 (0.0034)	0.2529 (0.0115)	0.4625 (0.0115)	0.3268 (0.0107)	0.7004 (0.0058)
GradientBoostedDecisionTree	0.9735 (0.0008)	0.8405 (0.0157)	0.4749 (0.0163)	0.6067 (0.0151)	0.7354 (0.0081)
KNOP	0.9891 (0.0006)	0.9335 (0.0077)	0.8037 (0.0134)	0.8637 (0.0080)	0.9006 (0.0067)
METADES	0.9893 (0.0006)	0.9342 (0.0108)	0.8093 (0.0141)	0.8672 (0.0080)	0.9034 (0.0070)
OLA	0.9825 (0.0007)	0.7865 (0.0104)	0.8154 (0.0140)	0.8006 (0.0076)	0.9027 (0.0068)
RandomForest	0.9890 (0.0005)	0.9346 (0.0094)	0.8018 (0.0119)	0.8630 (0.0070)	0.8996 (0.0059)
SingleBest	0.9804 (0.0007)	0.7631 (0.0121)	0.7913 (0.0128)	0.7769 (0.0080)	0.8901 (0.0063)
StaticSelection	0.9891 (0.0005)	0.9364 (0.0099)	0.8012 (0.0112)	0.8635 (0.0072)	0.8994 (0.0055)

TABLE IV
METRICS FOR BALANCED MODELS

Modelo	Acurácia	Precisão	Recall	F1_score	Roc_auc
DecisionTree	0.9723 (0.0011)	0.6320 (0.0125)	0.8588 (0.0099)	0.7280 (0.0083)	0.9181 (0.0047)
Knn	0.9560 (0.0014)	0.4944 (0.0086)	0.8909 (0.0118)	0.6359 (0.0076)	0.9249 (0.0056)
MLP	0.9296 (0.0118)	0.3736 (0.0390)	0.8891 (0.0211)	0.5244 (0.0356)	0.9103 (0.0057)
NaiveBayes	0.7204 (0.0112)	0.1075 (0.0034)	0.7504 (0.0137)	0.1880 (0.0050)	0.7347 (0.0053)
BaggingDecisionTree	0.9760 (0.0010)	0.6710 (0.0110)	0.8708 (0.0073)	0.7579 (0.0081)	0.9258 (0.0037)
BaggingKnn	0.9496 (0.0011)	0.4574 (0.0056)	0.9078 (0.0090)	0.6083 (0.0052)	0.9296 (0.0042)
BaggingMLP	0.9401 (0.0025)	0.4112 (0.0108)	0.8986 (0.0076)	0.5641 (0.0102)	0.9203 (0.0038)
BaggingNaiveBayes	0.7248 (0.0098)	0.1079 (0.0027)	0.7399 (0.0206)	0.1882 (0.0043)	0.7320 (0.0077)
GradientBoostedDecisionTree	0.8975(0.0018)	0.2782(0.0043)	0.8641(0.0114)	0.4208(0.0058)	0.8815(0.0057)
KNOP	0.9825 (0.0011)	0.7662 (0.0185)	0.8562 (0.0102)	0.8085 (0.0094)	0.9222 (0.0048)
METADES	0.9823 (0.0009)	0.7637 (0.0135)	0.8527 (0.0093)	0.8057 (0.0083)	0.9204 (0.0046)
OLA	0.9685 (0.0013)	0.5914 (0.0118)	0.8706 (0.0100)	0.7042 (0.0091)	0.9217 (0.0049)
RandomForest	0.9804 (0.0010)	0.7280 (0.0136)	0.8711 (0.0105)	0.7930 (0.0090)	0.9282 (0.0052)
SingleBest	0.9634 (0.0012)	0.5479 (0.0099)	0.8651 (0.0087)	0.6708 (0.0083)	0.9165 (0.0044)
StaticSelection	0.9803 (0.0012)	0.7270 (0.0147)	0.8690 (0.0119)	0.7916 (0.0109)	0.9272 (0.0060)

TABLE V
EVOLUÇÃO PÓS-BALANCEAMENTO

Model	Accuracy	Precision	Recall	F1_score	Roc_auc
DecisionTree	-0.009375	-0.140159	0.041966	-0.065747	0.015140
Knn	-0.025122	-0.316129	0.156954	-0.134437	0.061816
MLP	-0.047027	-0.444161	0.296942	-0.161213	0.117212
NaiveBayes	-0.197074	-0.143975	0.289179	-0.137280	0.035103
BaggingDecisionTree	-0.011712	-0.226057	0.062680	-0.092307	0.023809
BaggingKnn	-0.032195	-0.366985	0.173471	-0.168372	0.066007
BaggingMLP	-0.038449	-0.441817	0.291187	-0.145218	0.118946
BaggingNaiveBayes	-0.192956	-0.145017	0.277368	-0.138584	0.031615
GradientBoostedDecisionTree	-0.075990	-0.562332	0.389239	-0.185857	0.146148
KNOP	-0.006564	-0.167350	0.052488	-0.055193	0.021632
METADES	-0.007048	-0.170434	0.043405	-0.061463	0.017042
OLA	-0.014022	-0.195109	0.055156	-0.096351	0.019009
RandomForest	-0.008639	-0.206673	0.069335	-0.070010	0.028592
SingleBest	-0.017008	-0.215252	0.073801	-0.106027	0.026351
StaticSelection	-0.008810	-0.209443	0.067836	-0.071881	0.027787