Avaliação de Ferramentas de AutoML em Datasets de Detecção de Malwares Android

Guilherme Siqueira[1], Diego Kreutz[1], Joner Assolin[2], Estevão Costa[2]
Charles Miers[3], Rodrigo Mansilha[1], Jonas Pontes[2], Eduardo Feitosa[2]

[1]Universidade Federal do Pampa (UNIPAMPA)
{GuilhermeSiqueira.aluno,Kreutz,Mansilha}@unipampa.edu.br

[2]Universidade Federal do Amazonas (UFAM)
{joner.assolin,ecc,pontes,efeitosa}@icomp.ufam.br

[3]Universidade do Estado de Santa Catarina (UDESC)
charles.miers@udesc.br

Abstract. *Developing correct and effective predictive models requires technical knowledge and mastery of the problem, which often does not happen in practice, leading to biased and ineffective solutions. AutoML tools were developed to automate the training of machine learning models. In this paper, we evaluate the performance of four AutoML tools (Auto-Sklearn, AutoGluon, TPOT, QuickAutoML) in generating Android application classifiers taking into account three metrics: accuracy, recall, and execution time. We use seven different datasets to show performance variations amongst the tools.*

Resumo. *O desenvolvimento de modelos preditivos corretos e eficazes requer um conhecimento técnico e do domínio do problema, o que muitas vezes não ocorre na prática, levando a soluções enviesadas e pouco eficazes. Ferramentas de AutoML surgiram com o propósito automatizar as etapas que envolvem o treinamento de modelos de machine learning. Neste contexto, trazemos uma avaliação de desempenho de quatro ferramentas de AutoML (Auto-Sklearn, AutoGluon, TPOT, QuickAutoML) na geração de classificadores de aplicações Android, considerando três métricas: acurácia, revocação e tempo de execução. Utilizamos sete datasets distintos para demonstrar a variação de desempenho entre as ferramentas.*

1. Introdução

Atualmente (2022) a plataforma Android conta com mais de 2,5 bilhões de usuários e 2,9 milhões de aplicativos disponíveis na Google Play Store, totalizando 108 bilhões de *downloads* [Ruth, 2022]. Essa enorme popularidade impulsiona também o desenvolvimento de *malwares* para a plataforma. Estatísticas indicam que há mais de 500 mil amostras de aplicativos maliciosos catalogadas [Ruth, 2022]. Além da popularidade, o lançamento de aplicações maliciosas é alimentado ainda pela facilidade na distribuição, pois usuários de Android podem também instalar aplicativos de lojas diferentes da oficial.

Nesse contexto, a abordagem de detecção de *malware* baseada em *machine learning* (ML) destaca-se por possuir o benefício de conseguir identificar padrões que podem ser aplicados para novos tipos de *malware*. Entretanto, o aspecto negativo dessa

abordagem é o fato de exigir conhecimento técnico, incluindo conceitos de matemática, estatística e computação, além de um profundo conhecimento do domínio do problema.

A AutoML (*Automated Machine Learning*) surgiu como uma forma de simplificar e automatizar o pré-processamento dos dados, a engenharia de características, a otimização de hiper-parâmetros dos algoritmos e o treinamento de modelos de aprendizado de máquina, agilizando a entrega de produtos (i.e., modelos otimizados) e reduzindo a necessidade de conhecimento técnico na área de ML [Karmaker et al., 2021]. Em linhas gerais, AutoML reduz a quantidade e o esforço de especialistas na realização de tarefas de aprendizado de máquina, objetivando atingir implementações ideais com nenhum, ou quase nenhum, esforço de especialistas.

Existem diversas ferramentas de AutoML, com variadas características e finalidades [He et al., 2021], como a Auto-Sklearn [Feurer et al., 2015], Google AutoML[1], DataRobot[2], Qeexo[3], AutoGluon [Erickson et al., 2020], TPOT [Olson and Moore, 2016] e QuickAutoML [Siqueira et al., 2021]. Resumidamente, essas ferramentas permitem obter artefatos testáveis com uma complexidade e um custo de tempo menor em comparação com abordagens manuais. Em particular, as ferramentas de AutoML reduzem substancialmente a curva de aprendizagem, uma vez que eliminam a necessidade de diversos conhecimentos específicos da área de aprendizado de máquina. Entretanto, essas ferramentas são tipicamente de aplicação geral, i.e., não são desenvolvidas com o objetivo de gerar modelos otimizados para domínios específicos.

As ferramentas de AutoML orientadas a um domínio possuem o benefício de permitir o processamento especializado de dados, de modo a obter o máximo de informação relevante em relação ao domínio, principalmente através da criação de novas características com maior valor preditivo. Por exemplo, existem soluções de AutoML específicas para classificação de diagnósticos médicos [Alaa and Schaar, 2018, Tsamardinos et al., 2020]. Essas soluções conseguem resultados melhores que ferramentas gerais de AutoML pelo fato de especializarem etapas importantes do *pipeline*, como o pré-processamento e a engenharia de características.

Neste trabalho, apresentamos uma avaliação de desempenho de quatro ferramentas de AutoML (Auto-Sklearn, AutoGluon, TPOT e QuickAutoML) para o domínio específico do problema de detecção de *malwares* Android. Para resolver esse problema, o algoritmo de *machine learning* recebe como entrada um conjunto de características sobre um aplicativo e retorna uma classificação do aplicativo entre maligno ou benigno. Considerando o conjunto de dezenas de ferramenta existentes e encontradas na literatura e Internet, as quatro ferramentas foram selecionadas por:

1. contemplarem todas as etapas do processo de AutoML, segundo as etapas descritas em [Nagarajah and Poravi, 2019];
2. possuírem código-aberto;
3. serem atuais e incluírem mecanismos e métodos atualizados de *machine learning*;
4. serem utilizadas por outros autores em trabalhos recentes [Ferreira et al., 2021, Nagarajah and Poravi, 2019, Truong et al., 2019, Erickson et al., 2020]; e

[1] https://cloud.google.com/automl

[2] https://www.datarobot.com/platform/automated-machine-learning/

[3] https://qeexo.com/

5. serem parametrizáveis e funcionarem em linha de comando, permitindo a utilização de diferentes *datasets* e a automação dos processos de testes e execução.

Para avaliar as ferramentas, utilizamos sete *datasets* distintos, todos disponíveis no repositório GitHub[4], bem como os códigos das ferramentas de AutoML, *scripts* de execução e detalhes de instalação e configuração do ambiente. Baseado em métricas de desempenho, analisamos a eficácia, em termos acurácia e *recall*, e eficiência, em termos de tempo de execução, das ferramentas estudadas. Os resultados indicam que há, de fato, particularidades de domínio e de dados que impactam os modelos e as métricas resultantes das ferramentas de AutoML.

O restante deste trabalho está organizado como segue. Na Seção 2 explicamos o processo de AutoML geral para tornar o trabalho auto-contido. Na Seção 3 apresentamos o método utilizado para analisar e comparar as ferramentas de AutoML. Nas Seções 4 e 5 são discutidos os resultados e os trabalhos relacionados.

2. AutoML

Um *pipeline* de desenvolvimento de soluções baseadas em *machine learning* convencional requer intervenção humana em diversas etapas, como pré-processamento de dados, seleção de algoritmos e otimização de hiper-parâmetros. Além de ser mais propenso a falhas que máquinas, os humanos envolvidos no *pipeline* de *machine learning* devem reunir conhecimentos avançados sobre a técnica de resolução de problema, incluindo conceitos de matemática, estatística e computação, e sobre o domínio do problema, incluindo compreender a construção e o comportamento de aplicativos e *malware*, por exemplo.

O conceito de AutoML pretende substituir, gradualmente, os requisitos e funções de humanos especialistas na técnica ou no problema no *pipeline* de *machine learning*. Resumidamente, AutoML é um paradigma para automatizar o *pipeline* de *machine learning* e reduzir o esforço manual envolvido com esse tipo de tecnologias, de modo a acelerar o desenvolvimento de modelos significativos e eficazes para resolver problemas de diferentes domínios [Karmaker et al., 2021].

Algumas ferramentas atuais de AutoML permitem otimizar a seleção de algoritmos e seus hiper-parâmetros. Escolher a melhor combinação de algoritmo e hiper-parâmetro pode ser realizado através do método conhecido como *Combined Algorithm and Hyper-parameter Selection* (CASH) [Guo et al., 2019]. As ferramentas desse tipo recebem como entrada um problema e um *dataset* e retornam para o usuário a melhor combinação disponível, conhecida pela ferramenta, de modelo e suas configurações de hiper-parâmetros. Por exemplo, o AutoML pode escolher entre modelos *Support Vector Machine* (SVM) ou *Random Forests*. As ferramentas de AutoML também podem realizar a otimização de hiper-parâmetros, ou HPO (*Hyper-Parameter Optimization*), como número de camadas intermediárias e épocas de treinamento em um modelo de aprendizado profundo.

A Figura 1 ilustra as quatro tarefas necessárias para a execução do fluxo de *machine learning* [Nagarajah and Poravi, 2019]: (1) pré-processamento dos dados; (2) engenharia de características; (3) seleção de modelo; e (4) ajuste de modelo. Essas etapas são detalhadas nas Subseções 2.1 a 2.4.

[4]https://github.com/Malware-Hunter/sbseg22-quickautoml

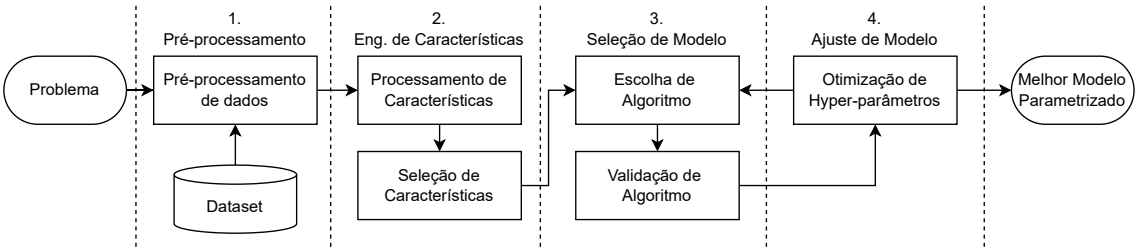

Figura 1. Fluxo de processamento de uma ferramenta de AutoML

2.1. Pré-processamento de dados

A primeira etapa do *pipeline* típico de AutoML é pré-processamento do *dataset*. Esta etapa é responsável pela correção de inconsistências e normalização de dados, já que o *dataset* pode reunir dados de múltiplas fontes com diferentes níveis de verificação e padronização. Exemplos de inconsistências incluem dados: ausentes, tipos de dados incorretos, dados redundantes e dados preenchidos incorretamente. Portanto, o objetivo desta etapa é aumentar a qualidade dos dados [Rahm and Do, 2000] e torná-los adequados para as etapas subsequentes.

Entre as ações realizadas na etapa de pré-processamento podemos citar a detecção e correção de erros, como valores ausentes ou fora de limiares aceitáveis, como estoque negativo. Na etapa de pré-processamento também pode ser realizado normalização de dados (e.g., padronização de unidades de medida, agrupamento para variáveis categóricas), discretização e transformação não linear (e.g., transformação logarítmica), bem como pré-processamentos específicos para certos tipo de dados, como remoção de pontuação [Nagarajah and Poravi, 2019].

2.2. Engenharia de características

A segunda etapa é identificar quais características do *dataset* são consideradas mais relevantes para o modelo de aprendizagem e ajustar sua apresentação para melhorar as próximas etapas. O objetivo é aumentar a eficiência computacional do processo (e.g., reduzir o uso de memória) focando em elementos que ajudem na resolução do problema e deixando em segundo plano ou descartando elementos irrelevantes. Por exemplo, para predizer o salário de uma pessoa, um algoritmo poderia selecionar o grau de instrução e descartar a cor do cabelo.

A principal operação da etapa de engenharia de característica é a redução de características. Essa operação reduz a dimensionalidade do *dataset* para economizar recursos. Uma redução pode viabilizar o carregamento para a memória de aplicação de aprendizado de máquina de grande porte, por exemplo. Entre as soluções utilizadas nesta etapa, estão as técnicas de redução de dimensionalidade, como PCA (Análise de Componentes Principais) ou algoritmos de seleção de características como SelectKBest (*Select K Best Features*), RFE (*Recursive Feature Elimination*), RFECV (*Recursive Feature Elimination and Cross-validated Selection*) para essa tarefa. Uma coleção mais aprofundada de outros algoritmos e soluções possíveis de serem empregados na etapa de engenharia de características está disposta na Seção 3.1.

Outra operação da etapa de engenharia de características é a expansão de características. Isso pode ser realizado através da decomposição de um campo em múltiplos

outros ou através do cálculo de métricas derivadas (e.g., percentil de um população) de métricas principais (e.g., salário). Isso pode ser útil para facilitar o processamento interno da técnica de aprendizado de máquina.

2.3. Seleção de Modelo

A terceira etapa é selecionar um modelo de aprendizado de máquina considerando as alternativas conhecidas pelo sistema, o *dataset* e o problema de entrada. É importante observar que algoritmos de aprendizado de máquina são meta-heurísticas que oferecem soluções aproximadas para problemas considerados computacionalmente difíceis para se encontrar a solução exata. Portanto, a meta-heurística mais adequada pode variar dependendo do problema e dos dados disponíveis.

Entre os algoritmos indexados pelas ferramentas de AutoML estão: KNN, RandomForest, AdaBoost, CatBoost, DecisionTree, SVM e suas variações LinearSVM, KernelSVM. Uma lista completa dos algoritmos suportados por cada ferramenta é relacionada na Seção 3.1. Adicionalmente, o manual de cada ferramenta apresenta descrições dos algoritmos suportados.

2.4. Ajuste de Modelo

Nesta etapa é realizado o processo de escolha e avaliação de valores para os parâmetros de entrada do algoritmo selecionado na etapa anterior. Para cada algoritmo é necessário fornecer uma combinação diferente de hiper-parâmetros, isto é, valores que controlam o processo de aprendizado de determinado modelo [Wu et al., 2019]. Por exemplo, algoritmos baseados em árvores necessitam que sejam definidos valores para profundidade máxima, número máximo de nós-folha e critério para medir a qualidade de uma partição. A otimização manual desses hiper-parâmetros é uma tarefa repetitiva, tediosa e com sucesso baseado no acaso ou na experiência obtida em treinamentos anteriores.

3. Método

Nesta seção detalhamos o método de comparação das ferramentas. Nós apresentamos as ferramentas de AutoML escolhidas (Subseção 3.1), os *datasets* (Subseção 3.2) e o ambiente de execução (Subseção 3.3). Os resultados da execução de cada uma das ferramentas sobre cada um dos *datasets* são apresentados e discutidos na Seção 4.

3.1. Ferramentas de AutoML

A Tabela 1 resume as principais características observadas na documentação oficial ou manual das ferramentas de AutoML consideradas neste estudo. As colunas refletem as etapas principais do processo de AutoML, ou seja, pré-processamento, engenharia de características, escolha de algoritmos de *machine learning*, e hiper-parâmetros a serem ajustados.

Com relação ao pré-processamento dos dados, cada ferramenta apresenta as suas particularidades. Por exemplo, a QuickAutoML automatiza parte desse processo, incorporando a detecção e a remoção de valores ausentes e registros duplicados, bem como a remoção de colunas com variância = 0 (i.e., preenchida com valores que nunca mudam para nenhum registro). A Auto-Sklearn implementa apenas métodos de tratamento de

Tabela 1. Etapas do *pipeline* de *machine learning* das ferramentas

Ferramenta	Pré-processamento de dados	Engenharia de características	Algoritmos de ML	Ajuste de hiper-parâmetros
QuickAutoML	Remoção de registros totalmente nulos, remoção de colunas com variância = 0, remoção de colunas com duplicatas	Criação de novas colunas especializadas para conjuntos de dados de aplicações Android	KNN, RandomForest, AdaBoost	*Pruning*
auto-sklearn	Tratamento de valores ausentes	Extremely Randomized Trees Preparation, FastICA, Feature Agglomeration, Randomized Kitchen Sinks, LinearSVM Preparation, Nystroem Sampler, Polynomial, Principal Component Analysis (PCA), KernelPCA, Randomized Trees Embedded, One Hot Encoding, balanceamento e *rescaling*	AdaBoost, BernoulliNB, DecisionTree, ExtremelyRandomizedTrees, GaussianNB, GradientBoosting, KNN, LDA, LinearSVM, KernelSVM, MultinomialNB, PassiveAggressive, QDA, RandomForest, SGD	Otimização bayesiana + meta-aprendizado
AutoGluon	Discretização e tratamento de valores nulos, processamento de valores textuais e de data	Variance Threshold	LightGBM, CatBoost, RandomForest, Extremely Randomized Trees, KNN	Fixa padrões definidos adaptativamente
TPOT	Não se aplica	Recursive Feature Elimination, Select KBest, Select KPercentile, Variance Threshold, Standard Scaler, Robust Scaler, Polynomial Features, Randomized Principal Component Analysis (Randomized PCA)	DecisionTree, RandomForest, GradientBoosting, SVM, LogisticRegression, KNN	Algoritmos genéticos

valores ausentes, através do uso da média, mediana ou do valor mais frequente da coluna. Já a AutoGluon introduz métodos de discretização de dados, tratamento de valores nulos e processamento de valores textuais e de datas, que são transformados em vetores numéricos utilizando $n - gramas$. Por fim, a TPOT não implementa nenhuma etapa de pré-processamento de dados.

Na etapa de engenharia de características podemos observar uma diferença significativa entre as ferramentas, e.g., Auto-Sklearn e TPOT incorporam diversas técnicas de seleção de características, como PCA, PPA (*Principal Polynomial Analysis*) e *One Hot Encoding*. Esse fato poderá destacar essas ferramentas em termos de tempo de execução (menor) e qualidade final dos modelos, isto é, acertividade na classificação. As outras duas ferramentas, QuickAutoML e AutoGluon, deixam a desejar na engenharia de características para AutoML. Vamos observar o impacto disso nos resultados dos modelos.

Com relação ao ajuste de hiper-parâmetros, podemos observar que cada ferramenta utiliza uma técnica distinta, variando de *pruning* (QuickAutoML) até algoritmos genéticos, como é o caso da TPOT. É importante ressaltar que algoritmos genéticos podem demandar uma complexidade computacional maior, possivelmente aumentando o tempo de execução da ferramenta. Em síntese, as técnicas e combinações de ajustes de hiper-parâmetros podem ter também um impacto significativo no tempo de execução das ferramentas.

Finalmente, há ainda uma diversidade significativa em termos de algoritmos de *machine learning*. Enquanto as ferramentas QuickAutoML, TPOT e AutoGluon avaliam de três a seis algoritmos, a Auto-Sklearn avalia quinze algoritmos. O fato de avaliar mais algoritmos, juntamente com a combinação de diversas técnicas de seleção de características, torna a ferramenta Auto-Sklearn uma forte candidata aos melhores resultados em termos de métricas. No outro extremo tem-se a AutoGluon, que possui uma etapa de engenharia de características mais simples e uma quantidade de combinações potencialmente menor de otimização de hiper-parâmetros devido aos padrões pré-definidos, o que a torna uma forte candidata aos menores tempos de execução, mas não necessariamente aos melhores resultados em termos de métricas.

3.2. *Datasets*

Para avaliar o desempenho das ferramentas selecionadas no processo de detecção de *malware* em aplicativos Android, utilizamos os seguintes *datasets*: Drebin_215 [Arp et al., 2014], Androcrawl [SISTO, 2013], Android Permission Dataset[5] e MDroid_46k_3k, criado por nós a partir da extração das características de 46.670 APKs disponíveis no repositório Androzoo[6], sendo 43.827 aplicativos benignos e 2.843 malignos. Cada *dataset* possui diversas características das categorias de permissões, intenções e chamadas de API. Para o *dataset* MDroid_46k_3k, que representa o maior conjunto de dados, criamos também três subconjuntos de dados, no qual cada um contém as características de uma única categoria (e.g., apenas permissões). Com isto, consideramos um total de sete *datasets* com variados tipos e números de características e variadas quantidades e proporcionalidades entre as amostras, conforme detalhado na Tabela 2. Os *datasets* estão publicamente disponíveis no GitHub[4].

Tabela 2. *Datasets*: características e amostragens

Dataset	**Tipos de Características**	**# Cars.**	**Amostras**
mdroid_46k_3k	Chamadas de API, Intenções, Permissões	3.048	46.670 (43.827B+2.843M)
mdroid_46k_3k_permissions	Permissões	1.316	
mdroid_46k_3k_api_calls	Chamadas de API	1.524	
mdroid_46k_3k_intents	Intenções	208	
androcrawl	Chamadas de API, Intenções, Permissões	81	96.732 (86.562B+10.170M)
drebin_215	Chamadas de API, Intenções, Permissões	209	15.031 (9.476B+5.555M)
apcd_permissions	Permissões	143	27.298 (9.074B+18.224M)

[5]https://www.kaggle.com/datasets/saurabhshahane/android-permission-dataset
[6]https://androzoo.uni.lu/

3.3. Ambiente

O ambiente do experimento consiste de uma imagem Docker, que contém os pacotes necessários para a instalação das dependências de cada ferramenta, como o Python 3.8 e o `pip`, bem como os *datasets* e os códigos que realizam os testes em si. As dependências das ferramentas são instaladas automaticamente na imagem Docker de forma isolada, através de um ambiente virtual do Python, ou seja, há um ambiente virtual para cada ferramenta. Isso é necessário porque algumas ferramentas possuem dependências conflitantes. Esse ambiente simplifica a reprodução deste trabalho, pois requer apenas conhecimentos básicos em Docker. Os códigos, bem como a configuração da imagem Docker, estão disponíveis no GitHub[4].

A execução do experimento foi conduzida em quatro computadores idênticos com processador Intel(R) Core(TM) i7-9700 CPU @ 3GHz (8 núcleos) e 16GB RAM e HD. O sistema operacional utilizado foi o GNU/Linux Debian 11 Kernel 5.10.0-14-amd64.

3.4. Métricas

Como métricas de avaliação de desempenho das ferramentas, selecionamos três: acurácia, *recall* e tempo de execução. A acurácia é utilizada para visualizar a porcentagem de predições corretas dado o total de predições realizadas, enquanto que o *recall* identifica quantos aplicativos foram corretamente classificados como malignos dentre o número de aplicativos malignos.

4. Resultados

A Tabela 3 apresenta os resultados da execução das quatro ferramentas de AutoML sobre todos os sete *datasets*. Cada linha da tabela representa as entradas (i.e., *dataset* e ferramenta) e as métricas de saída (i.e., acurácia, *recall* e tempo de execução). O tempo, no formato `Horas:Minutos:Segundos`, representa a média de dez execuções.

Como podemos observar, para o *dataset androcrawl*, as ferramentas Auto-Sklearn e AutoGluon obtiveram resultados similares. Enquanto a ferramenta Auto-Sklearn obteve 98,87% de acurácia, 92,99% de *recall* e tempo médio de execução de 1 minuto e 58 segundos, a AutoGluon obteve 98,87% de acurácia, 92,63% de *recall* e execução em 1 minuto e 4 segundos. Resumidamente, a AutoGluon atingiu o mesmo desempenho em termos de acurácia e *recall* em aproximadamente metade do tempo que a Auto-Sklearn. É interessante observar também que não houveram diferenças significativas entre duas ferramentas de AutoML apesar de ambas serem significativamente distintas em termos de recursos nas etapas de engenharia de características e algoritmos de ML, como pode ser visto na Tabela 1.

Já as ferramentas QuickAutoML e TPOT resultaram em um tempo de execução significativamente maior (31 minutos e 48 segundos e 18 minutos e 43 segundos, respectivamente), com acurácia similar (e.g., 98,68%) e *recall* ligeiramente maior (e.g., 94,01%). A diferença expressiva no tempo de execução indica que as ferramentas não lidam muito bem com conjuntos de dados com uma alta dimensionalidade (e.g., acima de 1.000 características). No caso da QuickAutoML, a ferramenta carece de algoritmos e métodos na etapa de engenharia de características, o que ajudaria a diminuir significativamente o tamanho dos *datasets* através da seleção das características mais significativas. No caso

Tabela 3. Resultados das ferramentas de AutoML (média de 10 execuções)

Dataset	Ferramenta	Acurácia	*Recall*	Tempo
androcrawl	AutoGluon	0,988753	0,926391	00:01:04
	QuickAutoML	0,988441	0,923399	00:31:48
	Auto-Sklearn	0,988785	0,929982	00:01:58
	TPOT	0,986812	0,940156	00:18:43
apcd_permissions	AutoGluon	0,711177	0,763394	00:00:14
	QuickAutoML	0,709846	0,815807	00:05:17
	Auto-Sklearn	0,714286	0,827454	00:01:57
	TPOT	0,707515	0,785857	00:02:29
drebin_215	AutoGluon	0,986698	0,989733	00:00:13
	QuickAutoML	0,959098	0,977862	00:02:27
	Auto-Sklearn	0,987707	0,993263	00:02:00
	TPOT	0,983273	0,986526	00:07:06
mdroid_46k_3k_all	AutoGluon	0,963835	0,561629	00:04:18
	QuickAutoML	0,942927	0,533693	01:03:04
	Auto-Sklearn	—	—	—
	TPOT	0,963122	0,538049	02:02:21
mdroid_46k_3k_api_calls	AutoGluon	0,962472	0,629153	00:02:27
	QuickAutoML	0,95817	0,086957	00:20:27
	Auto-Sklearn	—	—	—
	TPOT	0,962602	0,561629	01:39:30
mdroid_46k_3k_intents	AutoGluon	0,941176	0,215434	00:00:43
	QuickAutoML	0,782837	0,042735	00:00:56
	Auto-Sklearn	—	—	—
	TPOT	0,941176	0,133976	00:30:00
mdroid_46k_3k_permissions	AutoGluon	0,946565	0,240085	00:02:55
	QuickAutoML	0,874607	0,265833	00:14:34
	Auto-Sklearn	—	—	—
	TPOT	0,944812	0,301179	01:02:37

da TPOT, a ferramenta é desprovida de qualquer técnicas de pré-processamento dos dados e utiliza algoritmos genéticos para ajuste de hiper-parâmetros, o que ajuda a explicar o tempo de execução maior.

No caso do *dataset apcd_permissions*, todas as ferramentas obtiveram um tempo de execução menor, o que é esperado pelo fato de ser o menor conjunto de dados em termos de número de características. Com relação às métricas de acurácia e *recall*, todas as ferramentas atingiram um resultado significativamente abaixo de outros casos, como o *androcrawl*. A explicação para esses resultados (e.g., 71,11% de acurácia e 76,33% de *recall* da AutoGluon) pode estar relacionada à qualidade e relevância das características contidas no *dataset*. Por exemplo, ao executarmos o método SigPID [Sun et al., 2016] sobre o *dataset apcd_permissions*, identificamos apenas 66 permissões classificadas como relevantes, isto é, esse conjunto reduzido de características (com alguma relevância) pode conter pouca informação para os modelos aprenderem a classificar mais acertivamente as amostras entre benignas e malignas.

No caso do *dataset drebin_215*, podemos observar um alto valor de *recall* para todas as ferramentas, sendo a melhor a Auto-Sklearn (99,32%) e a pior a QuickAutoML (97,78%). Na acurácia podemos observar uma diferença percentual similar entre estas. Em termos de tempo de execução, é interessante observarmos que a AutoGluon mantém a sua liderança, mas há uma mudança de comportamento em relação à Auto-Sklearn, que se aproximou da QuickAutoML. Isso pode ser explicado pela quantidade de características. Diferentemente do *dataset androcrawl*, que contém apenas 81 características, o *drebin_215* contém 209 características, o que aumenta o trabalho da etapa de engenharia de características. Enquanto a QuickAutoML possui uma etapa de engenharia de características simplificada, a Auto-Sklearn incorpora uma gama significativa de algoritmos para redução de dimensionalidade do *dataset*. Naturalmente, quanto mais características no *dataset*, maior será o trabalho da ferramenta. Isso pode ser verificado também no *dataset mdroid_46k_3k* e suas variantes. A Auto-Sklearn não conseguiu sequer completar a execução devido ao consumo de memória causado pela quantidade maior de características.

Finalmente, no caso do *dataset mdroid_46k_3k_all* e seus subconjuntos, podemos observar mais alguns resultados interessantes. Por exemplo, a ferramenta Auto-Sklearn apresentou o erro "*Dummy prediction failed with run state StatusType.MEMOUT*", que está associado a um problema de sobre-consumo de memória (i.e., múltiplas cópias de memória dos mesmos dados) de alguns subprocessos da ferramenta de acordo com discussões técnicas no repositório da própria ferramenta (e.g., *issues/978*). Mesmo que o limite de configuração do sistema seja aumentado, a falha irá persistir e ainda está sem previsão de resolução. Como o *dataset mdroid_46k_3k_all*, e seus subconjuntos, contém um número mais elevado de características, o problema apareceu e, possivelmente, deve estar relacionado aos algoritmos da etapa de engenharia de características, que agora irão ter um trabalho muito mais significativo em termos de tempo de computação e número de instâncias concomitantes para identificar e selecionar as características mais relevantes.

Outra observação importante para o *dataset mdroid_46k_3k_all* e seus subconjuntos é o fato de os resultados serem positivos em termos de acurácia (e.g., 96,38% para o AutoGluon), mas muito baixos para *recall* (e.g., apenas 56,16% para o AutoGluon), uma das métricas mais importantes na detecção de *malwares* Android. Esses resultados

podem ser explicados pela relevância das características contidas nos *datasets* para o treinamento e aprendizado dos modelos. Utilizando os métodos de seleção de características SigPID [Sun et al., 2016] e JOWMDroid [Cai et al., 2021], rapidamente constatamos que esses *datasets* contém pouquíssimas características relevantes. Por exemplo, os métodos SigPID e JOWMDroid reduzem em 99,97% e 95,70%, respectivamente, o número de características do *dataset mdroid_46k_3k_all*, o que significa que o número de características relevantes para classificar amostras em benignas e malignas é bastante reduzido, limitando a capacidade de aprendizado e classificação mais acertiva dos modelos de *machine learning*.

5. Trabalhos Relacionados

Diversos trabalhos recentes avaliam ou propõe ferramentas de AutoML para diferentes contexto e domínios [Nagarajah and Poravi, 2019, Ferreira et al., 2021, Bezrukavnikov and Linder, 2021, Truong et al., 2019, Karmaker et al., 2021]. Há estudos recentes (e.g., [Nagarajah and Poravi, 2019, Karmaker et al., 2021]) que apresentam análises de dezenas de trabalhos de AutoML, observando as particularidades de diferentes ferramentas, como Auto-Weka, Hyperopt-Sklearn, TPOT, AutoCompet e PennAI, e apontando desafios na área, como interpretabilidade, transparência, níveis de personalização e recursos de depuração.

Em [Ferreira et al., 2021], os autores apresentam um estudo comparativo entre aplicações de AutoML quanto a *machine learning*, *deep learning* e *XGBoost*. As ferramentas Auto-Keras, Auto-PyTorch, Auto-Sklearn, AutoGluon, H2O AutoML, rminer, TPOT e TransmogrifAI foram avaliadas com relação ao tempo de execução e desempenho de predição em doze *datasets*, disponíveis na plataforma OpenML[7], em diferentes domínios de aplicação, como a área médica (diagnóstico de doenças cardíacas, concentração de plasma e diabetes), métodos contraceptivos e classificação de risco de um cliente para concessão de crédito. Os resultados mostram que Auto-Sklearn teve o pior resultado quanto ao tempo, com uma média de 3.600 segundos para cada *dataset*. AutoGluon apresentou o melhor resultado para esse métrica, com média de 70 segundos, seguida por H20 AutoML (158s), TransmogrifAI (317s), rminer (408s). Quanto à predição, TransmogrifAI obteve o melhor resultado, com média de 88%, seguida por H20 AutoML, rminer e TPOT, com médias próximas a 87%. Já a Auto-Sklearn e AutoGluon produziram os piores resultados médios em termos de predição, com 80% e 78%, respectivamente.

Outros trabalhos realizam análise e comparações similares. Por exemplo, em [Truong et al., 2019] os autores investigam o desempenho de ferramentas de AutoML utilizando *datasets* de domínios e temas como doenças cardíacas, sequenciamento de DNA, anúncios em páginas da internet e admissibilidade em creches. Os resultados apontam que H2O AutoML, Auto-Keras e Auto-Sklearn têm desempenho superior às ferramentas Ludwig, Darwin, TPOT e Auto-ML. É importante destacar que esses resultados estão fortemente atrelados aos domínios e dados dos *datasets*.

Outros trabalhos, como [Bezrukavnikov and Linder, 2021], abordam a avaliação de ferramentas de AutoML sob a perspectiva quantitativa e qualitativa, com especial atenção sobre a experiência de usuários não especialistas em *machine learning*. Para os

[7]https://www.openml.org/

experimentos, foram selecionadas as ferramentas TPOT, Auto-Keras e Auto-Gluon e um *baseline* CatBoost com parâmetros selecionados por cientistas de dados profissionais. A fase de treinamento foi realizada com *subsets* de 250, 500 e 100 amostras de um *datasets* sobre despesas de clientes de um banco em compras; a fase de teste considerou 29.000 amostras. As ferramentas de AutoML devem prever a faixa etária de um cliente. O *baseline* superou as ferramentas de AutoML quanto a tempo de execução em todos os casos e na precisão para o conjunto de treinamento com 250 e 500 amostras. Para os casos com dados distribuídos em 1.000 amostras, a AutoGluon apresentou precisão levemente superior ao *baseline*. Quanto ao aspecto qualitativo, a Auto-Keras apresentou maior facilidade de utilização e compreensão em virtude de sua documentação mais detalhada, segundo a avaliação dos usuários.

É importante observar que os resultados variam de trabalho para trabalho, o que é esperado. Em *machine learning*, e consequentemente em AutoML também, há uma forte correlação dos resultados dos modelos com o domínio do problema e, em particular, com os dados dos *datasets*. Resumidamente, *datasets* com diferentes tipos de dados e com diferentes níveis de qualidades dos dados irão produzir resultados bastante distintos entre as ferramentas de AutoML. Como detalhamos na Seção 3.1, cada ferramenta incorpora diferentes recursos para pré-processamento de dados e engenharia de características, o que impacta significativamente nos resultados dos modelos de acordo com o tipo e a qualidade dos *datasets*. Portanto, é necessário avaliar as ferramentas de AutoML para cada domínio de problema e conjunto de *datasets*. Como os estudos existentes indicam, não há uma ferramenta que irá se destacar para qualquer domínio e qualquer conjunto de *datasets*. Portanto, é importante ressaltarmos que este trabalho é um primeiro passo importante em direção de uma avaliação extensiva de ferramentas de AutoML para *datasets* específicos ao domínio de detecção de *malwares* Android.

6. Considerações Finais & Trabalhos futuros

A avaliação de ferramentas AutoML, utilizando a mesma base de comparação, é relevante para que pesquisadores e organizações possam compreender suas características, bem como aspectos positivos e negativos. Neste trabalho apresentamos uma primeira avaliação de quatro ferramentas de AutoML (AutoGloun, Auto-Sklearn, TPOT e QuickAutoML) para o domínio de detecção de *malwares* Android. Os sete *datasets* utilizados, bem como o código das ferramentas, os *scripts* de execução e os detalhes de instalação do ambiente estão disponíveis no GitHub[4]. Os resultados indicam que os dados específicos do domínio, bem como as particularidades das próprias ferramentas, geram impacto sobre as métricas de saídas das ferramentas. Os resultados apresentados também contribuem para mostrar como diferentes *datasets* podem causar uma variação no desempenho das ferramentas em termos de métricas como tempo de execução e acurácia.

Como trabalhos futuros podemos elencar:

1. incrementar etapas fundamentais do *pipeline*, como a de engenharia de características, nas ferramentas de AutoML existentes para incorporar métodos de seleção de características sofisticados e otimizados para o domínio de detecção de *malwares* Android, como [Mahindru and Sangal, 2019, Moutaz, 2020, Cai et al., 2021];

2. investigar as particularidades de implementação das ferramentas de AutoML, que podem ser eventualmente otimizadas para melhorar a qualidade dos modelos gerados para o domínio de detecção de *malwares* Android;
3. avaliar uma quantidade e diversidade expressiva de *datasets* existentes no respectivo domínio, como os mapeados em [Soares et al., 2021]; e
4. ampliar a quantidade de ferramentas de AutoML na avaliação.

Agradecimentos

Esta pesquisa foi parcialmente financiada, conforme previsto nos Arts. 21 e 22 do decreto no. 10.521/2020, nos termos da Lei Federal no. 8.387/1991, através do convênio no. 003/2021, firmado entre ICOMP/UFAM, Flextronics da Amazônia Ltda e Motorola Mobility Comércio de Produtos Eletrônicos Ltda. O presente trabalho foi realizado também com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Referências

Alaa, A. and Schaar, M. (2018). Autoprognosis: Automated clinical prognostic modeling via bayesian optimization with structured kernel learning. In *International conference on machine learning*, pages 139–148. PMLR.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., and Siemens, C. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, pages 23–26.

Bezrukavnikov, O. and Linder, R. (2021). A neophyte with AutoML: Evaluating the promises of automatic machine learning tools. https://arxiv.org/abs/2101.05840.

Cai, L., Li, Y., and Xiong, Z. (2021). Jowmdroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Computers & Security*, 100:102086.

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., and Smola, A. (2020). AutoGluon-Tabular: Robust and accurate AutoML for structured data. https://arxiv.org/abs/2003.06505.

Ferreira, L., Pilastri, A., Martins, C. M., Pires, P. M., and Cortez, P. (2021). A comparison of AutoML tools for machine learning, deep learning and xgboost. In *IJCNN*, pages 1–8.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In *NIPS*, pages 2962–2970.

Guo, X., van Stein, B., and Bäck, T. (2019). A new approach towards the combined algorithm selection and hyper-parameter optimization problem. In *IEEE SSCI*, pages 2042–2049.

He, X., Zhao, K., and Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622. 10.1016/j.knosys.2020.106622.

Karmaker, S. K., Hassan, M. M., Smith, M. J., Xu, L., Zhai, C., and Veeramachaneni, K. (2021). AutoML to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)*, 54(8):1–36.

Mahindru, A. and Sangal, A. L. (2019). DeepDroid: Feature selection approach to detect Android malware using deep learning. In *IEEE 10th ICSESS*, pages 16–19.

Moutaz, A. (2020). Automated malware detection in mobile app stores based on robust feature generation. *Electronics*, 9:435.

Nagarajah, T. and Poravi, G. (2019). A review on automated machine learning (AutoML) systems. In *IEEE 5th I2CT*, pages 1–6.

Olson, R. S. and Moore, J. H. (2016). TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR.

Rahm, E. and Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13.

Ruth, C. (2022). Over 30 million new malware samples found in 2022 as cyber threats evolve. shorturl.at/ilNZ5.

Siqueira, G., Rodrigues, G., Kreutz, D., and Feitosa, E. (2021). QuickAutoML: Uma ferramenta para treinamento automatizado de modelos de aprendizado de máquina. In *VI Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg)*. https://sol.sbc.org.br/index.php/errc/article/view/18547.

SISTO, A. (2013). AndroCrawl: studying alternative Android marketplaces. https://www.politesi.polimi.it/handle/10589/88407.

Soares, T., Siqueira, G., Barcellos, L., Sayyed, R., Vargas, L., Rodrigues, G., Assolin, J., Pontes, J., Feitosa, E., and Kreutz, D. (2021). Detecção de Malwares Android: datasets e reprodutibilidade. In *VI Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg)*.

Sun, L., Li, Z., Yan, Q., Srisa-an, W., and Pan, Y. (2016). SigPID: significant permission identification for android malware detection. In *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 1–8.

Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, C. B., and Farivar, R. (2019). Towards automated machine learning: Evaluation and comparison of AutoML approaches and tools. In *IEEE 31st ICTAI*, pages 1471–1479.

Tsamardinos, I., Charonyktakis, P., Lakiotaki, K., Borboudakis, G., Zenklusen, J. C., Juhl, H., Chatzaki, E., and Lagani, V. (2020). Just add data: Automated predictive modeling and biosignature discovery. *BioRxiv*.

Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., and Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40.