Avaliação da Eficácia de Classificadores de Malware ao Longo do Tempo

Fabrício Ceschin[1], **Marcos Castilho**[1], **André Grégio**[1], **David Menotti**[1]

[1] Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

{fjoceschin, marcos, gregio, menotti}@inf.ufpr.br

Abstract. *New malware samples are created to overcome defense mechanisms, demanding continuous improvement. Usually, the infection's behavior is not modified, but rather the way it is performed, thus invalidating signatures and detection heuristics. In this paper, we classify programs into malicious or benign using static features from a labeled dataset of over 38 thousand samples collected in the Brazilian cyberspace between 2013 and 2017. Then, we evaluate how these classifiers handle the number of changes faced in the malware scenario. We also release the dataset, attributes, classification algorithms and their parameters. Our results shed a light on the validity of classifiers and expose the need of constant updating security mechanisms.*

Resumo. *Novos programas maliciosos são criados para sobrepujar mecanismos de segurança, demandando contínua atualização. Em geral, o comportamento de infecção não é modificado, mas a maneira de executa-la, invalidando assim assinaturas e heurísticas de detecção. Neste artigo, propõe-se classificar estaticamente programas em maliciosos ou benignos usando um conjunto de dados com mais de 38 mil programas coletados no ciberespaço nacional entre 2013 e 2017. Avaliou-se então como os classificadores se comportam com as mudanças frequentes no cenário de malware. Todos os dados para reprodução são disponibilizados. Os resultados expõem a validade dos classificadores e a necessidade constante de atualização de mecanismos de segurança.*

1. Introdução

O problema dos ataques por *malware* torna-se mais grave com o passar dos anos, fazendo com que a dificuldade em se encontrar uma solução adequada aumente a distância entre a efetividade dos mecanismos de defesa e a complexidade, sofisticação, capacidade de evasão das ferramentas de proteção e disseminação por diversos meios dos exemplares atuantes. Aliado a isto, a quantidade massiva de *malware* moderno lançado na Internet diariamente traz a necessidade premente de criação de detectores automatizados. Em geral, a detecção de *malware* por meio de técnicas de aprendizado de máquina baseia-se na extração de características dos programas a fim de que se possa classificá-los em maliciosos ou benignos, ou mesmo apenas agrupar os maliciosos em famílias distintas, a depender dos atributos e conjuntos de dados utilizados.

Há, na literatura, inúmeras abordagens para classificação, agrupamento e detecção de *malware* cuja premissa é o uso de aprendizado de máquina [Huang and Stokes 2016, David and Netanyahu 2015, Leite et al. 2016, Prado et al. 2016]. Boa parte dos artigos

publicados na área apresentam problemas de metodologia e/ou reproducibilidade, seja por não disponibilizarem os dados, *scripts* e programas desenvolvidos para classificação dos exemplares e análise dos resultados, seja pela falta de representatividade/transparência na seleção de amostras. Ademais, o foco observado na literatura é na obtenção de taxas de acurácia próximas a 100%, não se notando a preocupação com o tempo de expiração das soluções de aprendizado de máquina produzidas. Logo, não se pode garantir que os resultados gerados são completamente isentos, uma vez que não há discussão sobre a eficácia dos classificadores ao longo do tempo, tampouco a base de dados para que outros pesquisadores façam novos testes.

Neste artigo, faz-se uma comparação entre alguns algoritmos clássicos de aprendizado de máquina, como *Support Vector Machines* (SVM), *Multi-layer Perceptron* (MLP) e k*-Nearest Neighbors* (KNN), voltados ao problema da classificação de programas em maliciosos ou benignos em um conjunto de dados representativo de exemplares coletados no espaço cibernético nacional de 2013 à 2017. Principalmente, provê-se uma análise crítica do tempo de expiração de um classificador (como é o caso do anti-vírus): identifica-se, através do treinamento (aprendizado) e teste (avaliação) dos classificadores gerados ano a ano, quando estes perdem a eficácia de detecção devido à evolução dos exemplares de *malware* em atuação. Além disso, supre-se uma lacuna na área de aprendizado de máquina aplicado à segurança, disponibilizando-se publicamente os vetores de atributos rotulados de 38.490 de programas únicos (19.979 maliciosos e 18.511 benignos), bem como as meta-informações que os geraram e os *scripts* (em *Python*) produzidos para extração, processamento e análise dos dados e resultados. Como contribuição mais significativa, além da validação da hipótese da expiração de detectores de *malware* ao longo do tempo, este trabalho também destaca a necessidade da frequência atualização dos mecanismos de segurança para manutenção da acurácia na detecção de ameaças. Embora esta máxima seja óbvia e a indústria já aplique atualizações constantes em seus produtos, a literatura acadêmica da área nunca se preocupou com uma análise mais embasada dos testes feitos usando classificadores gerados por algoritmos de aprendizado de máquina. São comuns as publicações que aplicam determinado algoritmo da moda em um conjunto restrito de amostras buscando máxima precisão, sem se avaliar como seria o comportamento do classificador quando aplicado em amostras que evoluem ao longo do tempo, tal qual acontece na vida real. Logo, não é de conhecimento dos autores que haja literatura que mostre como os classificadores gerados com dados de treinamento se comportariam no cenário de lançamento contínuo de *malware* que evolui constantemente.

O restante deste artigo está organizado da seguinte forma: na Seção 2, apresenta-se trabalhos relacionados com detecção de *malware* usando aprendizado de máquina; a base de dados utilizada neste trabalho é descrita na Seção 3; o processo de representação para aprendizado de máquina dos atributos extraídos dos programas é detalhado na Seção 4; os experimentos realizados, bem como a análise da expiração dos classificadores de *malware* ao longo do tempo são apresentados na Seção 5. Finalmente, na Seção 6, apresenta-se as considerações finais do trabalho.

2. Trabalhos Relacionados

A literatura de detecção, classificação e/ou agrupamento de programas maliciosos usando aprendizado de máquina é vasta [Gandotra et al. 2014], podendo ser dividida em três tipos de acordo com as características extraídas: baseados em atributos estáticos

provenientes do binário executável, em atributos dinâmicos provenientes da execução em ambientes controlados e, por fim, abordagens híbridas. Além disso, pode-se dividir as soluções entre as que consideram o componente de detecção (usam programas maliciosos e benignos para treinamento e teste do algoritmo de classificação) e aquelas que tentam classificar programas desconhecidos em grupos pré-existentes de programas sabidamente maliciosos. Entretanto, não há um *dataset* padrão que possa ser utilizado como *ground truth* para avaliação dos classificadores gerados, bem como raramente há disponibilização dos dados. A seguir, são mencionados alguns artigos que abordam a classificação de *malware* por meio da aplicação de técnicas de aprendizado de máquina, os quais apresentam os trabalhos relacionados de suas respectivas épocas.

Os primeiros trabalhos para classificação automatizada de programas maliciosos surgiram a partir de 2005. Neste mesmo ano, em [Gheorghescu 2005] apresenta-se um esquema para comparar vírus e identificar os desconhecidos com base no cálculo das distâncias entre os comportamentos armazenados observados. Em 2007, em [Bailey et al. 2007], foi proposto uma abordagem para classificação de *malware* em larga escala cuja entrada foi uma representação do comportamento dos exemplares como "mudanças de estado" que significavam a interação entre o programa e o sistema alvo. Os perfis gerados foram, então, comparados entre si para fins de agrupamento em famílias feito por cálculo de distância. Outras abordagens importantes baseadas em comportamento aparecem em [Bayer et al. 2009] e [Rieck et al. 2011]. No primeiro, é feita a identificação de comportamentos similares para agrupamento em famílias via algoritmo de *locality sensitive hashing* (LSH); no segundo, o objetivo é o mesmo, porém alcançado com o uso de *clustering* hierárquico. O agrupamento de *malware* em famílias também foi explorado em [Grégio et al. 2012], onde a distância de Jaccard foi usada no cálculo da similaridade entre perfis extraídos do *debugging* de programas maliciosos. Os perfis continham instruções específicas de escrita em memória na tentativa de se identificar ações maliciosas que diferenciassem famílias de *malware*.

Cabe ressaltar que os métodos apresentados até então fazem uso de técnicas para classificar apenas programas maliciosos em famílias e, dado um programa novo desconhecido (não rotulado), atribuí-lo a um dos grupos produzidos. Assim, nos casos apresentados, não foi considerada a existência de programas "benignos", limitando as contribuições para a proteção dos usuários. Mesmo em artigos mais recentes [Annachhatre et al. 2015, Ahmadi et al. 2016], o que se vê são conjuntos de dados menores (oito mil no primeiro e 20 mil no segundo), não abrangentes ao longo dos anos (coletados no mesmo ano ou ano anterior do artigo) e comuns (provenientes de *sites* de compartilhamento de *malware* ou *honeypots*). É importante notar também que os trabalhos cujos conjuntos de dados são compostos apenas por programas maliciosos podem ter resultados tendenciosos, conforme evidenciado em [Li et al. 2010] e que se baseia justamente nos resultados alcançados no já citado [Bayer et al. 2009]. O presente trabalho, por sua vez, busca sanar essas limitações ao mesmo tempo em que dá enfoque ao cenário brasileiro, que contém exemplares peculiares em relação aos de outros países. Com isso, espera-se prover um conjunto de dados representativo em tempo e variedade de amostras presentes no cenário nacional, e que permita, através da disponibilização dos vetores de atributos rotulados, novas análises e comparação com outros trabalhos da literatura por parte de outros pesquisadores. A seguir, são citadas duas abordagens nacionais recentes que consideram conjuntos de dados compostos tanto por *malware* quanto por *goodware*.

Tabela 1. Distribuição dos exemplares de *malware* e *goodware* entre 2013 e 2017.

Ano	<=2013	=2014	=2015	=2016	>=2017	Total
Malware	10.075	8.688	6.461	728	848	26.800
Goodware	11.930	512	1.057	5.009	3	18.511

Existem questionamentos envolvendo abordagens baseadas em aprendizado de máquina: a maioria das métricas de avaliação não consideram o tempo, um fator importante já que os exemplares de *malware* estão sempre evoluindo (eles alteram seus conceitos, um problema chamado de *concept drift*) em resposta a pressões externas (novas tecnologias e detectores) e internas (novas funcionalidades) [Singh et al. 2012]. Tentando resolver este problema, [Roberto Jordaney and Cavallaro 2016] apresenta o *conformal evaluator*, um *framework* de avaliação que usa métricas estatísticas para medir a qualidade dos resultados produzidos. [Roy et al. 2015] conclui que *AUPRC* (*Area Under the Precision Recall Curve* é uma métrica melhor para comparar resultados de diferentes abordagens em detectores para Android baseados em aprendizado de máquina.[Allix et al. 2015] mostra que utilizar um conjunto aleatório de *malware* conhecido para treinar um detector, como é feito em vários experimentos na literatura, produz resultados significativamente tendenciosos, ao passo que [Kantchelian et al. 2013] apresenta um *ensemble* de classificadores que é responsivo a mudanças, já que é composto de múltiplos classificadores, cada um representando uma família de *malware* diferente.

Em [Prado et al. 2016], propõe-se uma metodologia para detecção de *malware* baseada em heurísticas que se utiliza do comportamento de execução das amostras (*system calls*). Os experimentos foram feitos com 2.959 amostras (2.394 maliciosas e 565 benignas) para validar a metodologia proposta. No entanto, a reprodução é limitada, pois embora os vetores de características e a lista de MD5 dos exemplares utilizados tenha sido disponibilizada, os atributos extraídos não foram tornados públicos. Além disso, muito conhecimento *a priori* é necessário de um especialista para modelagem da heurística, não sendo possível o desenvolvimento de um sistema automático sem intervenção e análise do especialista a cada nova versão do sistema. Usando uma abordagem não-supervisionada baseada em *clustering*, em [Leite et al. 2016], o algoritmo *Fuzzy C-Means* (FCM) é utilizado para agrupamento de malware por comportamento usando lógica *fuzzy*, ao invés da lógica convencional (K-means). Experimentos mostram que o FCM é capaz de modelar, de forma mais fidedigna, o real comportamento malicioso exibido durante uma infecção. Esse trabalho utilizou exemplares maliciosos de 2013 à 2017 totalizando 21.455 amostras. E da mesma forma que o trabalho anterior, não é possível reproduzir estes experimentos pois não há dados públicos suficientes.

3. Conjunto de Dados Maliciosos e Benignos

O estudo realizado neste trabalho utiliza uma massa de dados de $\approx$180 GB em arquivos executáveis que foram coletados (Seção 3.1) no espaço cibernéticos nacional de 2013 à 2017 como ilustra a Tabela 1. Para cada arquivo coletado—rotulados como *goodware* ou *malware*, dependendo da sua origem—foram extraídos atributos para sua representação (Seção 3.2). A base de vetores de atributos e seus rótulos serão utilizados para aprendizado supervisionado de classificadores [Bishop 2006] capazes de predizer a classe de um arquivo desconhecido.

3.1. Coleta

Para a construção do *dataset*, foi necessário obter duas classes de *software*: *goodware* e *malware*. Os exemplares de *goodware* foram coletados utilizando um *web crawler* em três sites de *download*: *Sourceforge* [Slashdot Media 2016], *Softonic* [Softonic Internacional S.A. 2016] e *CNET Download* [CNET 2016]. No total, ≈130 GB de arquivos foram baixados, contabilizando 18.511 amostras únicas de *goodware*. Os exemplares de *malware* foram obtidos de uma instituição financeira nacional que prefere se manter anônima, advindos de mecanismos de segurança instalados em seus clientes ou coletados de *phishing*. Até o momento deste trabalho, foram coletados ≈50 GB de binários de *malware*, totalizando 26.800 exemplares (19.979 únicos).

3.2. Extração de Atributos

Esta etapa tem como objetivo extrair os atributos dos executáveis obtidos, de forma que fiquem disponíveis publicamente para futuros trabalhos. Para isso, a biblioteca *pefile* [Carrera 2016], que permite acessar atributos de arquivos do tipo *PE (Portable Executable)*, foi utilizada. Os atributos extraídos de cada executável estão destacados nas subseções a seguir [Pietrek 1994, Yonts 2010].

3.2.1. Atributos Gerais

Atributos extraídos utilizando informações gerais do arquivo:

- **Name:** nome do arquivo.
- **Size:** tamanho do arquivo.
- **MD5:** hash md5 que identifica unicamente um arquivo.
- **SHA1:** hash SHA1 que identifica unicamente um arquivo, semelhante à md5.
- **Entropy:** medida de entropia dos arquivos. Geralmente, quanto maior a entropia, maior a chance do arquivo estar comprimido ou criptografado.

3.2.2. Header

Atributos extraídos usando informações do cabeçalho do arquivo:

- **Machine:** identificador da CPU para o qual o arquivo é destinado.
- **NumberOfSections:** número de seções do arquivo.
- **TimeDateStamp:** horário em que o arquivo foi gerado, em segundos, desde às 4 horas da tarde de 31 de Dezembro de 1969.
- **FormatedTimeDateStamp:** horário formatado em que o arquivo foi gerado, no formato "ano-mês-dia hora-minuto-segundo".
- **PointerToSymbolTable:** offset da tabela de símbolos COFF.
- **NumberOfSymbols:** número de símbolos na tabela de símbolos COFF (Common Object File Format).
- **SizeOfOptionalHeader:** tamanho do header opcional do arquivo.
- **Characteristics:** flags com informações do arquivo. Indicam se existem realocações, se é uma imagem executável e/ou se é link dinâmico para biblioteca.

3.2.3. Header Opcional

Atributos extraídos usando informações do cabeçalho opcional do arquivo:

- **Magic:** identificador da arquitetura do arquivo.
- **SizeOfCode:** tamanho combinado e arredondado de todas as seções de código.
- **SizeOfInitializedData:** tamanho total de todas as seções que são compostas por dados inicializados.
- **SizeOfUninitializedData:** tamanho das seções que o loader aloca no espaço de endereço virtual e que não ocupa espaço no arquivo em disco.
- **BaseOfData:** RVA (Relative Virtual Address) em que a seção de dados do arquivo começa.
- **ImageBase:** possível local em que o arquivo será mapeado na memória.
- **SizeOfImage:** tamanho total das partes da imagem para o loader.
- **SizeOfHeaders:** tamanho do header do arquivo e da tabela de seção.

3.2.4. Outros

Outros atributos que podem ser úteis para discriminar malware e goodware:

- **PE_TYPE:** flag da biblioteca pefile que verifica se o arquivo é válido.
- **ImportedDlls:** lista de bibliotecas dinâmicas (DLL) utilizadas pelo arquivo.
- **ImportedSymbols:** lista de funções que o arquivo utiliza (todas as funções listadas pertencem à alguma biblioteca listada em ImportedDlls).
- **Identify:** lista de nomes de empacotadores, compiladores ou ferramentas usadas.

Os atributos coletados foram, então, escritos em dois arquivos de texto no formato *csv*, um para cada classe de *software* (*goodware* e *malware*).

4. Representação

A análise dos dados coletados permitiu dividir os atributos em três categorias:

- **Atributos únicos:** são únicos para cada executável e não são interessantes do ponto de vista de aprendizado de máquina, dado que não são discriminantes para as classes, apenas para os arquivos. Inclui as *hashes MD5* e *SHA1* e o nome do arquivo (*Name*). Podem ser relevantes em uma fase anterior, pois é possível checar *blacklists* ou *whitelists* de programas benignos ou malignos por suas *hashes*.
- **Atributos numéricos:** números inteiros ou reais que devem ser normalizados para que fiquem na mesma escala. Neste caso, os atributos numéricos são os seguintes: *BaseOfCode*, *BaseOfData*, *Characteristics*, *DllCharacteristics*, *FileAlignment*, *ImageBase*, *Machine*, *Magic*, *NumberOfRvaAndSizes*, *NumberOfSections*, *NumberOfSymbols*, *PE_TYPE*, *PointerToSymbolTable*, *Size*, *SizeOfCode*, *SizeOfHeaders*, *SizeOfImage*, *SizeOfInitializedData*, *SizeOfOptionalHeader*, *SizeOfUninitializedData*, *TimeDateStamp* e *Entropy*.
- **Atributos textuais:** conjunto de palavras que deve ser pré-processado e também normalizado antes de utilizado. No problema apresentado, existem três conjuntos de palavras: *Identify*, *ImportedDlls* e *ImportedSymbols*.

Note que apenas os atributos numéricos e textuais foram utilizados neste trabalho. As seções a seguir tratam do pré-processamento dos atributos textuais (Subseção 4.1) e da normalização de todos os atributos (Subseção 4.2).

4.1. Pré-Processamento

O pré-processamento dos atributos textuais tem como objetivo agrupar os textos (documentos) parecidos. Desta forma, programas que utilizam as mesmas bibliotecas, funções e compiladores tendem a ficar próximos no espaço de características. Para que isso seja possível, cada conjunto de palavras é transformado em um documento (no contexto de recuperação de informação), em que cada palavra é separada por um espaço. Assim, três "documentos" são gerados por arquivo, um para cada atributo textual. Em seguida, é feito o *case-folding* e a normalização de todos os textos: todos eles são mantidos em caixa baixa e qualquer caractere especial é removido, além de acentos e números. Com isso, é possível obter medidas estatísticas sobre cada documento, com base em suas palavras. Neste trabalho, utilizamos o modo de representação *Vector Space Model (VSM)*, o qual representa um documento por meio da importância relativa de suas palavras [Turney and Pantel 2010, Manning et al. 2008].

4.1.1. Vector Space Model

Dado um vocabulário de um conjunto de documentos, isto é, todas as palavras que aparecem neste conjunto, cada documento *i* é representado por um vetor $\vec{d_i} = (w_{i,1}, w_{i,2}, ..., w_{i,t})$, em que $w_{i,j}$ representa o *TF-IDF* (do inglês *Term Frequency—Inverse Document Frequency*) da *j-ésima* palavra do vocabulário. O *TF-IDF* é uma medida estatística utilizada para avaliar o quão importante uma palavra é para um documento em relação à uma coleção de documentos [Manning et al. 2008]. Esta medida é obtida através da multiplicação de dois termos:

- *Term Frequency (TF)*: mede com que frequência uma palavra/termo *t* ocorre em um texto/documento, i.e.,

$$TF(t) = \frac{\text{Número de vezes que } t \text{ aparece no documento}}{\text{Número total de palavras do documento}} \quad (1)$$

- *Inverse Document Frequency (IDF)*: mede o quão importante é uma termo *t*, i.e.,

$$IDF(t) = \log_e \left(\frac{\text{Número total de documentos}}{\text{Número de documentos com a palavra } t} \right) \quad (2)$$

Em resumo, cada texto/documento é representado por um vetor esparso que contém suas medidas de *TF-IDF* para cada palavra do vocabulário. Este pode ser reduzido a um número *V* de palavras, sendo essas as que mais estão presentes nos textos/documentos.

4.2. Normalização

Após o pré-processamento dos atributos textuais, o *Vector Space Model* de cada um é concatenado aos demais atributos de um arquivo, gerando um vetor de tamanho $22 + 3 \times V$, em que 22 é o número de atributos numéricos e *V*, o tamanho do vocabulário criado dos atributos textuais. Entretanto, ainda há uma diferença de escala entre as características extraídas, sendo necessário normaliza-las. A normalização feita sobre as características extraídas foi a *MinMax*, que escala todas as *features* (características) em um intervalo entre zero e um, utilizando a Equação 3, em que x_i é o valor da *feature x* de um arquivo

i, $min(x)$ é o menor valor dessa *feature* no conjunto de treino e $max(x)$, o maior valor dessa *feature* no mesmo conjunto.

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3}$$

A partir disto, todos os arquivos estão normalizados e prontos para a classificação.

5. Experimentos Realizados

Com as representações (conjunto de *features*) extraídas conforme a seção anterior, foram inicialmente realizados experimentos com quatro classificadores: *Multilayer Perceptron* (*MLP*), o tipo mais frequente de rede neuronal em classificação; *Support Vector Machines* (*SVM*); *K-Nearest Neighbors* (*KNN*); *Random Forest* (um *ensemble* de árvores de decisão). O *Multilayer Perceptron* utilizado é implementado pela *TFLearn*, uma *API* de alto nível construída sobre a *TensorFlow*, biblioteca de inteligência artificial do *Google* [Damien et al. 2016]. Para os experimentos, o *MLP* consistiu de duas camadas escondidas—a primeira constituída de *V*/2 neurônios e a segunda de *V*/3, em que *V* é o tamanho do vocabulário utilizado pelo *Vector Space Model*. Os demais classificadores utilizados são implementados pela biblioteca *Scikit Learn* [Pedregosa et al. 2011], sendo o *SVM* (implementação *SVC*), utilizado com *kernel* linear (*default* $C = 1$ – experimentos foram feitos com *kernel* RBF e *grid-search*, mas não foram promissores), e o *KNN*, com o valor de *K* fixado em 5. Valores diversos foram experimentados para os parâmetros e os mencionados foram os que obtiveram os melhores resultados em geral. A Subseção 5.1 apresenta os resultados gerais obtidos, seguido pelos resultados utilizando limiar (Subseção 5.2) e pela análise da expiração de classificadores (Subseção 5.3).

5.1. Resultados Gerais

Para validação dos experimentos, 50% dos exemplares de *goodware* e *malware* foram utilizados no conjunto de teste, enquanto que o restante foi para o conjunto de treino. Este procedimento foi realizado dez vezes pelo esquema de validação cruzada com dez partições (*folds*). O tamanho do vocabulário *V* adotado foi de 100 palavras.

A Tabela 2 apresenta os valores médios obtidos desse processo para as medidas de acurácia, *Recall*, *Precision* e *F1-score* (a média harmônica de *precision* e *recall*)—medidas clássicas para problemas de classificação binário (*malware*/*goodware*) em que há desbalanceamento de classes—para cada classificador utilizado. É possível observar que o classificador *Random Forest* obteve os melhores resultados, seguido pelo *KNN*, *MLP* e *SVM*, que ficou quase três pontos percentuais abaixo em praticamente todas as métricas. Entretanto, as medidas apresentadas podem não ser interessantes para aplicações reais nesta área, uma vez que falsos-positivos podem influenciar diretamente na segurança de um usuário, como o bloqueio/quarentena de uma aplicação legítima, o que seria representado por *Precision* em 100%.

5.2. Resultados utilizando Limiar

Imagine que um usuário instale um *software* benigno (*goodware*). O antivírus instalado deve ser maleável o suficiente para não classificar esse novo *software* como

Tabela 2. Valores médios (10 execuções) obtidos para os classificadores *SVM*, *MLP*, *KNN* e *Random Forest*.

Classificador	Acurácia	F1-score	Recall	Precision
SVM	95.45%	95.56%	93.11%	98.15%
MLP	97.57%	97.70%	97.69%	97.71%
KNN	97.65%	97.76%	97.51%	98.02%
Random Forest	**98.26%**	**98.34%**	**97.81%**	**98.87%**

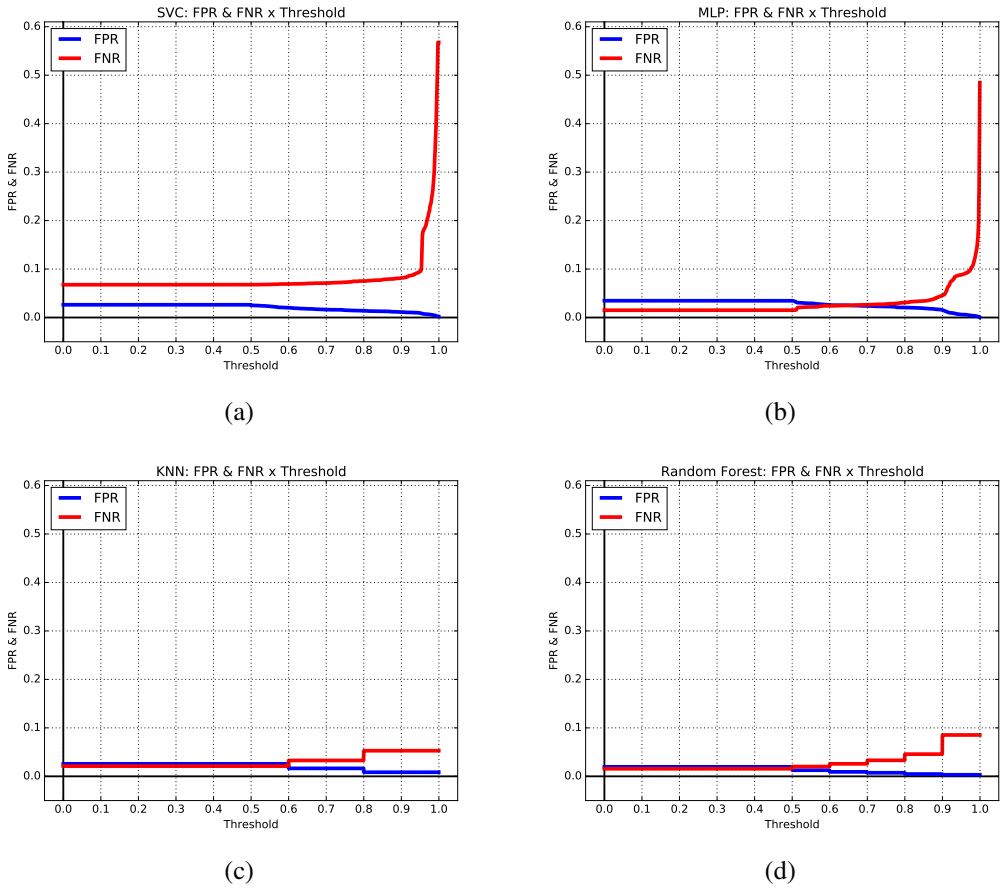


Figura 1. Gráfico: FPR & FNR x Threshold utilizando SVM (SVC), Multilayer Perceptron, KNN e Random Forest.

malware. Para isso, deve-se minimizar o número de falsos positivos a ponto de sacrificar a identificação de alguns tipos de *malware*. Em aprendizado de máquina, isso pode ser feito através de um limiar (*threshold*) que define um valor mínimo, que determina a probabilidade de um exemplo ser de determinada classe, para que um item seja efetivamente rotulado. Com base nisso, o desempenho dos classificadores foi avaliado utilizando limiares, com o objetivo de minimizar o número de falsos-positivos.

As Figuras 1(a), 1(b), 1(c) e 1(d) apresentam as taxas *FPR* (*False Positive Rate*) e *FNR* (*False Negative Rate*) para os classificadores *SVM*, *Multilayer Perceptron*, *KNN* e *Random Forest*, respectivamente, em um intervalo de limiares entre zero e um. Como esperado, quanto maior o limiar, maior o número de falsos-negativos, ou seja, maior o

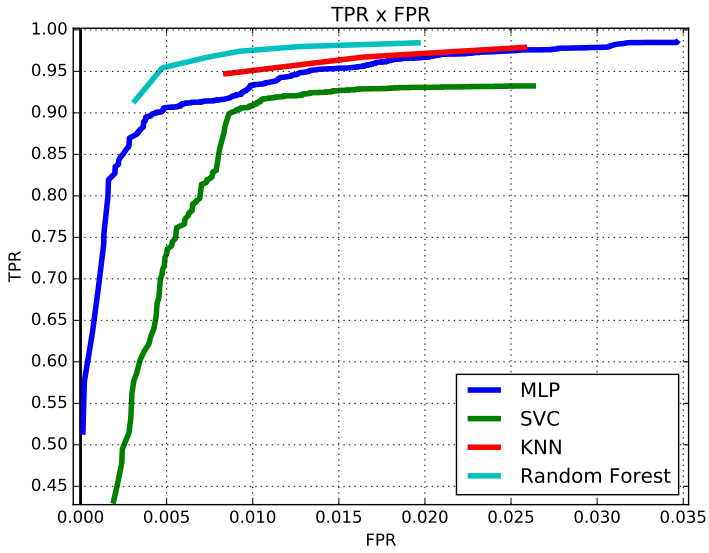


Figura 2. Curva *ROC* dos classificadores utilizados no experimento de limiar. Melhor compromisso de TPR vs FPR alcançado pelo Random Forest, mas precisão de 100% é obtida apenas pelo MLP.

número de *malware* que passou sem ser detectado (considerado *goodware*). Em contrapartida, ao passo que aumentam os falsos-negativos, os falsos-positivos diminuem, reduzindo o risco de classificar um *goodware* como *malware*. Tanto o *SVM* como o *MLP* apresentaram resultados semelhantes: quando o limiar atinge um valor próximo de 90%, os falsos-positivos diminuem razoavelmente, ao passo que os falsos-negativos aumentam drasticamente, aproximando-se de 60% de *FNR*, no caso do *SVM*, e de 50% de *FNR*, no caso do *MLP*. Já o *KNN* e o *Random Forest* apresentaram resultados mais consistentes. No caso do *KNN*, o *FPR* começa a baixar a partir de 60% de limiar e mantém-se estável até 80%, quando o valor cai para $\approx$1% e volta a se manter estável. O inverso acontece com o *FNR*, que se mantém estável em cerca de 5% a partir de 80% de limiar. Analisando o *Random Forest*, ele parece ter propriedades ainda mais interessante para o problema, dado que, aos 70% de limiar, apresenta um *FPR* e *FNR* abaixo de 5%. Entretanto, a partir de 90%, embora o *FPR* chegue muito próximo de zero, o *FNR* chega a quase 10%.

A Figura 2, curva *ROC* (*Receiver Operating characteristic*) dos classificadores mencionados, deixa claro o que já foi relatado nos gráficos anteriores: por conta do aumento do limiar, menor será o *FPR*, isto é, menos exemplares de *goodware* serão classificados erroneamente. Entretanto, isso tem um efeito colateral sobre o *TPR*, que tende a baixar. A curva também deixa claro o bom comportamento do *Random Forest*, que consegue manter baixo *FPR* e, ainda assim, ter *TPR* acima de 90%, mesmo com limiar muito alto. O comportamento do *KNN* também se mostrou eficiente, uma vez que consegue manter um *FPR* pouco acima de 2% e um *TPR* acima de 95%. Isso mostra que ambos os classificadores apresentam grande certeza sobre a maioria dos dados aprendidos. Já o *SVM* e o *MLP* têm uma queda considerável do *TPR* à medida em que o limiar aumenta.

Por outro lado, se pensarmos em um classificador (anti-vírus) "ideal" para o cenário comercial—aquele que não classifica nenhum *goodware* como *malware*—teríamos

apenas o MLP com esta propriedade. No entanto, para alcançar isto, a taxa de falsos negativos ($FNR = 1 - TPR$) obtida ficaria perto da casa dos 50%, ou seja, um em cada dois *malware* seriam corretamente detectados.

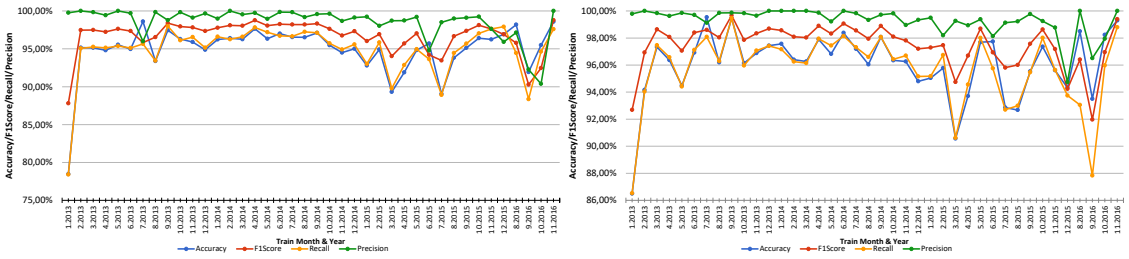
5.3. Análise da Expiração de Classificadores

Dois experimentos representando cenários reais foram conduzidos para se avaliar a expiração de classificadores que obtiveram os melhores resultados nos testes, i.e., *Random Forest* e *KNN*.

5.3.1. Experimento #1

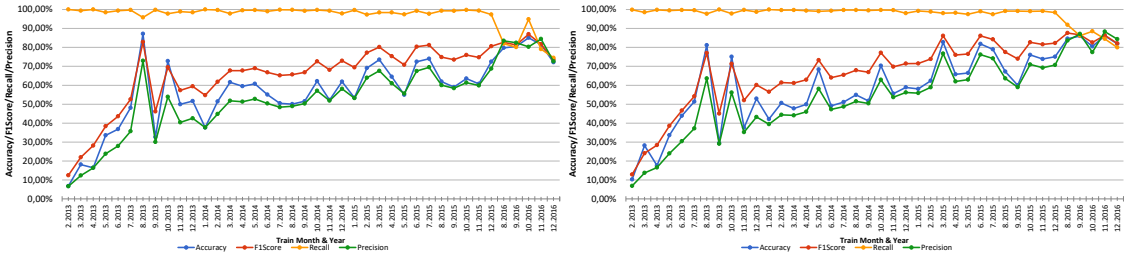
Neste experimento, os exemplares foram divididos em dois grupos: um contendo exemplos criados até determinado mês de um ano[1], usados para treino, e o outro com exemplos criados um mês após o mês escolhido, usado para validação. Por exemplo, se o mês é Fevereiro e o ano, 2015, o conjunto de treino terá exemplos que foram criados até Fevereiro de 2015 (conjunto cumulativo), enquanto que o conjunto de validação terá exemplos criados apenas em Março de 2015 (um mês após Fevereiro de 2015). Este experimento simula uma situação do mundo real, já que nós treinamos com dados que já foram vistos, i.e., *malware* e *goodware* que foram criados antes de determinado mês, e testado com dados do presente (do mês selecionado). A Figura 3(a) mostra os resultados para o *KNN*, enquanto a Figura 3(b) apresenta os resultados para o *Random Forest*, com acurácia, *f1score*, *recall* e *precision*. Em ambos os casos, existe uma grande variância nas métricas conforme o tempo passa. No caso do KNN, a melhor acurácia foi obtida em Julho/2013, com 98,61%, e o pior, em Janeiro/2013, com 87,81%. O *f1score* também é pior em Janeiro/2013, com 87,81%, e o melhor, em, Novembro/2016, com 98,78%. Olhando para o *recall*, o pior resultado também foi em Janeiro/2013, com 78,41%, enquanto que o melhor foi em Setembro/2013, com 97,99%. Por outro lado, a melhor *precision* foi obtida em Outubro/2016, com 90,38%, e a melhor, em Fevereiro/2013, Maio/2013, Fevereiro/2014 e Novembro/2016, com 100%. Estes resultados mostram que o *KNN* é um bom classificador para evitar falsos positivos, já que obteve 100% de precisão em 4 meses, e tem limitações para evitar falsos negativos, neste caso, já que o *recall* nunca atinge 100%. No caso do *Random Forest*, a pior acurácia foi em Janeiro/2013, com 86,50%, e o melhor, em Julho/2013, com 99,54%, enquanto que o pior *f1score* foi 91,97% em Setembro/2016 e o melhor, em Setembro/2013, com 99,66%. Olhando para o *recall*, o pior obtido foi em Janeiro/2013, com 86,53%, enquanto que o melhor foi obtido em Setembro/2013, com 99,46%. A pior *precision* foi obtida em Dezembro/2015, com 94,74%, e o melhor, em Fevereiro/2013, Dezembro/2013, Janeiro/2014, Fevereiro/2014, Março/2014, Junho/2014, Agosto/2016 e Novembro/2016, com 1200% em todos eles. É possível concluir que o *Random Forest* é um melhor classificador em comparação ao *KNN*, já que todas as métricas foram melhores na maioria dos casos. A queda significativa e a recuperação de todas as métricas com o passar do tempo em ambos os classificadores evidencia a existência de um *concept drift* neste *dataset*.

[1]Nos casos dos *malware*, o ano corresponde ao que o exemplo foi coletado, i.e., quando ele foi recebido por nosso parceiro; No caso dos *goodware*, foi utilizada a data de compilação.



(a) KNN (b) Random Forest

Figura 3. Resultados obtidos no Experimento #1.



(a) KNN (b) Random Forest

Figura 4. Resultados obtidos no Experimento #2.

5.3.2. Experimento #2

No experimento #2, inverteu-se os conjuntos de treino e validação do experimento #1. Com isso, o conjunto de treino possui exemplos de um determinado mês e ano apenas, enquanto que o conjunto de validação possui exemplos criados um mês antes deste mês (conjunto cumulativo). Por exemplo, dado o mês de Fevereiro e o ano de 2015, o conjunto de treino irá conter exemplos criados apenas este mês e ano, enquanto que o conjunto de teste conterá exemplos criados até um mês antes (Janeiro/2015). Este experimento também simula o mundo real, entretanto, nós usamos apenas dados de um mês selecionado (presente) para treinar e o resto (passado, antes deste mês), para validação. Com isso, nós podemos ver o quão representativo exemplos de um mês são para todo o conjunto de dados. A Figura 4(a) mostra os resultados obtidos pelo *KNN*, enquanto que a Figura 4(b) mostra os resultados obtidos pelo *random forest*, com as métricas de acurácia, *f1score*, *recall* e *precision*. Analisando os resultados do *KNN*, é possível ver uma acurácia realmente muito baixa no começo (Janeir/2013), com apenas 6,78%, sendo o melhor resultado 87,15%, em Agosto/2013. O melhor *f1score* também foi obtido em Agosto/2013, com 81,82%, e o pior, também em Janeiro/2013, com 12,47%. É curioso olhar para o *recall* e a *precision*: o *recall* atinge bons resultados, com 100% em Janeiro/2013, Abril/2013 e Janeiro/2013 (melhores resultados) e começa a cair em Dezembro/2015, obtendo 74,50% em Dezembro/2016 (pior resultado). A *precision*, por outro lado, tem 6,65% em Janeiro/2013 (pior resultado) e começa a aumentar conforme o tempo passa, apesar de algumas quedas no caminho, chegando a 84,45% em Novembro/2016. Apesar do *random forest* apresentar melhores resultados em geral, seus gráficos são muito similares aos do *KNN*. O *Random Forest* obtem a melhor acurácia em Março/2015, com 82,78%, e o pior, em Janeiro/2013, com 10,26%. Seu pior *f1score* foi também obtido em Janeiro/2013, enquanto que o melhor foi obtido em Agosto/2016, com 87,57%. O com-

portamento do *recall* também é similar ao *KNN*, já que começa a cair em Dezembro/2015 e possui valores próximos à 100% após este mês (nunca chega a 100% como o KNN, o melhor resultado foi em Janiro/2014, com 99,99%) e o pior resultado também foi em Dezembro/2016, com 79,96%. A *precision* também é similar ao *KNN*, com o pior resultado em Janeiro/2013, com 6,88%, e o melhor, em Novembro/2016, com 88,36%.

6. Conclusão

Neste artigo, foi estudado, implementado e avaliado um método estático para a detecção de malware baseado em aprendizado de máquina para analisar a expiração de classificadores (SVM, MLP, KNN e *Random Forest*) ao longo do tempo. Os atributos usados foram extraídos de arquivos executáveis do Windows no formato PE, tanto da classe *goodware* quanto da classe *malware*. Os vetores de atributos (e a descrição destes) obtidos a partir da base de dados construída para esse trabalho estão disponíveis à comunidade científica, de forma que uma comparação justa com outros trabalhos possa ser realizada, bem como para que os resultados possam ser reproduzidos ou testados com outras técnicas. Dos experimentos realizados neste trabalho podemos destacar os seguintes resultados: (i) o classificador *Random Forest* foi aquele que permitiu obter o melhor compromisso de classificação entre *goodware* e *malware*; (ii) em cenários específicos (e.g., comerciais, que necessitam de precisão próxima à 100%), classificadores usando MLP podem ser mais interessantes por permitirem controlar o ponto de atuação do detector; (iii) a hipótese levantada da expiração de classificadores ao longo do tempo foi verificada com sucesso, destacando a necessidade de uma frequente atualização dos mecanismos de segurança para manutenção da acurácia.

Referências

Ahmadi, M., Ulyanov, D., Semenov, S., et al. (2016). Novel feature extraction, selection and fusion for effective malware family classification. In *CODASPY*.

Allix, K., Bissyandé, T. F., Klein, J., and Le Traon, Y. (2015). Are your training datasets yet relevant? In *ESSoS'15*.

Annachhatre, C., Austin, T. H., et al. (2015). Hidden markov models for malware classification. *Journal of Computer Virology and Hacking Techniques*, 11(2):59–73.

Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., and Nazario, J. (2007). Automated classification and analysis of internet malware. In *10th International Conference on Recent Advances in Intrusion Detection (RAID'07)*, pages 178–197.

Bayer, U., Comparetti, P. M., et al. (2009). Scalable, behavior-based malware clustering. In *the Network and Distributed System Security Symposium (NDSS)*.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Carrera, E. (2016). pefile: A python module to read and work with pe (portable executable) files. `https://github.com/erocarrera/pefile`.

CNET (2016). CNET Downloads. `http://download.cnet.com/`.

Damien, A. et al. (2016). Tflearn. `https://github.com/tflearn/tflearn`.

David, O. E. and Netanyahu, N. S. (2015). Deepsign: Deep learning for automatic malware signature generation and classification. In *International Joint Conference on Neural Networks (IJCNN)*.

Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, 5(2):56–64.

Gheorghescu, M. (2005). An automated virus classification system. In *VB Conf.*

Grégio, A. R. A., de Geus, P. L., Kruegel, C., and Vigna, G. (2012). Tracking memory writes for malware classification and code reuse identification. In *DIMVA'12*.

Huang, W. and Stokes, J. W. (2016). Mtnet: A multi-task neural network for dynamic malware classification. In *DIMVA'16*.

Kantchelian, A., Afroz, S., Huang, L., Islam, A. C., Miller, B., Tschantz, M. C., Greenstadt, R., Joseph, A. D., and Tygar, J. D. (2013). Approaches to adversarial drift. In *AISec'13*.

Leite, L., e Silva, D. G., and Grégio, A. (2016). Agrupamento de malware por comportamento de execução usando lógica fuzzy. In *XVI SBSeg*.

Li, P., Liu, L., Gao, D., and Reiter, M. K. (2010). On challenges in evaluating malware clustering. In *RAID'10*.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pietrek, M. (1994). Peering Inside the PE: A Tour of the Win32 Portable Executable File Format. `https://msdn.microsoft.com/en-us/library/ms809762.aspx`, acessado em novembro de 2016.

Prado, N., Penteado, U., and Grégio, A. (2016). Metodologia de detecção de malware por heurísticas comportamentais. In *XVI SBSeg*.

Rieck, K., Trinius, P., Willems, C., and Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.*, 19(4):639–668.

Roberto Jordaney, Zhi Wang, D. P. I. N. and Cavallaro, L. (2016). Misleading metrics: On evaluating machine learning for malware with confidence. *Technical Report 2016-1 — Royal Holloway, University of London*.

Roy, S., DeLoach, J., Li, Y., Herndon, N., Caragea, D., Ou, X., Ranganath, V. P., Li, H., and Guevara, N. (2015). Experimental study with real-world data for android app security analysis using machine learning. In *ACSAC'15*.

Singh, A., Walenstein, A., and Lakhotia, A. (2012). Tracking concept drift in malware families. In *AISec'12*.

Slashdot Media (2016). SourceForge. `https://sourceforge.net/`.

Softonic Internacional S.A. (2016). Softonic. `https://en.softonic.com/`.

Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *CoRR*, abs/1003.1141.

Yonts, J. (2010). *Building a Malware Zoo*. The SANS Institute.