UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

THALES HENRIQUE DANTAS SOUTO

TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA DETECÇÃO DE SISTEMAS VULNERÁVEIS

CAMPINA GRANDE - PB

2019

THALES HENRIQUE DANTAS SOUTO

TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA DETECÇÃO DE SISTEMAS VULNERÁVEIS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Eanes Torres Pereira.

CAMPINA GRANDE - PB

2019

S728t Souto, Thales Henrique Dantas.
Técnicas de aprendizagem de máquina para detecção de sistemas vulneráveis. / Thales Henrique Dantas Souto. – 2019.

11 f.

Orientador: Prof. Dr. Eanes Torres Pereira.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Aprendizagem de máquina. 2. Vulnerabilidade de sistemas. 3. Algoritmo Random Forest. 4. Algoritmo LGBM. 5. Sistemas vulneráveis. 6. Processamento de dados. 7. Machine learning. I. Pereira, Eanes Torres. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

THALES HENRIQUE DANTAS SOUTO

TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA DETECÇÃO DE SISTEMAS VULNERÁVEIS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Eanes Torres Pereira
Orientador – UASC/CEEI/UFCG

Professor Dr. José Antão Beltrão Moura
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Examinador – UASC/CEEI/UFCG

Trabalho aprovado em: 25 de novembro 2019.

CAMPINA GRANDE - PB

TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA DETECÇÃO DE SISTEMAS VULNERÁVEIS:

Trabalho de Conclusão de Curso[1]

Thales Henrique D. Souto
Unidade Acadêmica de Sistemas e Computação, Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brasil, thales.souto@ccc.ufcg.edu.br

Eanes Torres Pereira
Unidade Acadêmica de Sistemas e Computação, Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brasil, eanes@computacao.ufcg.edu.br

Resumo

A indústria de *malwares* continua a ser um mercado organizado e eficiente dedicado a invadir meios de segurança tradicionais [1]. Uma vez que um sistema é infectado, indivíduos mal intencionados podem causar prejuízos para pessoas e empresas de diversas formas. O propósito deste trabalho é prever, através da implementação de técnicas de classificação, se um sistema é vulnerável. O estudo faz a avaliação dos algoritmos *Random Forest* e *LGBM* aplicados aos dados de máquinas com sistemas operacionais *Windows*. Utilizaremos um conjunto de dados que a *Microsoft* disponibilizou em uma plataforma online, esses dados são divididos em treino e teste com informações sobre máquinas que utilizam seus sistemas operacionais. Para ir além, o estudo realizado pode ser útil como base para pesquisas mais aprofundadas no campo de análise de vulnerabilidades com métodos de *machine learning* e processamento de dados.

1. Introdução

Malware é um termo da língua inglesa para se referir a todo tipo de software que tem o objetivo de causar danos ao sistema como interrupção de serviço, sequestro de rede, exploração de recursos e roubo de informações sigilosas. Esses softwares podem ser classificados de diversas formas, como por exemplo: *Trojan, Backdoor, Keylogger, Botnets, Ransomware* entre outros [2].

O processo de detecção de malwares está continuamente evoluindo [3], apesar deste avanço algumas máquinas mantém sistemas desatualizados e com isso a contaminação por programas maliciosos. Estes programas muitas vezes exploram falhas de versões de sistemas, falhas estas que já foram atualizadas e corrigidas em versões atualizadas.

As empresas muitas vezes não se dão conta que seus servidores podem estar rodando um sistema operacional com uma versão que existem bugs. Ao tentar essa abordagem de prever se um sistema é vulnerável utilizando *machine learning*, ajuda saber se esses servidores estão em perigo.

Um dos maiores desafios que softwares anti-*malwares* enfrentam hoje é a grande quantidade de dados que precisam ser avaliados como potencial atividade maliciosa. Por exemplo, os produtos da Microsoft que detectam *malware* em tempo real são executados em cerca de 600 milhões de computadores ao redor do mundo [4]. Isto gera milhões de dados diariamente para serem analisados como potenciais *malwares*.

1.1 Trabalhos relacionados

Em 2015, R. J. Mangialardo e J. C. Duarte [13] fizeram um estudo a partir de análise estática e dinâmica de *malwares*, com objetivo de classificar o tipo de *malware* encontrado em blocos de *bytes* extraídos de alguns sistemas. Isso envolvia exame de memória e análise forense. Para medir a eficácia de suas predições utilizaram as métricas Acurácia, Revocação (*Recall*), Precisão e F1-*Score*. Usaram a acurácia para comparar métodos de análise de *malware* (unificada, estática e dinâmica) e usaram as outras métricas para medir a eficiência de seu modelo ao classificar os tipos de *malwares*.

Em 2016, Felan Carlo C. Garcia e Felix P. Muga avaliaram a eficiência do algoritmo de Floresta Aleatória (*Random Forest*) para classificar tipos de *malware* [14] utilizando um *dataset* com cerca de 9.300 amostras. Incrementaram o modelo fazendo uso da técnica

de validação cruzada *K-fold* e chegaram a uma acurácia de 94,64%.

Como apresentado pelos trabalhos relacionados, técnicas de aprendizado de máquina se mostram eficientes quando queremos classificar os tipos de *malwares*, este é o ponto onde difere de nossa abordagem que será útil para predizer quão vulnerável está um sistema operacional.

2. Metodologia

2.1 Objetivos

2.1.1 Objetivo Geral

Neste estudo iremos fazer uma avaliação de se é possível detectar sistemas vulneráveis utilizando algoritmos de aprendizagem de máquina e análise de dados exploratória. Nosso objetivo é produzir um estudo que possa ser estendido para outras bases de dados e servir como auxílio para pesquisas futuras.

2.1.2 Objetivos Específicos

Ao final do trabalho apresentaremos resultados de métricas aplicadas nas predições de probabilidade de uma máquina *Windows* ser infectada por algum tipo de *malware*, com base nas diferentes informações da mesma.

2.2 Base de dados

Os dados foram obtidos a partir de uma plataforma online chamada *Kaggle* [15]. Estes dados foram disponibilizados a partir de uma competição organizada pela *Microsoft* na plataforma.

2.2.1 Descrição dos dados

Os dados contém 83 colunas em que cada uma delas é uma informação sobre uma única máquina arbitrária. O total de registros na base de treino é 8.921.483. A coluna *HasDetections* representa se a máquina em questão é vulnerável ou não, ela será a nosso rótulo, ou seja, será a coluna que iremos usar como *groundtruth*.

As colunas foram divididas em numéricas, categóricas e binárias (0 ou 1) e a distribuição das mesmas está representada na Imagem 1:

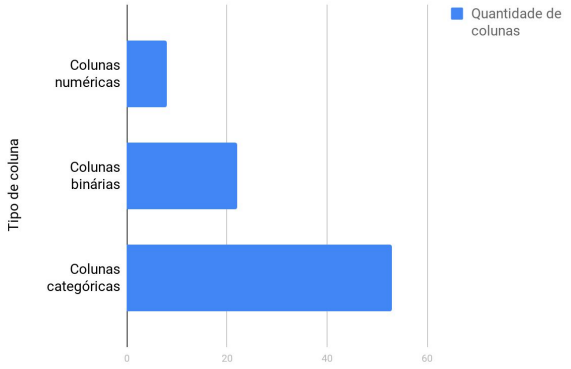

Figura 1. Quantidade de colunas numéricas, binárias e categóricas na sequência de cima para baixo.

Ao fazer uma análise em nosso rótulo de predição constatamos que os dados são balanceados, o que facilita a construção do nosso modelo, evitando enviesamento. Na Figura 2 temos a distribuição da coluna *HasDetections*:

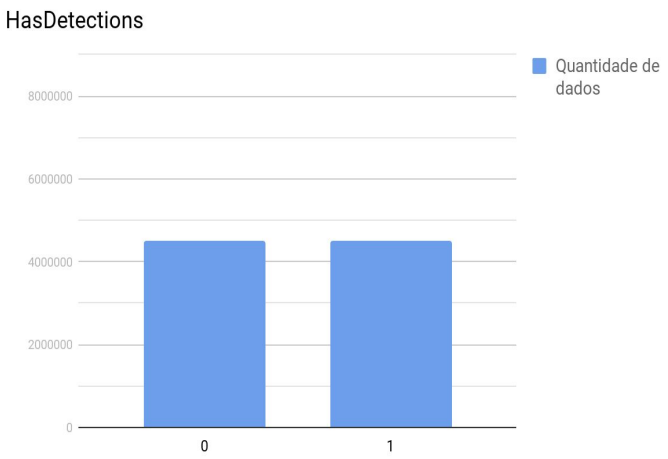

Figura 2. Quantidade de dados diferentes na coluna HasDetections

2.3 Pré-Processamento dos dados

Como nosso *dataset* tem uma grande quantidade de colunas, fica inviável trabalhar com todas visto que muitas delas servem apenas como identificadores. Sendo assim devemos fazer alguns tratamentos.

2.3.1 Remover colunas

Devido a necessidade de grande recurso computacional para o nosso modelo criar as categorias das colunas não numéricas, iremos remover as colunas que apresentam alta cardinalidade, ou seja, muitas categorias. Por isso removemos as colunas: *'AvSigVersion', 'DefaultBrowsersIdentifier', 'AVProductStatesIdentifier', 'CityIdentifier', 'OsBuildLab', 'Census_OEMNameIdentifier', 'Census_OEMModelIdentifier', 'Census_ProcessorModelIdentifier', 'Census_FirmwareManufacturerIdentifier', 'Census_FirmwareVersionIdentifier'*.

Após investigar colunas que têm muitos valores faltando (*null*), ou seja, mais de 50% dos dados são nulos, chegamos a conclusão que as seguintes colunas devem ser removidas, ilustradas através da Tabela 1.

	Porcentagem
PuaMode	99,97%
Census_ProcessorClass	99,59%
DefaultBrowserIdentifier	95,14%
Census_IsFlightingInternal	83,04%
Census_InternalBatteryType	71,04%
Census_ThresholdOptIn	63,52%
Census_IsWIMBootEnabled	63,44%

Tabela 1. Porcentagem das colunas com mais de 50% de valores nulos

2.4 Treinamento e validação

A criação de dados de treinamento e validação envolve dividir os dados em 80% para treino e 20% para validação. Ao realizar alguns estudos empíricos e investigações acerca dos dados de treinamento e de teste, chegamos a seguinte descoberta: treinamento e teste possuem uma variação temporal, ou seja, teste começa na mesma data que treinamento, entretanto termina 2 meses depois do fim do treinamento. Sabendo disso a nossa divisão de treinamento e validação deve seguir da mesma maneira. O Código 1, nos leva a essa conclusão.

```
train[‘Date’] =
train[‘AvSigVersion’].map(datedict)

test[‘Date’] =
test[‘AvSigVersion’].map(datedict)

train[‘Date’].min().date() // 2013-07-18
train[‘Date’].max().date() // 2018-09-25

test[‘Date’].min().date() // 2013-07-18
```

```
test['Date'].max().date() // 2018-11-25
```

Código 1. Trecho de código para saber data máxima e mínima dos conjuntos de dados.

2.5 Modelos

Os algoritmos utilizados para as predições neste estudo foram LGBM [16] e *Random Forest* [17], ambos utilizam a estrutura de árvores de decisão.

Árvore de decisão (*Decision tree*) é uma estrutura de diagrama no qual cada nó interno representa um teste de algum atributo, cada ramo representa a saída do teste corrente, e cada nó folha representa uma decisão após testar todos os atributos. Os caminhos da raíz até as folhas representam as regras de classificação [5].

No LGBM foi utilizada a validação cruzada, uma técnica de validação de modelo para avaliar como os resultados obtidos podem ser generalizados para um conjunto de dados "desconhecidos" pelo modelo, é usada principalmente quando se deseja estimar a precisão de um modelo na prática [6]. O método de validação cruzada que utilizamos foi o *K-fold* que usa todas as amostras disponíveis como amostras de treinamento e validação. O *K-fold* comparado a outras técnicas como *Hold-out* e *Leave-One-Out*, consegue chegar a resultados mais precisos, e algumas vezes demandando menor recurso computacional dependendo do tamanho do *K* [7].

2.5.1 *Random Forest*

Floresta Aleatória (*Random Forest*) é um algoritmo de aprendizagem supervisionada. que cria uma floresta, ou seja, uma combinação (*ensemble*) de árvores de decisão de forma aleatória.

2.5.2 *Light Gradient Boost Model* (LGBM)

Assim como as florestas aleatórias, o *Gradient Boost* utiliza-se de várias árvores de decisão, entretanto aplica-se pesos negativos em predições erradas e pesos positivos em predições corretas de cada indivíduo (árvores).

Em nosso estudo será usado um *framework* também da *Microsoft,* chamado *LightGBM*. Esse foi projetado para funcionar de forma distribuída e com objetivo de pouco uso de memória, maior eficiência computacional, suporte ao uso de GPU para treinar modelos e capaz de trabalhar com grande quantidade de dados [8].

2.6 Métricas

Para avaliar a eficiência das predições dos nossos modelos aplicamos algumas métricas conhecidas em aprendizagem de máquina.

2.6.1 Precisão (*Precision*)

A precisão é a métrica que calcula a eficiência do modelo para verdadeiros positivos, ou seja, daqueles classificados como corretos, quantos efetivamente eram? Pode ser explicada através da Equação 1:

$$Precisão = \frac{Verdadeiros\ Positivos\ (TP)}{Verdadeiros\ Positivos\ (TP) + Falsos\ Positivos (FP)}$$

Equação 1. Equação para o cálculo da precisão.

2.6.2 Revocação (*Recall)*

O *recall* é a frequência em que o classificador encontra os exemplos de uma classe, ou seja, "quando realmente é da classe X, o quão frequente você classifica como X?" [9]. Traduzido pela Equação 2:

$$Revocação = \frac{Verdadeiros\ Positivos\ (TP)}{Verdadeiros\ Positivos\ (TP) + Falsos\ Negativos\ (FN)}$$

Equação 2. Equação para calcular a revocação.

2.6.3 F1-*Score*

Essa métrica combina precisão e *recall* de modo a trazer um número único que indique a qualidade geral do seu modelo. Segue ilustração da mesma através da Equação 3.

$$F1 = \frac{2 \times precisão \times revocação}{precisão + revocação}$$

Equação 3. Equação para calcular a métrica F1.

3. Resultados

Os modelos foram treinados no conjunto de treinamento e avaliados usando os dois conjuntos: treinamento e validação, ambos partições do *dataset* **train.csv**.

3.1 Resultados das métricas com *Random Forest*

Após executarmos as predições com *Random Forest* obtemos os resultados em treinamento e validação representados pelas Tabelas 2 e 3 respectivamente.

TREINAMENTO	precisão	revocação (*recall*)	f1-*score*
HasDetections = 0	0,60	0,57	0,59
HasDetections = 1	0,60	0,63	0,62
média	0,60	0,60	0,605

Tabela 2. Resultados das métricas no conjunto de treinamento com *Random Forest*.

VALIDAÇÃO	precisão	revocação (*recall*)	f1-*score*
HasDetections = 0	0,62	0,50	0,56
HasDetections = 1	0,56	0,67	0,61
média	0,59	0,585	0,585

Tabela 3. Resultados das métricas no conjunto de validação com *Random Forest*

3.2 Resultados das métricas com LGBM

Levando em consideração que o LGBM tem mais eficiência computacional na execução dos modelos, decidimos aplicar a validação cruzada com $k = 5$. Feito isso, chegamos aos resultados mostrados nas Tabelas 4 e 5.

TREINAMENTO	precisão	revocação (*recall*)	f1-*score*
HasDetections = 0	0,63	0,61	0,62
HasDetections = 1	0,62	0,64	0,63
média	0,625	0,625	0,625

Tabela 4. Resultados das métricas no conjunto de treinamento com LGBM

VALIDAÇÃO	precisão	revocação (*recall*)	f1-*score*
HasDetections = 0	0,61	0,58	0,60
HasDetections = 1	0,60	0,63	0,61

média	0,605	0,605	0,605

Tabela 5. Resultados das métricas no conjunto de validação com LGBM.

4. Conclusão e trabalhos futuros

Neste trabalho nós mostramos como dados de milhões de máquinas com sistema operacional *Windows* disponibilizados pela *Microsoft* na plataforma *Kaggle* podem ser útil para treinarmos um modelo que classifica se uma máquina está vulnerável a algum *malware*. Os algoritmos usados foram *LightGBM* e *Random Forest,* ambos baseados em árvores de decisão. Avaliamos a eficácia dos modelos aplicando as métricas Precisão, *Recall*, *F1-Score* na base de treino que foi dividida em treino e validação. Para um possível trabalho futuro segue como sugestão o uso da técnica de *model stacking* que consiste em combinar informações de múltiplos algoritmos preditivos para chegar a um resultado mais preciso [10]. Redução de dimensionalidade em colunas com altas cardinalidades [11] e técnicas de otimização Bayesiana [12] também podem servir como ferramentas para alcançar diferentes resultados dos obtidos neste estudo.

Referências

[1] Industrial Control Systems Emergency Response Team (ICS-CERT) Advanced Analytical Laboratory (AAL). Malware Trends, 2016. Pág. 1.

[2] ERCIM NEWS: Cybercrime and Privacy Issues, Number 90, July 2012.

[3] Sanjay K. Sahay, Ashu Sharma, Hermant Rathore. Evolution of Malware and its Detection Techniques, 2019.

[4] Microsoft. Sam cybersecurity engagement kit, 2018. URL: https://assets.microsoft.com/en-nz/cybersecurity-sam-engagement-kit.pdf (Acessado em 04/11/2019)

[5] Decision tree, Wikipedia. URL: https://en.wikipedia.org/wiki/Decision_tree (Acessado em 10/11/2019)

[6] Alex Souza. Validação Cruzada: Conceito e Exemplo em R, 2019. URL: https://pessoalex.wordpress.com/2019/04/16/validacao-cruzada-conceito-e-exemplo-em-r/ (Acessado em 12/11/2019)

[7] Jacques Nelson Corleta Schreiber . Técnicas de validação de dados para sistemas inteligentes: uma abordagem do software SDBayes, 2017. Pág. 4.

[8] LightGBM, Microsoft. URL: https://github.com/microsoft/LightGBM (Acessado em 30/10/2019)

[9]Renato dos Santos Leal. Métricas Comuns em Machine Learning: como analisar a qualidade de chatbots inteligentes, 2017. URL: https://medium.com/as-m%C3%A1quinas-que-pensam/m%C3%A9tricas-comuns-em-machine-learning-como-analisar-a-qualidade-de-chat-bots-inteligentes-m%C3%A9tricas-1ba580d7cc96 (Acessado em 07/11/2019)

[10] Ben Gorman. A Kaggler's Guide to Model Stacking in Practice, 2016. URL: http://blog.kaggle.com/2016/12/27/a-kagglers-gu

ide-to-model-stacking-in-practice/ (Acessado em 12/11/2019)

[11] Patricio Cerda, Gael Varoquaux. Encoding high-cardinality string categorical variables, 2019.

[12] Javier Gonzalez. Introduction to Bayesian Optimization, 2017.

[13] R. J. Mangialardo e J. C. Duarte. Integrating Static and Dynamic Malware Analysis Using Machine Learning, 2015.

[14] Felan Carlo C. Garcia e Felix P. Muga. Random Forest for Malware Classification, 2016.

[15] Malware Prediction Challenge. URL: https://www.kaggle.com/c/microsoft-malware-prediction/data (Acessado em: 24/09/2019)

[16] Gradient Boosting. URL: https://en.wikipedia.org/wiki/Gradient_boosting (Acessado em: 30/10/2019)

[17] Random Forest. URL: https://en.wikipedia.org/wiki/Random_forest (Acessado em: 30/10/2019)