

Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais

Campus Rio Pomba

Ciência da Computação

LUIZ ANTONIO SANTOS GONSALVES JUNIOR

**Classificação de URLs no domínio da Segurança da Informação através de
técnicas de Machine Learning: Uma abordagem prática**

Rio Pomba

2019

LUIZ ANTONIO SANTOS GONSALVES JUNIOR

Classificação de URLs no domínio da Segurança da Informação através de técnicas de Machine Learning: Uma abordagem prática

Trabalho de Conclusão de Curso apresentado ao *Campus* Rio Pomba do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação

Orientador: Maurício Archanjo Nunes Coelho

Rio Pomba

2019

Ficha Catalográfica elaborada pela Biblioteca Jofre Moreira – Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Rio Pomba

G635c

Gonsalves Junior, Luiz Antônio Santos.

Classificação de URLs no domínio da segurança da informação através de técnicas de machine learning: uma abordagem prática./ Luiz Antônio Santos Gonsalves Junior. – Rio Pomba, 2019.

117. : il.

Orientador: Prof. Maurício Archanjo Nunes Coelho.

Trabalho de Conclusão de Curso (Graduação em Computação) - Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Rio Pomba.

1. Segurança da informação. 2. Aprendizagem por máquinas. 3. Inteligência artificial. I. Coelho, Maurício Archanjo Nunes. II. Título.

CDD: 004

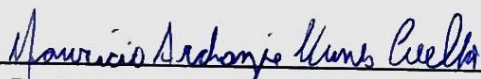
LUIZ ANTONIO SANTOS GONSALVES JUNIOR

Classificação de URLs no domínio da Segurança da Informação através de técnicas de Machine Learning: Uma abordagem prática

Trabalho de Conclusão de Curso apresentado ao *Campus* Rio Pomba do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação

Aprovada em: 05/12/2019

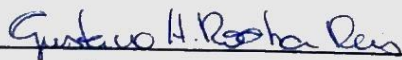
BANCA EXAMINADORA



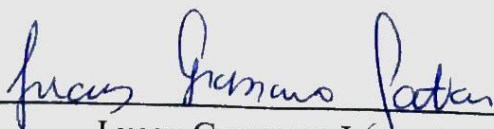
Prof. Dr. Maurício Archanjo Nunes Coelho -
Orientador
Instituto Federal de Sudeste de Minas Gerais



Bianca Portes de Castro
Instituto Federal de Sudeste de Minas Gerais



Gustavo Henrique da Rocha Reis
Instituto Federal de Sudeste de Minas Gerais



Lucas Grassano Lattari
Instituto Federal de Sudeste de Minas Gerais

Para Célia e Felomena.

AGRADECIMENTOS

Em primeiro lugar agradeço ao meu Deus que sempre me possibilitou, à sua maneira, meu crescimento como ser humano. Agradeço por sempre me acompanhar e permitir minha existência.

Agradeço imensuravelmente à meus familiares, em especial, à meus pais que sempre me apoiaram, me guiaram em meu caminho e me proporcionaram várias possibilidades em minha vida.

Agradeço também a meus professores do Instituto Federal do Sudeste de Minas Gerais campus Rio Pomba, de modo especial, ao meu orientador Maurício Archanjo Nunes Coelho, que esteve extremamente presente e que me forneceu suporte e conselhos bastante valiosos, sempre de forma respeitosa, compreensiva e paciente. Sem sua contribuição, este trabalho não teria sido possível.

Um agradecimento também aos professores da Escola Estadual Capitão Godoy, onde tive a oportunidade de cursar o ensino médio. Suas contribuições fomentaram não somente minhas bases, como também as bases de muitos outros alunos. Por ser, efetivamente até os dias atuais, o colégio melhor preparado na cidade onde vivia, sou grato pelos excelentes profissionais que o integravam/integram e que tive a satisfação de conhecer.

Finalmente, agradeço aos diversos cientistas, pesquisadores, engenheiros e estudiosos citados nas referências deste trabalho. Sua contribuição para a construção deste trabalho foi sem precedentes.

“[...] The barriers are self imposed. If you want to set off and go develop some grand new thing, you don’t need millions of dollars of capitalization. You need enough pizza and Diet Coke to stick in your refrigerator, a cheap PC to work on, and the dedication to go through with it. [...]”
(John Carmack, em entrevista parafraseada do livro ‘Masters of Doom’)

*“[...] Tired of lying in the sunshine staying home to watch the rain.
You are young and life is long and there is time to kill today.
And then one day you find ten years have got behind you.
No one told you when to run, you missed the starting gun.
So you run and you run to catch up with the sun but it’s sinking
Racing around to come up behind you again.
The sun is the same in a relative way but you’re older,
Shorter of breath and one day closer to death. [...]”*
(Pink Floyd, trecho da canção ‘Time’)

RESUMO

Este trabalho busca abordar a temática do problema de se classificar links de websites sobre a possibilidade de representarem alguma ameaça. Para tanto, é apresentado um tratamento que utiliza de técnicas de Inteligência Artificial para efetivar esta classificação, onde os links em si vêm a compor os conjuntos de treinamento para os modelos. Na construção deste trabalho, foi implementada uma aplicação capaz de efetuar a classificação de tais links, contando também com uma forma simplificada de construção de bases de informação para treinamento de modelos. Foram exploradas diversas individualidades acerca de cada elemento, algumas tomando por base trabalhos anteriores e outras, no entanto, formuladas no decorrer deste trabalho. Ao final, são discutidos os resultados obtidos com a utilização da aplicação em casos de teste.

Palavras-chave: Segurança da informação. Aprendizagem de Máquina. Inteligência artificial. Árvores de decisão. Máquinas de vetor suporte.

Abstract

This paper seeks to address the thematic of the issue of classifying website links over the possibility of representing some threat. Therefore, it is presented a treatment that utilizes Artificial Intelligence techniques for making this classification, where the links in itself compose the training sets for the models. In building this paper, an application capable of classifying such links has been implemented, also counting with a simplified form of building the information sets for training the models. Several individualities were explored about each element, some of which based on previous works and others, however, formulated in the course of this work. In the end, the results obtained by the use of the application in test cases are discussed.

Keywords: Information security. Machine Learning. Artificial Intelligence. Decision trees. Support vector machines.

LISTA DE FIGURAS

Figura 1 – Camadas no modelo TCP\IP.	24
Figura 2 – Camadas de rede no modelo OSI.	24
Figura 3 – Exemplificação de infraestrutura no modelo cliente-servidor.	27
Figura 4 – Divisões de uma URL.	28
Figura 5 – Exemplo de comunicação através do TCP. É perceptível que as marcações laterais indicam o fluxo temporal. Para fins de exemplificação, o conteúdo da comunicação foi representado de forma genérica.	30
Figura 6 – Endereço IPV4.	31
Figura 7 – Demonstração da hierarquia de consulta de servidores DNS.	33
Figura 8 – Conjunto hipotético em um plano 2D.	39
Figura 9 – Conjunto hipotético com uma possível fronteira delimitadora simples em um plano 2D.	39
Figura 10 – Descrição das etapas de <i>machine learning</i>	45
Figura 11 – Representação de um hiperplano separador.	46
Figura 12 – Separação de conjuntos através de um hiperplano.	48
Figura 13 – Hiperplanos separador representado com exemplos de classes distintas com triângulos e círculos.	50
Figura 14 – Hiperplanos separadores equivalentes gerando classificações distintas.	51
Figura 15 – Hiperplano e margens de separação dos conjuntos.	52
Figura 16 – Hiperplano classificador representado juntamente com H_1 e H_2	54
Figura 17 – Árvore de decisão simples para um problema hipotético de classificação de participação em uma entrevista de emprego.	58
Figura 18 – Vetores e ângulos para definição do produto escalar	104
Figura 19 – Diagrama de Classes (Simplificado)	121

LISTA DE TABELAS

Tabela 1 – <i>Features</i> anteriormente exploradas	73
Tabela 3 – Matriz de confusão 2x2 exemplificada	82
Tabela 4 – Resultados do J48	82
Tabela 5 – Matriz de confusão para 200 exemplos	83
Tabela 6 – Matriz de confusão para 1000 exemplos	83
Tabela 7 – Matriz de confusão para 2000 exemplos	83
Tabela 8 – Matriz de confusão para 3000 exemplos	83
Tabela 9 – Resultados do SVM	84
Tabela 10 – Matriz de confusão para 200 exemplos	85
Tabela 11 – Matriz de confusão para 1000 exemplos	85
Tabela 12 – Matriz de confusão para 2000 exemplos	85
Tabela 13 – Matriz de confusão para 3000 exemplos	85
Tabela 14 – Resultados do Random Forest	86
Tabela 15 – Matriz de confusão para 200 exemplos	86
Tabela 16 – Matriz de confusão para 1000 exemplos	86
Tabela 17 – Matriz de confusão para 2000 exemplos	86
Tabela 18 – Matriz de confusão para 3000 exemplos	87
Tabela 19 – Tabela de importância de atributos para 3000 exemplos	88
Tabela 20 – Valores aproximados das medidas de performance para 2000 exemplos .	88
Tabela 21 – Valores aproximados das medidas de performance para 3000 exemplos .	89
Tabela 22 – Tempo médio de execução da extração de <i>features</i>	89
Tabela 23 – Exemplo de resultado de extração	106

LISTA DE ABREVIATURAS E SIGLAS

WWW	World Wide Web
URL	Uniform Resource Locator
OSI	Open Systems Interconnection
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
DNS	Domain Name System
ICANN	Internet Corporation for Assigned Names and Numbers
SSAC	Security and Stability Advisory Committee
IANA	Internet Assigned Numbers Authority
W3C	World Wide Web Consortium
ASN	Autonomous System Number
IA	Inteligência Artificial
OWASP	Open Web Application Security Project
TLD	Top Level Domain
TCP	Transmission Control Protocol
IP	Internet Protocol
MDN	Malware Distribution Network
VM	Virtual Machine
CSV	Comma Separated Values
API	Application Programming Interface
OO	Orientada à Objetos

PLA	Perceptron Learning Algorithm
SVM	Support Vector Machine
CART	Classification and Regression Trees
MLP	Multilayer Perceptron
WEKA	Waikato Environment for Knowledge Analysis
TF-IDF	Term Frequency - Inverse Document Frequency

SUMÁRIO

1	Introdução	15
1.1	Segurança na Internet	17
1.2	Ameaças através de hiperlinks: Visão geral	19
1.3	Inteligência Artificial e problemas de classificação	20
1.4	Problema	21
1.5	Proposta do trabalho	21
2	Fundamentação teórica	23
2.1	Redes de computadores, WWW e URLs	23
2.1.1	A Internet e a <i>web</i>	25
2.1.2	URLs	27
2.1.3	HTTPS	29
2.1.4	Protocolo TCP	29
2.1.5	Endereçamento e DNS	30
2.1.6	WHOIS	33
2.2	Segurança da Informação	34
2.2.1	<i>Malwares</i>	34
2.2.2	Engenharia social	35
2.2.3	Ataques envolvendo URLs	36
2.2.4	Abordagens em segurança para o problema de URLs maliciosas	37
2.3	Fundamentos matemáticos	38
2.3.1	Vetores e separabilidade	38
2.3.2	Multiplicadores de Lagrange	41
2.3.3	Estatística	41
2.3.3.1	Probabilidade	41
2.3.3.2	Entropia	41
3	Inteligência Artificial	43
3.1	<i>Machine Learning</i>	43
3.2	Problemas de classificação	44
3.2.1	Treinamento	44
3.2.2	Novas instâncias	45
3.3	Técnicas	46
3.3.1	Classificadores lineares	46
3.3.2	SVM	46
3.3.2.1	Fundamentos do Perceptron	47

3.3.2.2	SVM de margens rígidas	49
3.3.3	Random Forest	57
3.3.3.1	Árvores de decisão	57
3.3.3.2	CART	58
3.3.3.3	Operações com múltiplas árvores	60
3.3.4	C4.5 e J48	61
3.3.5	Outros	63
3.4	WEKA	63
4	Trabalhos relacionados	65
4.1	Características das URLs	66
4.2	Outras ferramentas de análise de URLs	66
5	Desenvolvimento	68
5.1	Dependências	68
5.1.1	OpenCSV	68
5.1.2	GeoIP2 (Maxmind)	68
5.1.3	Apache Commons	69
5.1.4	dnspython	69
5.1.5	pyasn	70
5.2	Aplicação de princípios de Engenharia de Software	70
5.3	Estruturação	70
5.3.1	Classes Handler	71
5.3.2	Classes Feature Extractor	71
5.3.3	Utilities	72
5.3.4	Classe Intelligence	72
5.3.5	Programa principal	72
5.4	<i>Feature engineering</i> e modelagem	72
5.4.1	<i>Features</i> anteriormente exploradas	73
5.4.2	<i>Features</i> propostas	76
5.4.3	<i>Features</i> desconsideradas	76
5.4.4	Modelo	77
5.5	Coleta de informações	77
5.5.1	Dependência de outros serviços	78
5.6	Classificação	78
5.7	Armazenamento	79
6	Resultados	80
6.1	Dataset utilizado	80
6.2	Detalhamento da execução	81

6.2.1	J48	82
6.2.2	SVM	84
6.2.3	Random Forest	85
6.3	Detalhamento de performance	87
6.4	Extração e construção dos exemplos	89
6.5	Discussão	89
7	Conclusão	91
7.1	Trabalhos futuros	91

REFERÊNCIAS	94
------------------------------	-----------

A	Tipos de <i>malware</i>	101
A.1	Vírus	101
A.2	<i>Worm</i>	101
A.3	<i>Trojan</i>	101
A.4	<i>Spyware</i>	101
A.5	<i>Ransomware</i>	102
A.6	Outros	102
B	Definições Matemáticas	103
B.1	Detalhamento sobre vetores	103
B.2	Detalhamento sobre Multiplicadores de Lagrange	104
C	Exemplo de extração de URL	106
D	Árvore de decisão gerada pelo J48 para 3000 exemplos	107
E	Diagrama de classes	121

1 Introdução

A Internet se tornou uma tecnologia indispensável à maioria dos indivíduos, independentemente de etnia, crenças, nacionalidade ou qualquer outro fator que possa vir a ser usado para dissociar os mesmos. Como testemunhos concretos deste fenômeno, é apropriado citar os estudos de Curran (2012) e, em um contexto mais recente e prático, Madakam, Ramaswamy e Tripathi (2015). Se tratando de dados mais tangíveis, de acordo com o Internet World Stats, temos que 56,8% da população mundial se encontra conectada à Internet, com estes dados referentes ao mês de março de 2019¹.

Diversas foram as colaborações e eventos notórios que permitiram o avanço da Internet até a forma como ela se encontra hoje. Podem ser citadas, neste caso, as bases propostas através dos estudos de Bush et al. (1945), o advento da World Wide Web (WWW) em meados da década de 90 e os subsequentes trabalhos de diversos pesquisadores para o desenvolvimento da Internet (como exemplo, Hinden et al. (2017)). É perceptível que a Internet em sua totalidade não só auxiliou em processos tais como comunicação, comércio, educação, pesquisa e socialização, como também, criou as suas próprias feições de tais processos. Ainda, de acordo com o que é exposto por outros autores, é possível observar estes efeitos utilizando de analogias. “A Internet é a tecnologia decisiva da era da informação, assim como o motor elétrico foi o vetor das transformações tecnológicas da era industrial” (CASTELLS, 2014, p. 7, Tradução nossa).

A WWW é dita como um repositório de informações interligadas por diversos pontos espalhados ao redor do mundo (FOROUZAN, 2010). A mesma também pode ser entendida como o conjunto de sistemas integrado através da Internet, que visa fornecer conteúdo na forma de hipermídia. Por hipermídia, considera-se hipertexto, imagens, vídeos e diversos outros formatos de conteúdo. Estes sistemas integrados algumas vezes se manifestam através dos chamados *websites* que, basicamente, são páginas de conteúdo em hipertexto com funcionalidades específicas. Algumas vezes, estes *websites* também são denominados "páginas web".

Assim, a WWW, traduzida como "rede mundial de computadores", pode ser usufruída por qualquer usuário que disponha de uma conexão com a Internet, estes usuários podendo, além de visualizar, também contribuir, criando páginas na mesma. Cada uma dessas páginas pode fazer referência a diversas outras páginas do próprio sistema ou de outros na rede, compondo assim, um emaranhado de sistemas e permitindo que outros usuários possam usufruir e modificar o mesmo. Ilustrando bem esta característica está a seguinte descrição feita pelo próprio criador da rede, Berners-Lee (2010, p. 3, Tradução nossa):

¹ Fonte: <<https://www.internetworldstats.com/stats.htm>>

O primeiro princípio de design fundamentando a alta usabilidade e crescimento da *web* é a universalidade. Quando você faz um link, pode atribuir este link a qualquer coisa. Isto significa que pessoas devem ser capazes de colocar qualquer coisa na *web*, não importando qual o computador que tenham, software que usem ou língua humana que falem e não importando ainda se usem uma conexão com ou sem fio.

Para se ter uma ideia mais clara da dimensão do alcance da Internet na sociedade como um todo, é só observar o fato de que a Organização das Nações Unidas (ONU) declarou que o direito de acessar a Internet deve ser considerado um direito humano básico (KRAVETS, 2011). Ainda, com o crescimento do uso de dispositivos portáteis, como smartphones por exemplo (Penwarden et al. (2014) e Smith (2013)), e seu costumeiro custo reduzido, tem-se uma propagação da facilidade do acesso para as pessoas dos mais diversos níveis financeiros. No entanto, mais do que disponibilidade, é necessário que a rede seja um local seguro para seus usuários. Salientando essa preocupação, há o fato de que até 2012, mais de 50 nações já haviam publicado algum documento estratégico sobre Segurança da Informação, e o que a mesma significa para o desenvolvimento de tais nações (KLIMBURG, 2012).

Por segurança no ramo da computação, também denominada segurança da informação, entende-se como sendo a proteção oferecida a um sistema de informação, com a finalidade de se obter e manter integridade, disponibilidade e confidencialidade para os mais diversos elementos envolvidos no dito sistema (STALLINGS; BROWN, 2014). Para garantir a segurança, fazem-se necessárias a elaboração e implantação de normas específicas, bem como, a implementação de sistemas que sejam condizentes com a tríade anteriormente citada. Para o caso da World Wide Web, além da segurança que, idealmente, os responsáveis pelos próprios *websites* mantêm, têm-se ainda um escopo mais amplo: as práticas de segurança que são fomentadas pelos órgãos que regem a rede.

Como toda tecnologia, uma rede é naturalmente neutra e dessa forma, sendo a Internet uma rede que agrupa numerosas outras redes, a mesma é submetida à índole de seus usuários. Grande parte faz o uso da Internet de forma razoavelmente correta, algumas vezes descuidada, porém geralmente inofensiva aos demais usuários. No entanto, há aqueles que fazem mal uso da Internet, utilizando-a para disseminação dos mais diversos tipos de ataques.

Não é incomum o uso da Internet, em especial de sua faceta referente à *web*, para fraudes e tentativas de lucro de forma ilícita. No Brasil, por exemplo, de acordo com um relatório divulgado pela empresa de segurança computacional Kaspersky Lab, no segundo trimestre de 2017, cerca de 18,1% dos usuários foram alvo de ataques que faziam uso de páginas *web* fraudulentas².

² Fonte: <https://www1.folha.uol.com.br>

1.1 Segurança na Internet

A Internet é uma tecnologia com potencial para transformar os cenários competitivos em muitas áreas, ao mesmo tempo em que fornece as bases para o surgimento de áreas totalmente novas (AFUAH, 2002). Visto que várias empresas e órgãos vieram a assumir sua presença na *web*, assim também algumas das práticas criminosas que envolviam os mesmos passaram a se modernizar e transferiram sua manifestação também pela *web*. Um exemplo bem claro disto são às fraudes bancárias que, com a chegada dos serviços de Internet Banking, começaram a se adaptar também ao ambiente *web*, especialmente, na forma de *websites* maliciosos. No ano de 2019, uma operação conjunta³ do governo dos Estados Unidos e da União Européia prendeu um grupo de cibercriminosos que utilizou-se de fraudes no ambiente da *web* para tentar roubar cerca de U\$ 100 milhões de dólares de serviços de Internet Banking.

Cibercriminosos são uma designação para aqueles que utilizam da informática e da Internet para cometer delitos. Porém, não é apenas à disrupção de empresas e prejuízos às instituições bancárias que se limita o foco destes criminosos. Ataques a indivíduos específicos, sequestro de informações e espionagem são apenas algumas das outras ameaças que pairam sobre os usuários. Todas elas partilham algo em comum, pois todas estas ameaças podem se utilizar de *websites* maliciosos como vetor de ataque.

Por *website* malicioso, entende-se como aquele que foi desenvolvido ou modificado com o intuito de causar algum malefício ao usuário que vier a acessar o mesmo. Seu modo de operação, na maioria das vezes, requer que o usuário venha a acessá-lo para que então alguma forma de ataque se manifeste, este acesso sendo obtido muitas vezes através de um engodo ao usuário por algo que o site venha a oferecer (ESHETE, 2013). Um *website* malicioso se diferencia completamente de um *website* legítimo, dado que este último não representa, inicialmente, ameaça aos seus usuários. É interessante ressaltar que, embora um *website* não tenha sido inicialmente criado como uma possível ameaça, o mesmo pode vir a representar uma, caso o servidor ou qualquer outro recurso do qual o mesmo dependa esteja comprometido. Por razões como essa, detectar este tipo de ameaça é uma tarefa que exige bastante dinamismo.

Atualmente, as chamadas redes sociais também constituem um ambiente bastante propício a ameaças para seus usuários. Por aglomerarem uma quantidade massiva de pessoas, um cibercriminoso encontra um verdadeiro amontoado de possíveis alvos. O uso de redes sociais como vetor de ataque direto é especialmente comum para ataques que buscam obter acesso às próprias páginas das redes sociais de usuários específicos e, exemplificando, uma das formas mais comuns seria através de páginas *web* falsas. Os administradores de tais redes sociais tentam conter estas práticas criminosas, porém muitas

³ Detalhes disponíveis em: <https://www.zdnet.com/>

vezes são ineficientes em fazer o mesmo (WILLIAMS, 2019). O que acontece para o caso das redes sociais é que, dada a dependência única e exclusiva que seu conteúdo tem para com seus usuários, é necessário um monitoramento constante e minucioso sobre cada acesso, o que demanda recursos e implica em questões éticas sobre privacidade.

Diversos são os órgãos que operam quanto às questões referentes à segurança na rede, sendo eles internacionais ou representantes específicos de uma nação. A Internet Corporation for Assigned Names and Numbers (ICANN) pode ser considerada como o órgão maior responsável pela gestão da Internet globalmente, havendo ela mesma uma subsidiária específica para o trato de questões de segurança, denominada Security and Stability Advisory Committee (SSAC). Já o World Wide Web Consortium (W3C) é um agente regulamentador de padrões para a *web*, dentre os quais lida também com padrões referentes à segurança⁴, como padrões de criptografia e privacidade. Outros órgãos também contribuem para a manutenção da segurança na Internet e, inclusive, existem empresas privadas que apoiam iniciativas nesta área. Há ainda a organização Open Web Application Security Project, OWASP⁵, que se define como uma organização sem fins lucrativos que visa a melhoria da segurança de software globalmente, em especial para *websites* e aplicações *web*.

Se tratando especificamente no que tange *websites*, tem-se que os provedores e serviços de hospedagem também contribuem parcialmente para a segurança da rede, através de técnicas de bloqueio e manutenção da segurança de sites hospedados, respectivamente. Os provedores ainda, dependendo da legislação do país onde operam, podem manter registros de sites acessados, que podem vir a ser usados em possíveis investigações a respeito de crimes virtuais. Ainda, dado que a Segurança da Informação se dispõe como uma área bastante lucrativa e com alta demanda, existem também empresas privadas que realizam serviços referentes à segurança na *web*, bem como, desenvolvem novas técnicas e abordagens para tal.

O ambiente Internet e da World Wide Web atual possui considerável insegurança, sendo que tal insegurança é fruto, principalmente, da falta de confiabilidade fomentada por alguns usuários mal intencionados. A Internet foi criada seguindo a ideia de que seus usuários sempre seriam confiáveis (dado que eram em sua totalidade figuras dos domínios militar e científico, fato exposto por KUROSE e ROSS (2013)), porém no instante em que atingiu o grande público esta fragilidade veio a ser exposta. Ainda, conforme elucidado na época por Blumenthal e Clark (2001, p. 20, Tradução nossa): “De todas as mudanças que estão transformando a Internet, a perda de confiança pode ser a mais fundamental delas”. No entanto, também existem aqueles que buscam melhorar esta situação, desenvolvendo

⁴ Um detalhamento completo pode ser obtido através do link: <<https://www.w3.org/Security/>>

⁵ Mais informações sobre esta organização em: <https://www.owasp.org/index.php/Main_Page>

e fomentando práticas voltadas à segurança na *web*, representados aqui pelas empresas, organizações e comitês citados anteriormente.

1.2 Ameaças através de hiperlinks: Visão geral

Websites maliciosos são ferramentas poderosas nas mãos de pessoas mal intencionadas. Alguns fatores contribuem para a adoção desta tecnologia neste tipo de prática. Por exemplo, hoje em dia cria-se um site com muita facilidade, seja ele um site simples ou até mais complexo, haja vista a quantidade de ferramentas disponíveis para tal (PROVOS et al., 2007). Ainda, o custo de hospedagem de um site é relativamente barato, bem como, o de aquisição de um domínio na *web*. Um outro fator que pode ser citado aqui é a facilidade no reaproveitamento de códigos maliciosos e ofuscação dos mesmos. Por ofuscação, entende-se como as alterações no contexto sintático de um código a fim de tornar complexa a compreensão daquilo que o mesmo produz (SIKORSKI; HONIG, 2012).

As ameaças que pairam sobre um usuário ao fazer uso de alguma página *web* maliciosa são das mais diversas naturezas. Páginas podem, por exemplo, executar como se fossem um usuário desavisado algumas ações em detrimento a algum outro serviço *web* ao qual o mesmo esteja naquele momento autenticado. Ameaças que alterem o comportamento dos navegadores também são bastante comuns, além é claro dos já citados *websites* fraudulentos. Outros tipos de ataque mais elaborados podem vir a se utilizar de técnicas diferentes.

Um *website*, sendo ele malicioso ou não, pode ser identificado na WWW através de seu *Uniform Resource Locator* (Localizadora de Recurso Uniforme), o URL. Por definição, um URL é um conjunto de caracteres organizados de forma compacta, que representa um recurso disponível via Internet (BERNERS-LEE; MASINTER; MCCAILL, 1994). Em uma analogia, tem-se uma URL como sendo o “nome” de um recurso universalmente único. Por recursos, se define todo e qualquer elemento que possa ser providenciado em vias da Internet, variando desde hipermídias até mesmo acesso completo a servidores. URLs se dispõem na maioria das vezes na forma de hiperlinks dentro dos próprios *websites*, acessíveis bastando apenas uma designação por parte do usuário. Uma URL possui diversas características específicas à ela, tanto sintáticas (referentes à escrita da mesma) quanto semânticas (referentes à interpretação da mesma).

Desta forma, tem-se que *websites* são representáveis através de construções textuais de simples compreensão e únicas a cada *website* existente (KUROSE; ROSS, 2013). Isto faz com que elas sejam uma forma possível de serem utilizadas para distinguir dois *websites* e, estendendo um pouco, há também a possibilidade de, dada uma URL, dizer se a mesma se classifica em um grupo de URLs com características em comum. Assim sendo, estendendo-se ainda mais essa visão sobre as localizadoras, tem-se que URLs maliciosas

podem ter características específicas que denotem que são efetivamente deste tipo, e tais características podem vir a se apresentar com certa frequência em localizadoras maliciosas distintas (ESHETE, 2013). Desta forma, ao avaliar uma URL levando em conta estas características específicas, podemos classificá-la no que tange a sua “índole”.

1.3 Inteligência Artificial e problemas de classificação

A Inteligência Artificial (IA) é um campo relativamente novo das ciências e engenharias, tendo seus primeiros estudos iniciados por volta do final da Segunda Guerra Mundial. Atualmente é uma área que está em foco, especialmente na mídia como um todo e entre pesquisadores dos mais diferentes campos. A IA possui diversas linhas de desenvolvimento, que culminam em subcampos diferentes dentro da própria disciplina, cada qual, com suas especificações e formas de trabalho (NORVIG, PETER & RUSSEL, STUART J., 2010).

Podemos definir IA como sendo a arte de criar máquinas que realizam funções que requerem inteligência quando efetuadas por seres humanos (KURZWEIL et al., 1990). Nestes termos, podemos elucidar que aplicações em IA são desenvolvidas tendo como seu propósito a resolução de problemas para os quais uma solução utilizando a cognição humana é necessária.

Um tipo de problema bastante comum no ramo da IA é o de se discernir uma entidade baseados em grupos distintos de entidades. Um problema onde se pretende determinar a qual grupo, dentre um número finito e determinado de grupos, um elemento específico melhor se ajusta é definido como problema de classificação (MÜLLER; GUIDO et al., 2016).

A análise e classificação de informações textuais não é um campo dos mais recentes dentro da abrangência das áreas da Computação e, em adesão ao que é apresentado por Ikonomakis, Kotsiantis e Tampakas (2005), são uma importante área de pesquisa. Prova disso é que um exemplo de uso bastante comum de IA é o de trabalhos como Manning et al. (2014) e Berger, Pietra e Pietra (1996), na área de Processamento de Linguagem Natural, que de certa forma, é uma ascendência dos conceitos de processamento textual. Dada a natureza literal das URLs, é bastante viável a construção de analisadores para as mesmas utilizando-se de técnicas de IA, visto que outros trabalhos também exploraram tal área. Um exemplo disto é Urcuqui et al. (2017), que explorou a aplicação de análise de dados através de um sistema empregando técnicas de IA para classificação de URLs.

No ramo da Segurança da Informação como um todo, o uso de técnicas baseadas em IA também é bastante costumeiro. A saber, o trabalho Sinclair, Pierce e Matzner (1999) considerou o uso de algumas destas técnicas para a detecção de intrusão em redes de computadores, enquanto que Sahs e Khan (2012) serviu-se de IA para detecção de softwares

mal intencionados em plataformas mobile. Estas sendo apenas duas das aplicações dentro de Segurança da Informação que vêm a empregar IA.

1.4 Problema

Nestes termos, o problema que é passível de percepção é a falta de segurança e ameaças que um usuário pode vir a encontrar ao acessar *websites*. Sendo assim, tal problema será abordado no aspecto referente ao de classificação de URLs, onde através de características inerentes à própria URL será determinado se a mesma representa ou não alguma ameaça. Por ameaça, engloba-se aqui uma variedade grande de utilizações prejudiciais de *websites*.

Esta classificação deve ser efetuada de forma à detectar, com alto nível de precisão, URLs maliciosas ou legítimas. É de interesse ainda, detectar as características de uma URL que mais sejam passíveis de indicar que a mesma represente algum risco ao usuário. Por fim, é importante também ter atenção quanto à necessidade de obtenção de tais características de cada URL em uma forma dinâmica, visto que o ambiente da aplicação possui uma tendência a mudanças.

1.5 Proposta do trabalho

Uma das principais motivações para o desenvolvimento e aplicação de técnicas de IA para a Segurança da Informação é a onda crescente de ameaças que afloram sobre usuários no ambiente digital (fortemente demonstrada pelas produções Goodman (2015) e Mazurczyk e Caviglione (2015)). Muitas das vezes tais ameaças exigem uma análise dinâmica e constante da situação, um espaço ideal para aplicação de IA.

Dentre tais ameaças, o uso de URLs maliciosas pode ser percebido como uma das mais graves e recorrentes. O mesmo consiste em uma ferramenta de alta usabilidade para os criminosos, pois além da facilidade com a qual um atacante pode diligenciar tal ameaça, fatores como a alta capacidade de divulgação da mesma contribuem para seu sucesso. A proteção para ataques envolvendo este tipo de tecnologia exige ainda, dinamismo e formalismo (Seção 1.2), dada a natureza do problema. Existem alguns sistemas, como por exemplo o Google SafeBrowsing ⁶ e o Llama (VIGNA, 2014) que oferecem tal proteção, porém, sendo fechadas e fornecendo pouco ou nenhum detalhamento quanto a sua implementação. Há também sistemas abertos, que operam de forma a classificar URLs utilizando de técnicas similares e que possibilitam o alcance à sua implementação.

O objetivo primário deste trabalho se define como o de avaliar e demonstrar as capacidades de técnicas de Inteligência Artificial para classificar URLs, através da

⁶ Detalhamento em: <<https://safebrowsing.google.com/>>

formulação e implementação de uma ferramenta capaz de efetivamente utilizar-se destas técnicas, com a finalidade de obter um resultado suficientemente preciso. Neste trabalho serão apresentados os métodos que foram utilizados, bem como, as especificidades da implementação da dita ferramenta.

Como um objetivo secundário, teríamos a comparação dos resultados obtidos através do uso da ferramenta, com as diferentes técnicas de Inteligência Artificial implementadas na mesma. Tem-se ainda a apresentação dos aspectos referentes à modelagem do problema, bem como, conceitos necessários para a sua compreensão.

2 Fundamentação teórica

Neste capítulo são abordados os conceitos, terminologias e princípios básicos necessários à compreensão do conteúdo. Cada tópico está especificado a seguir, abordando áreas específicas. Este capítulo também busca proporcionar ao leitor um ganho a mais de conhecimento no contexto do que foi abordado. Além dos conceitos mais técnicos como os referentes a segurança e redes de computadores, apresentamos também alguma fundamentação teórica, embasada na grande área da matemática.

2.1 Redes de computadores, WWW e URLs

Uma rede, no contexto computacional, pode ser definida formalmente como um conjunto de dispositivos que se encontram conectados entre si, a fim de realizar a disposição de funcionalidades uns para os outros. As redes de computadores podem ter diversas formas (topologias) e agrupar diferentes quantidades de dispositivos, isto é claro, obedecendo a certos limites. É possível a implementação de diversas funcionalidades sobre uma rede, bastando para isto seguir algumas convenções pré-estabelecidas a fim de manter a rede operando com sucesso e facilitar sua integração com outros sistemas. Destas convenções, surge o modelo de serviços (KUROSE; ROSS, 2013) e também o chamado modelo Open System Interconnection (OSI).

O modelo de serviços, também chamado de TCP/IP é um modelo baseado em camadas, onde cada camada representa um nível de abstração para a comunicação em rede. Geralmente, a mesma é representada com suas camadas em uma estrutura semelhante a uma pilha (*stack*). Esta organização em camadas nos permite como vantagem a modularização, pois desta maneira nos é permitido modificações nas camadas de forma que as mesmas não afetem as demais, acima ou abaixo na pilha. As camadas deste modelo são as de aplicação, de transporte, de rede e de interface. Uma representação mais clara é a da Figura 1.

O modelo OSI é um modelo também em camadas que serve de base para diversas aplicações em rede. Surgido na década de 1970, o mesmo segue até hoje como uma das bases que desenvolvedores utilizam. Este se diferencia do modelo de serviços dado que acrescenta duas outras camadas, a de apresentação e a de sessão, além de subdividir a camada mais inferior. No entanto, como ressalva, não é porque o modelo de serviços não possui tais camadas que o mesmo é de alguma forma insatisfatório. Na verdade, o que se assume aqui é que seja de responsabilidade do desenvolvedor implementar as características destas “camadas extras” em sua aplicação que utilize do modelo de serviços (KUROSE; ROSS, 2013). Uma visualização mais conveniente consta na Figura 2.

A grosso modo então, as camadas nos dois tipos de modelo são equivalentes, salvo que as três camadas mais altas no modelo OSI são representadas no modelo de serviços

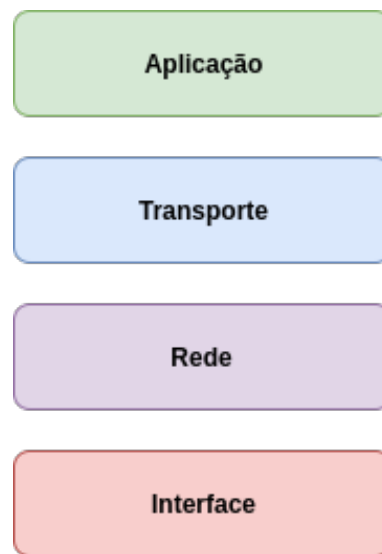


Figura 1 – Camadas no modelo TCP\IP.

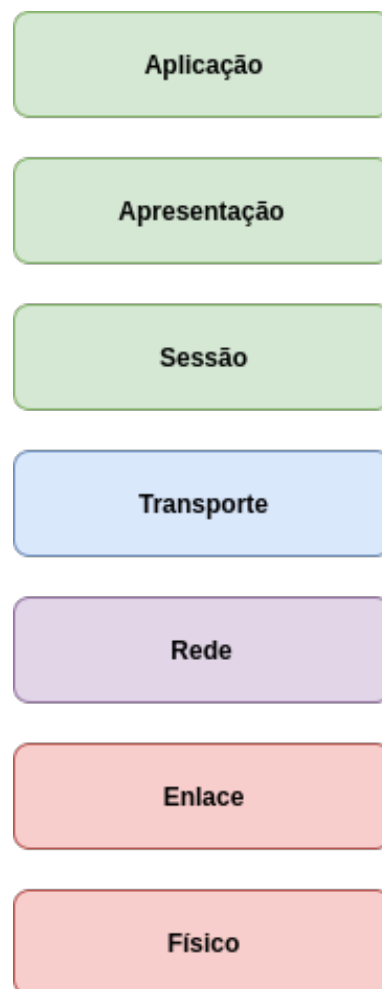


Figura 2 – Camadas de rede no modelo OSI.

unicamente pela camada de aplicação e, também, a divisão na camada mais próxima ao hardware. Enquanto o modelo OSI especifica de uma forma mais teórica as funcionalidades de cada camada da rede, o modelo de serviços é apresentado como mais prático, sendo composto em cada uma de suas camadas por diferentes protocolos independentes que podem ser combinados conforme a necessidade (FOROUZAN, 2010).

Visto que este trabalho é embasado sobre conceitos acerca da WWW e as aplicações executando sobre a mesma (como *websites*), para uma compreensão clara do conteúdo, é coerente que o mesmo defina de forma mais específica conceitos referentes a camada de aplicação. Esta camada é definida por KUROSE e ROSS (2013, p. 37) da seguinte forma:

A camada de aplicação é onde residem aplicações de rede e seus protocolos. [...] Certas funções de rede, como a tradução de nomes fáceis de entender, que são dados a sistemas finais da Internet, [...] também são executadas com a ajuda de um protocolo de camada de aplicação.

2.1.1 A Internet e a *web*

Segundo a definição: “A Internet é uma rede de computadores que interconecta centenas de milhões de dispositivos de computação ao redor do mundo” (KUROSE; ROSS, 2013, p. 3). Dada a característica global da mesma, a mesma comumente é associada a uma entidade única, o que não é de todo errado mas se distingue da realidade.

Os pilares do desenvolvimento da mesma remontam à década de 1960 nos Estados Unidos da América, com o surgimento do que ficou conhecido como comutação de pacotes. Basicamente, pacotes são unidades mínimas para transferência de informação e a comutação de pacotes é o uso de tais pacotes com o intuito de enviar informação de um dispositivo computacional para outro. De início utilizava-se da infraestrutura da rede telefônica, extensa e abundante na época, o que culminou na criação da ARPANet, instalada inicialmente em instituições de pesquisa. A ARPANet é considerada como a precursora evidente das redes e da Internet moderna (KUROSE; ROSS, 2013).

Com a chegada das redes ao mundo corporativo (em especial graças aos esforços de pesquisadores das companhias IBM e General Electric), por volta da década de 1970 e o início modesto da presença de redes domésticas em 1980, esta área de desenvolvimento começou a ganhar bastante impulso e atenção (KUROSE; ROSS, 2013). Então, em meados da década de 1990, surge a WWW. A WWW é uma invenção de Tim Berners-Lee, então pesquisador do European Center for Nuclear Physics, o CERN. A mesma tinha a finalidade de prover a pesquisadores espalhados pelo mundo uma forma de transferirem informações sobre seus trabalhos de forma simples e eficiente. Tim Berners-Lee (2000, p. 4) ainda afirma que sua visão era da seguinte maneira:

Supondo que toda e qualquer informação guardadas em computadores no mundo todo estivesse ligada e supondo que eu pudesse programar meu computador para criar um “espaço” aonde qualquer coisa pudesse estar ligada a outra. Todos os bits de informação em todos os computadores do CERN e no mundo inteiro estariam disponíveis para mim e para todas as pessoas. Haveria um espaço global de informação único. Uma vez que um bit daquela informação estivesse sendo representado através de um endereço, eu poderia solicitar a meu computador que obtivesse o mesmo.

Para realizar tudo isso, Berners-Lee criou o HTML, sigla para Hypertext Markup Language (se traduzindo como linguagem de marcação para hipertexto), que se trata de uma linguagem de marcação que permite a criação de documentos em hipertexto, e o HTTP Hypertext Transfer Protocol (protocolo de transferência de hipertexto) que age como um regente sobre a forma como a comunicação na WWW é efetuada. Hipertexto diz respeito a uma especificação de documentos com a capacidade de se interligarem com outros aos quais façam alguma forma de referência ou que sejam de alguma forma complementares. O prefixo “hiper” aqui assume um sentido muito parecido com aquele que possui em ramos da matemática tal qual a geometria, como algo representando uma “dimensão além”. O primeiro a descrever um modelo base para o hipertexto, inicialmente atrelado a um dispositivo, foi Bush et al. (1945, p. 98, Tradução nossa), que escreveu:

Considere um dispositivo futuro para uso individual, que seria um tipo de biblioteca e arquivo privado mecanizado. É preciso um nome, e, para criar um aleatoriamente, “memex” basta. Um *memex* é um dispositivo em que um indivíduo guarda todos seus livros, anotações e comunicações, e que é mecanizado, podendo ser consultado com alta velocidade e flexibilidade. É um grande suplemento íntimo para sua memória. [...] Ele é capaz ainda de proporcionar um passo imediato, no entanto, para indexação associativa, a ideia básica de que há uma disposição em que qualquer item pode ser induzido a vontade para selecionar imediatamente e automaticamente outro.

A estrutura que define a forma como as aplicações na WWW operam se baseia no modelo cliente-servidor. Este modelo é conhecido como um paradigma de redes e se baseia na premissa de que o cliente envia ao servidor uma solicitação de algum serviço e o servidor possui a responsabilidade de responder (servir) a esta solicitação do cliente (FOROUZAN, 2010). Ambos cliente e servidor são entidades computacionais, neste caso, executando processos específicos que permitem esta comunicação. Para operar utilizando deste paradigma, é necessário além da conexão entre as máquinas envolvidas, uma forma de identificação para as mesmas. Nesta denominação, cliente e servidor assumem o nome de hospedeiros (*hosts*) ou sistemas finais.

Para que seja possível a comunicação entre dois *hosts* diferentes, é necessário que entre os mesmos subsista um caminho. Este caminho, outrora sendo a já citada infraestrutura telefônica, hoje é construído (por assim dizer) através do processo de roteamento. Este é definido como sendo o processo de criação de rotas, que por sua vez, se resume a determinação pela camada de rede de um caminho entre o remetente e o destinatário, fazendo uso de algoritmos específicos (KUROSE; ROSS, 2013). Se tratando especificamente de um exemplo mais visual, o mesmo é observável na Figura 3.

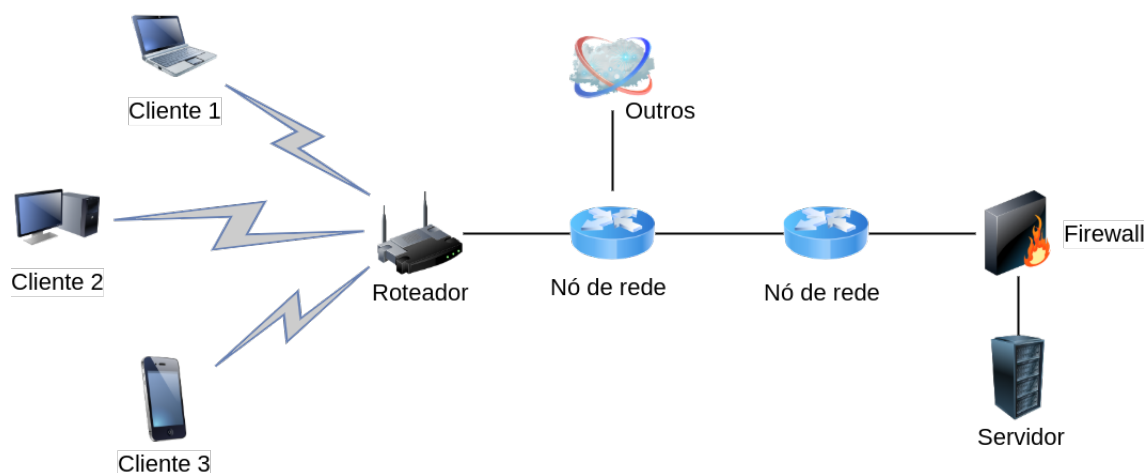


Figura 3 – Exemplificação de infraestrutura no modelo cliente-servidor.

É interessante apresentar o conceito de protocolo para redes de computadores. A palavra protocolo se define como um conjunto pré-estabelecido de regras e normas para uma comunicação boa e eficaz. Na verdade, se tratando do mundo real, um exemplo bem prático seria uma conversa entre duas pessoas que acabaram de se conhecer. A boa educação e a gentileza regem que, antes de tudo deve haver uma apresentação de ambas as partes envolvidas no diálogo e logo a seguir, a demonstração de intenções, por exemplo, seguida de uma troca de informações amistosa. Ao final, também é interessante que ambos se "despeçam" um do outro educadamente. Paralelamente a isto para o caso de uma rede de computador tem-se a definição de protocolo, segundo KUROSE e ROSS (2013, p. 7): “Um protocolo define o formato e a ordem das mensagens trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem ou outro evento”.

2.1.2 URLs

As URLs são parte essencial da WWW, visto que a mesma se baseia no modo de operação de um espaço de recursos único (Subseção 2.1.1), é necessário ser capaz de identificarmos os recursos acessíveis pelo mesmo. Em outras palavras, um usuário tentando acessar algum recurso disponível via *web* precisa do URL de tal recurso. Segundo Forouzan (2010, p. 853): “A URL [...] é um padrão para a especificação de qualquer tipo de

informação na Internet. Uma URL é constituída por quatro partes: protocolo, *host*, porta e caminho [...]. Ainda, no contexto deste trabalho, é necessário definir a URL em termos de cada uma de suas partes, para proporcionar uma melhor compreensão do processo de análise da mesma. É interessante, para isto, referir-se à Figura 4.

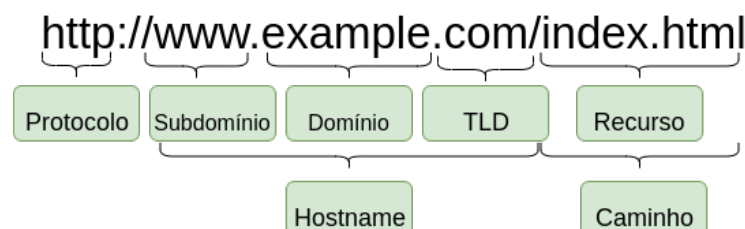


Figura 4 – Divisões de uma URL.

Como já foi referido, tal qual uma representação textual comum, uma URL é munida de características quanto a sintaxe e semântica. As partes a seguir denotam subdivisões no que tange o aspecto sintático de uma URL. Para as características semânticas, é necessário que se abstraia a significância de tais partes por meio de regras específicas, por exemplo.

O protocolo de uma URL diz respeito ao tipo do protocolo de rede que irá ser utilizado para a requisição representada por determinada localizadora. Nas URLs de uso para navegação em *websites* geralmente predominam os protocolos HTTP e HTTPS. É interessante notar ainda que um mesmo servidor pode atuar oferecendo diversos protocolos para um mesmo serviço *web*, podendo inclusive servir um mesmo recurso em protocolos distintos (FOROUZAN, 2010).

Já o *host*, também denominado nome de domínio, nome de recurso ou *hostname* diz respeito ao computador dentro da rede onde as informações estão armazenadas. É comum que computadores armazenando-as recebam nomes alternativos (também chamados “alias”), começando com *www* ou algum outro prefixo, apesar disto não ser obrigatório. Um *host* representando um computador, que por sua vez está hospedando uma página *web* pode assumir qualquer nome desde que o mesmo seja representável (FOROUZAN, 2010). Notasse no entanto que esta nomenclatura “host” difere em atribuição da mesma nomenclatura quando tratamos do modelo cliente-servidor.

O prefixo “*www*” representa um subdomínio, enquanto que o sufixo “*com*” do *host*, representa um outro domínio que é denominado Top Level Domain (TLD). O subdomínio nada mais é que uma divisão do domínio, podendo ser disposta para um fim específico, para a possibilidade de organizar a navegação sobre o site (FOROUZAN, 2010). Para o caso do exemplo, o “*www*” significaria que determinada URL responderia comumente por um *website*. Se tratando dos TLDs é dito que os mesmos são os de mais alta hierarquia de consulta, que servem para organizar a busca de domínios através do Domain Name System (DNS). Os mesmos operam como se fossem intermediários

de delegação, que agrupam servidores que, por sua vez, agrupam outros servidores que possuem características em comum na sua funcionalidade (KUROSE; ROSS, 2013). Um TLD pode ainda ter subdivisões específicas, por conta de denominações de cada país, por exemplo.

Uma URL pode ainda conter uma especificação para o número da porta do *host* que a mesma representa. Isto é feito pois serviços diferentes executam sobre portas diferentes (KUROSE; ROSS, 2013). O mais comum, no entanto, é que o número da porta não seja especificado, considerado assim o acesso “padrão”, devidamente configurado no servidor. A última parte a ser descrita diz respeito ao caminho (*path*) que nada mais é que o endereço específico do recurso, dentro do servidor, que está sendo acessado. O caminho possui uma estrutura bastante similar a de diretórios e subdiretórios em um sistema operacional e dependendo da implementação, subsiste literalmente na forma de diretórios (FOROUZAN, 2010). O caminho pode ainda incluir a extensão de um arquivo sendo acessado ou também, parâmetros passados através de uma requisição.

2.1.3 HTTPS

Em vias de tornar a navegação na *web* mais segura, foi criado o protocolo HTTPS, que diz respeito ao protocolo HTTP combinado com a utilização de protocolos de segurança específicos. O mesmo é baseado em certificados, onde autoridades certificadoras específicas os emitem (RESCORLA, 2000). O uso de tal protocolo permite o estabelecimento de uma conexão dita segura, além da indicação no cliente do uso do protocolo em si. Um problema que advém da confiança exclusiva neste tipo de protocolo diz respeito ao fato de que qualquer um pode vir a adquirir um certificado válido, e sendo assim, a indicação no cliente não fornece um panorama completo. Em resumo, o uso do HTTPS pode indicar que a conexão se dará de uma forma segura, porém, não indica que uma página em si é segura para o usuário.

2.1.4 Protocolo TCP

Também é pertinente definir mesmo que de forma superficial, o funcionamento do Transmission Control Protocol (TCP), um protocolo de camada de transporte. Uma boa razão pra isso é que o HTTP é dito rodar sobre o TCP, este último regendo questões como ciclos de troca de mensagens, por exemplo. Ainda, segundo KUROSE e ROSS (2013, p. 73): “O HTTP usa o TCP como seu protocolo de transporte subjacente [...]. O cliente HTTP primeiro inicia uma conexão TCP com o servidor. Uma vez estabelecida, os processos do navegador e do servidor acessam o TCP por meio de suas interfaces de socket”. Uma ilustração a este processo se encontra na Figura 5.

O TCP é um serviço orientado a conexão. Isto significa que os *hosts* envolvidos

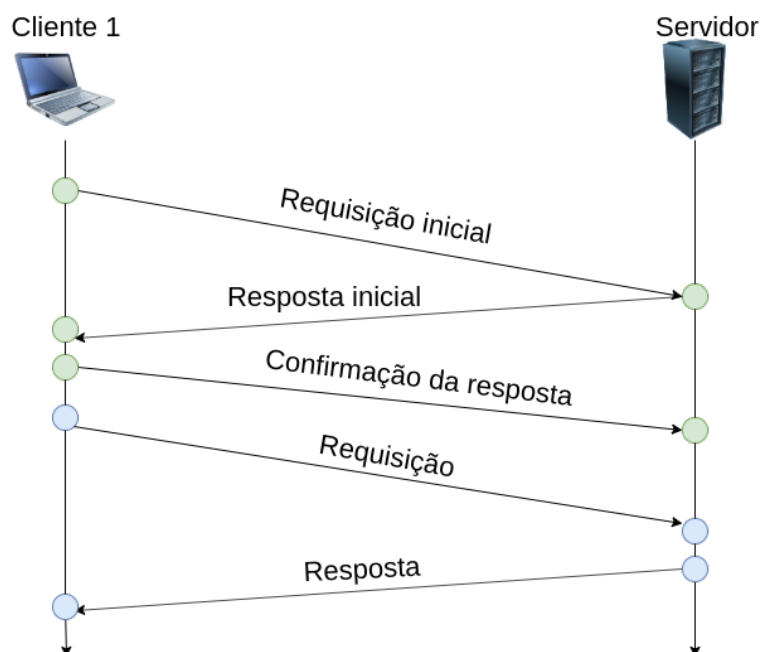


Figura 5 – Exemplo de comunicação através do TCP. É perceptível que as marcações laterais indicam o fluxo temporal. Para fins de exemplificação, o conteúdo da comunicação foi representado de forma genérica.

na troca de pacotes estabelecem normas antes que as mensagens de camada de aplicação comecem a fluir. Além disso, o TCP é dito como um serviço confiável da camada de transporte, significando que os processos envolvidos na comunicação por TCP podem confiar na recepção das mensagens sem erros e organizadas corretamente. Este protocolo é muitas vezes citado por fazer o máximo possível para garantir a troca de mensagens (KUROSE; ROSS, 2013).

Tão importante quanto o TCP, existe também o Internet Protocol, o IP. O mesmo é um protocolo que se encontra na camada de rede, implementado para compor um protocolo complementar ao TCP sendo que ambos, muitas das vezes, são referenciados como TCP/IP. O IP é responsável por especificar o formato dos pacotes que são enviados e recebidos entre roteadores e *hosts* (KUROSE; ROSS, 2013). O IP assume a forma de um conjunto de regras que engloba maneiras de manusear os pacotes na camada de rede, o formato de um datagrama (que é o nome assumido por um pacote na camada de rede, quando acrescido de mais algumas informações (PETERSON; DAVIE, 2007)) e regras de endereçamento. Para este trabalho, o foco maior é dado as regras de endereçamento, haja visto que um detalhamento total e completo sobre o funcionamento do protocolo IP não é necessário para compreensão dos desenvolvimentos desta monografia.

2.1.5 Endereçamento e DNS

De acordo com KUROSE e ROSS (2013, p. 95):

Assim como seres humanos podem ser identificados de muitas maneiras, o mesmo acontece com hospedeiros da Internet. [...] Nomes de hospedeiro [...] são fáceis de lembrar e, portanto, apreciados pelos seres humanos.

Como nomes de hospedeiro são representáveis simplesmente com palavras, isto torna fácil memorizar, inferir ou mesmo transmitir tais endereços. Ainda, ao se utilizar serviços de busca, isto fica evidenciado de forma bem clara, pois bastam que sejam digitadas palavras comuns àquele *website* para que o mesmo, provavelmente, seja encontrado. No entanto, a nível computacional, estes nomes não são muito práticos. Isto acontece por diversos fatores, como o a necessidade de processamento exigida e também, que tais nomes não trazem muita informação no tocante a localização de um *host* (servidor). Por essas e outras, hospedeiros também são identificados de outra forma: através do endereço IP (KUROSE; ROSS, 2013).

Um endereço IP é uma sequência composta por elementos representáveis em bits. Atualmente, duas versões se mantêm em uso: O IPV4, o mais comum, que representa os endereços com uma codificação de 32 bits (Figura 6), e o IPV6 (futuramente é pretendido como substituto ao IPV4), que utiliza 128 bits (KUROSE; ROSS, 2013). Para o caso do IPV4, sua codificação pode ser escrita com quatro octetos (bytes), onde cada octeto é um número decimal variando entre 2^8 possibilidades. Esta notação é comumente separada por “pontos”. No entanto, a atribuição de tais endereços não é “aleatória”, por assim dizer. Na realidade, a mesma segue definições rígidas atribuídas e formalizadas pelos órgãos de regimento da Internet.

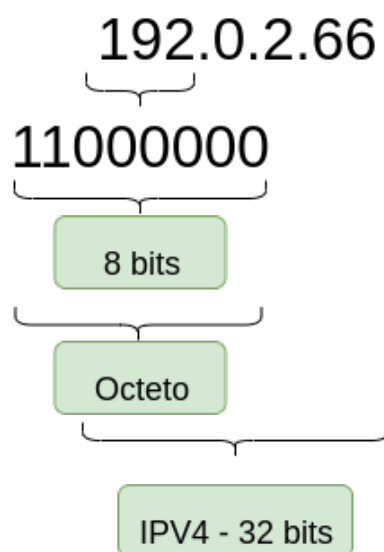


Figura 6 – Endereço IPV4.

Salvo casos de exceções para usos específicos, como o caso de alguns serviços de hospedagem compartilhados por exemplo, um endereço IP deve ser globalmente único.

Isto significa, para o caso de *websites*, que o mesmo deve servir para identificar um único *host* que atua como servidor (KUROSE; ROSS, 2013). Cada “parte” de um IP (octeto) pode ser utilizada para identificar uma rede ou subrede da qual o *host* que responde por aquele IP faça parte. Para que um computador seja capaz de identificar um *website* quando seu operador tenta acessá-lo com o nome do hospedeiro, é necessário um processo de “tradução”. Para tanto, um outro protocolo é estabelecido, conhecido como o DNS.

O DNS pode ser entendido como um banco de dados distribuído e hierárquico, composto por diversos servidores em níveis diferentes de hierarquia. O DNS também pode ser definido como um protocolo de camada de aplicação, responsável pelas consultas a tal banco de dados distribuído, oferecendo a capacidade de tradução de nomes em endereços IP (KUROSE; ROSS, 2013). Este banco de dados constitui-se de forma similar a uma “lista telefônica”, de modo que cada registro de um domínio (nome) se relaciona a um endereço IP, daí a ideia de tradução. O chamado servidor de nomes é o componente responsável por atender requisições de tradução (solução) de nome. Para que um *website* seja efetivamente passível desta tradução, é necessário seu registro em algum órgão de vínculo com a Internet Assigned Numbers Authority (IANA), uma subsidiária do ICANN. Muitas vezes este servidor de nomes é referido como “servidor DNS” (KUROSE; ROSS, 2013).

O processo de tradução utiliza-se de consultas a servidores espalhados globalmente, onde nenhum destes possui todos os mapeamentos de hospedeiros. No processo de uma consulta a um servidor DNS, a aplicação cliente realiza uma requisição que é repassada por servidores até que a tradução necessária seja encontrada (Figura 7). Como já foi especificado, o DNS é organizado de forma hierárquica. Há três níveis compondo esta hierarquia: os servidores DNS raiz (de mais alto nível, responsáveis geralmente por repassar para servidores mais específicos), os servidores DNS de TLD (para domínios de alto nível) e os servidores DNS autoritativos (geralmente atrelados à organizações) (KUROSE; ROSS, 2013).

Exemplificando o caso de uma consulta comum, um cliente solicitaria uma tradução a um servidor raiz, que por sua vez lhe encaminharia o endereço IP do servidor DNS de TLD responsável. Em seguida, o cliente novamente contataria um servidor, desta vez, o de TLD obtido. Este responderia com o IP do servidor autoritativo que efetivamente é responsável por aquele endereço desejado. Por fim, o servidor autoritativo é contatado e responde com o IP do servidor hospedeiro desejado. Isto tudo é claro, ocorrendo em uma velocidade extrema. Uma demonstração da característica hierárquica das consultas DNS se encontra na Figura 7.

No escopo da Internet, sistemas inteiros ou coleções de sistemas sobre uma autoridade comum podem vir a representar um agrupamento autônomo. Os chamados Autonomous System Numbers (ASNs) são definidos como identificadores para tais grupos

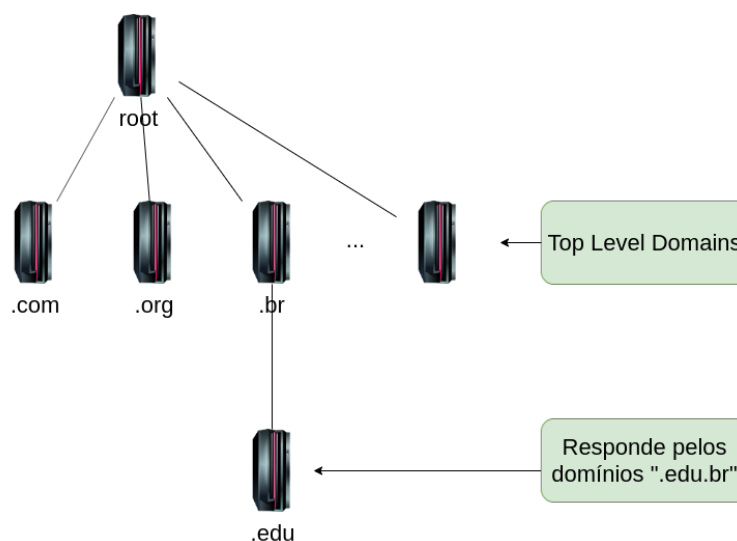


Figura 7 – Demonstração da hierarquia de consulta de servidores DNS.

autônomos (HUSTON, 2006). Consultas a um servidor DNS podem trazer, além de informações do domínio em si, informações acerca de algum ASN.

Através do servidor de nomes, é possível a obtenção de diversos dados acerca de um *website*. Analisando exclusivamente os dados de propriedades provenientes de uma consulta a um servidor DNS, também é possível a caracterização de *websites* maliciosos, conceito explorado com êxito por Bilge et al. (2011). Ainda, é aceitável também a utilização das propriedades de uma consulta DNS na companhia de informações específicas da URL para a sua efetiva caracterização.

2.1.6 WHOIS

Um outro serviço proveniente da WWW é denominado WHOIS. O mesmo é baseado no protocolo TCP, porém, sua utilização é específica para a obtenção de informações em formato passível de leitura por um ser humano. Originalmente utilizado para prover serviços de contato com administradores de sistemas e informações a respeito de domínios registrados, atualmente dependendo da implementação, o mesmo pode agrupar uma série de outras funcionalidades (DAIGLE, 2004). O WHOIS pode, por exemplo, ser usado para consulta direta de informações acerca dos responsáveis pelo registro de um domínio, sobre os servidores de nomes que o atendem e outras mais.

O funcionamento do WHOIS também é fundamentado segundo o modelo cliente-servidor. As requisições e respostas são transmitidas como texto puro, o que facilita de certa forma sua análise (DAIGLE, 2004). A aplicação cliente do protocolo WHOIS é disponibilizada por padrão em diversos sistemas operacionais.

Ainda se tratando de serviços, um outro que é proveniente da camada de aplicação é o GEOIP. O GEOIP é um serviço que opera sobre uma base de dados, fornecendo

consultas para se recuperar informações de localização sobre um endereço IP. Estas informações podem incluir o país de origem, região e cidade do mesmo, bem como, outras informações. Os dados são dependentes do fornecedor da base em questão (CANALI et al., 2011). A implementação de tais sistemas é possível graças a forma como os endereços IP são atribuídos globalmente (Subseção 2.1.5).

2.2 Segurança da Informação

Nesta seção, serão abordados conceitos básicos de segurança da informação, bem como, a explanação de algumas terminologias da área. A segurança da informação não é um conceito restrito a sistemas na *web*. Tampouco é um conceito restrito somente a computadores ou a pessoas envolvidas diretamente com os mesmos. Na realidade, a segurança da informação tende a envolver não só o aspecto tecnológico, como também, o humano e o organizacional.

2.2.1 *Malwares*

Um *malware*, palavra que deriva da expressão *malicious software*, se define como uma entidade que se enquadra em toda a gama de softwares que operam de forma a causar danos intencionais a um usuário, dispositivo computacional ou rede (SIKORSKI; HONIG, 2012). *Malwares* são altamente prejudiciais a usuários e a sistemas, este prejuízo variando conforme a sua programação. *Malwares* podem ser utilizados para a obtenção de informações pessoais do indivíduo, por exemplo, ou também para assumir o controle de redes inteiras.

O processo de se estudar *malwares* e seu comportamento é definido como análise de *malware*, sendo considerada uma importante subárea da segurança da informação. Também é possível definir a mesma como a arte da dissecação de programas maliciosos, a fim de entender seu funcionamento, como identificá-lo e também eliminá-lo (SIKORSKI; HONIG, 2012). Conforme a definição de Elisan (2015, p. 4, Tradução nossa):

Análise de malware é o processo de extração de informação de malware através de inspeção estática e dinâmica pelo uso de diferentes ferramentas, técnicas e processos. É uma aproximação metódica a fim de revelar a diretiva principal de um malware através da extração da maior quantidade possível de dados sobre um malware enquanto esta ativo ou inativo.

Um *malware*, tal qual um patógeno em biologia, precisa de um vetor de ataque para ser capaz de atingir seu alvo (CRANDALL et al., 2009). URLs maliciosas na WWW, o foco deste trabalho, são um exemplo destes vetores. Uma outra característica de *malwares* é que os mesmos podem fazer uso de URLs de outra forma, que é o caso onde uma

URL é responsável por algum aspecto no código da aplicação maliciosa. Em épocas mais recentes, o surgimento das chamadas Malware Distribution Networks, MDNs, tem levado ao reascender de preocupações e desenvolvimentos na área. O termo diz respeito a um conglomerado de páginas, servidores e pontos de redirecionamento criminosos, geralmente fornecidos como um serviço à aqueles que efetuam ataques baseados em *malware* (WANG et al., 2013).

Para concluir este segmento, uma descrição resumida acerca dos tipos de *malware* pode ser encontrada no Apêndice A.

2.2.2 Engenharia social

A engenharia social diz respeito ao uso de técnicas específicas para a manipulação de indivíduos, a fim de se obter algum tipo de vantagem. No ramo da segurança da informação, tais técnicas são de ampla notoriedade e usabilidade tanto por profissionais da área quanto por criminosos que buscam prejudicar sistemas ou indivíduos. A eficiência dos ataques baseados em engenharia social se apresenta quando pessoas não seguem boas práticas de segurança (MITNICK; SIMON, 2011).

Para se efetuar um ataque baseado em engenharia social, é necessário que este atacante possua tanto conhecimentos de comportamento e psicologia quanto prática em discurso, sem omitir também os conhecimentos acerca da WWW, quando este ataque se efetua em vias da *web*. Por não se tratar de uma exclusividade de disciplinas da computação, a engenharia social pode ser simplesmente demonstrada de uma forma razoável com a observação de situações cotidianas. Conforme enunciado por Hadnagy (2010, p. 32, Tradução nossa):

A engenharia social é usada diariamente por pessoas comuns, em situações do cotidiano. Uma criança tentando ir a uma loja de doces ou um empregado procurando conseguir um aumento estão fazendo uso de engenharia social. Engenharia social acontece em governos ou no marketing de pequenos negócios. Infelizmente, ela também está presente quando criminosos [...] enganam pessoas para obter informações que as tornem vulneráveis à crimes.

No contexto de URLs maliciosas, um exemplo de forma de uso da engenharia social para algum ataque reside, praticamente de forma certa, na elaboração de um modo de se atrair um usuário da WWW a acessar algum link malicioso. Para tanto, diversos conceitos podem ser empregados, incluindo mas não se limitando a utilização de conhecimentos prévios sobre a vítima, oferecer propostas falsas de alto valor e também, técnicas de extorsão.

2.2.3 Ataques envolvendo URLs

Para que seja possível a compreensão da real funcionalidade dos desenvolvimentos deste trabalho, é necessária a especificação dos usos de URLs em práticas criminosas na *web*. Ademais, segundo Choi, Zhu e Lee (2011, p. 125, Tradução nossa):

“A identificação de tipos de ataque é útil pois o conhecimento da natureza de uma ameaça em potencial nos permite ter uma reação apropriada, bem como o uso de uma contramedida eficiente contra tal ameaça”.

Um dos usos mais comuns dado a URLs no contexto de ataques diz respeito ao *phishing*. *Phishing* pode ser definido da seguinte forma, de acordo com Jagatic et al. (2007, p. 1, Tradução nossa): “Phishing é uma forma de engenharia social na qual um atacante tenta adquirir de forma fraudulenta informação sensível de uma vítima ao personificar uma terceira parte que seja dita confiável”. Um exemplo clássico de phishing pode ser descrito ao se analisar a situação de um ataque onde há uma “cópia” do design de uma rede social, hospedada em um servidor caseiro, para obtenção de informações de login de usuários.

Para aumentar a eficiência de seu ataque, muitos cibercriminosos ao praticar *phishing*, utilizam URLs registradas com domínios muito similares aos das páginas as quais se pretende obter informações pelo ataque (BOSE; LEUNG, 2007). Com a facilidade no uso de ferramentas de obtenção e manipulação de hipertexto e desenvolvimento *web*, tal forma de ataque pode ser efetuada até mesmo por pessoas não tão proficientes em cibercrimes.

URLs também podem ser utilizados como vias de circulação de *malwares*. Páginas que hospedam *malwares* ou que fornecem arquivos com alguma forma de infecção, por exemplo, não são raridade. Um ataque conhecido como *drive-by download* se define como um download de software que acontece sem o conhecimento ou consentimento do usuário. Em um ataque “comum”, esta técnica permite que o usuário se infecte meramente por visitar o *website* malicioso (EGELE et al., 2009).

Os *drive-by downloads* são uma técnica bastante explorada, juntamente com a exploração direta de falhas em navegadores. Os *drive-by downloads* também possibilitam a exploração de falhas em *plugins* de navegadores, o que aumenta ainda mais o risco deste tipo de ameaça (PROVOS et al., 2007). Ainda sobre os *drive-by downloads*, de acordo com o que é elucidado por Egele et al. (2009, p. 96, Tradução nossa):

O código malicioso advindo de tais ataques tipicamente tem a capacidade de assumir total controle da máquina da vítima. Geralmente teclas pressionadas são gravadas, senhas podem ser roubadas e pode ocorrer o vazamento de informações sigilosas. Há também,

a possibilidade de que a máquina da vítima se junte a uma botnet [...].

Existe também a categoria das URLs de prática de *spamming*. Esta prática, que pode ser entendida como o ato de associado à *spams*, pode ser definido como ações deliberadas para se dar uma importância injustificável para alguma página na *web*, considerando o real valor daquela página (GYONGYI; GARCIA-MOLINA, 2005). O spam pode assumir diversas formas, cada qual com níveis distintos de periculosidade. Como exemplo, é citado o uso de *spamming* na divulgação de boatos, venda de produtos ou divulgação de fraudes online.

2.2.4 Abordagens em segurança para o problema de URLs maliciosas

Dado que o problema do uso de URLs para fins maliciosos já é de longa data, foram desenvolvidas diversas formas de se combater o mesmo. Para o combater a tal problema, é necessária antes a detecção de uma URL como sendo, realmente, incumbida de alguma atividade maliciosa. Consequentemente, é factível que existam diversas formas de se detectar quanto ao fato de uma URL ser maliciosa ou não. Sobre isto, Sahoo, Liu e Hoi (2017) propõe o agrupamento de tais formas em duas categorias mais abrangentes: Aquelas baseadas em *blacklisting* (ou outras heurísticas) e as baseadas em *machine learning*.

Blacklisting consiste em uma técnica bastante comum para resolução deste problema. Remonta ao uso de uma lista de URLs, geralmente compondo um banco de dados, onde consultas são efetuadas a cada nova URL visitada. Aquelas presentes na lista são consideradas maliciosas e, no caso de sua visita, uma providência deverá ser tomada. Alguns dos problemas encontrados neste tipo de abordagem incluem inabilidade de se operar dinamicamente (para mudanças no nome de URLs por exemplo) e a complexidade de se definir uma URL como realmente maliciosa. Para tanto, extensões a esta técnica através de heurísticas são, em alguns casos, implementadas (SAHOO; LIU; HOI, 2017).

Diversas aplicações computacionais só vêm a fazer algum sentido, em um aspecto que pode ser contemplado por um ser humano, quando as mesmas são executadas. Uma das verificações mais formais para a tentativa de se validar algum conteúdo no âmbito da segurança diz respeito ao uso de uma *virtual machine* (VM) para acesso a tal conteúdo. Dadas as capacidades da VM sendo, para este tipo de situação, equivalentes as de um dispositivo comum, há maneiras de se avaliar alterações que indiquem algum comportamento suspeito por parte do conteúdo sendo executado na mesma. Esta metodologia é utilizada por exemplo, no trabalho Moshchuk et al. (2006). Especificamente para aplicação no problema de URLs maliciosas, Wang et al. (2006) propõe e demonstra uma ferramenta que utiliza em parte esta técnica baseada em VMs, para detecção de links que servem para *drive-by downloads*. Outros trabalhos também optaram por seguir tal linha (MA et

al., 2009).

Os chamados *honeyclients* constituem uma outra solução para este problema. *Honeyclients* são clientes *web* que iniciam contato a servidores com suspeita de atividade maliciosa, para tentativa de iniciar um resposta de ataque. *Honeyclients* são uma tecnologia baseada nos chamados *honeypots*, ferramentas que se valem da incitação a ataques maliciosos (literalmente definida como “isca”) para então, examinar o comportamento das entidades envolvidas (IKINCI; HOLZ; FREILING, 2008). *Honeyclients* podem, por exemplo, ser utilizados em combinação com sistemas de automação para busca e classificação de URLs. Exemplos de trabalhos nesta modalidade são o Microsoft HoneyMonkey (WANG et al., 2006) e o Projeto MonkeySpider (IKINCI; HOLZ; FREILING, 2008).

No entanto se tratando de técnicas mais modernas e que implicam em menor consumo de recursos, àquelas baseadas em IA logo ganham seu destaque. Trabalhos como Urcuqui et al. (2017) e Canali et al. (2011), são exemplos claros da viabilidade da aplicação de tais técnicas para a solução deste tipo de problema em específico. Além de tal viabilidade, a eficiência para a execução e a capacidade de generalizar um problema são atrativos que tornam este tipo de técnica ideal para a solução de problemas em que se pretende classificar uma entidade.

É notável ainda que mesmo algumas das técnicas mais elementares de IA conseguem uma solução interessante para o problema. É enfatizado, no entanto, que mesmo a mais simples das técnicas baseadas em IA possui todo um formalismo em sua construção, este formalismo sendo muitas vezes advindo de áreas da matemática e lógica.

2.3 Fundamentos matemáticos

Nesta seção apresenta-se a base matemática sobre a qual este trabalho se fundamenta. Os tópicos seguintes retratam as terminologias e noções matemáticas elementares para a compreensão do conteúdo que se segue, sobretudo, os tópicos da área de IA. Para uma explicação mais aprofundada e específica, é recomendável atentar aos materiais citados em referência.

2.3.1 Vetores e separabilidade

No ramo da álgebra linear, tem-se que um vetor se define como uma entidade matemática que possui representações de módulo, direção e sentido. Formalmente, segundo Santos (2010), um vetor pode ser entendido como uma classe de segmentos de reta que possuem mesmo módulo, direção e sentido. Autores como Carrell (2005) também chegam a definir um vetor como uma matriz coluna simples. No ramo da computação, existem estruturas de dados que também são conhecidas como vetores e comumente este tipo de estrutura é utilizada para representar a entidade matemática vetor. Em vias de evitar

equivocos advindos de tal nomenclatura, este trabalho irá se referir a tais estruturas de dados como arrays unidimensionais, ou simplesmente, arrays.

Um detalhamento maior acerca de tais definições sobre vetores pode ser encontrado no Apêndice B deste trabalho.

Com a apresentação do conceito de vetor, é acessível e interessante introduzir o conceito de separação linear. Supondo a existência de um conjunto hipotético de n pontos referentes a subconjuntos de dados, dispostos em um plano bidimensional, e que tais pontos possam ser visualmente agrupados. Desta maneira, é interessante observar a Figura 8.

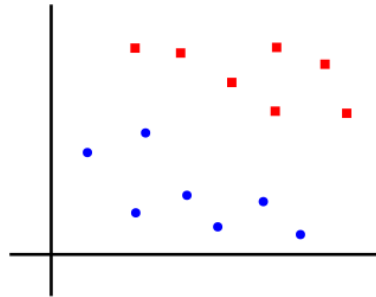


Figura 8 – Conjunto hipotético em um plano 2D.

Conforme é perceptível na representação da figura, para este caso, é suficientemente simples traçar uma linha que separe tais dados, demonstrada na Figura 9.

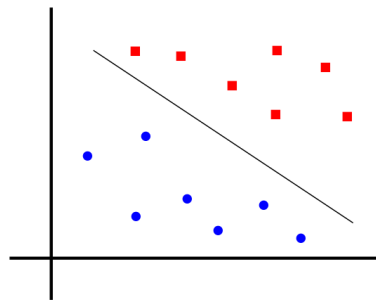


Figura 9 – Conjunto hipotético com uma possível fronteira delimitadora simples em um plano 2D.

Para um outro conjunto hipotético em apenas uma dimensão, se pode encontrar um ponto que separe tais dados. Em outro conjunto em três dimensões, um plano pode ser utilizado para separar tais dados. Na totalidade destes exemplos, tais dados são ditos como linearmente separáveis (KOWALCZYK, 2017).

A visualização e a formulação da separabilidade são bastante simples até o caso tridimensional. Porém, para generalizar casos em dimensões de maior número de coordenadas, é necessária a definição de um outro artifício. Para tais dimensões acima do caso tridimensional, é possível utilizar um hiperplano para a separação. Em geometria, um hiperplano em um espaço n -dimensional é definido como um subespaço $(n - 1)$ -dimensional

incluso em tal ambiente (KOWALCZYK, 2017). Para se evidenciar a noção matemática de hiperplano, uma das maneiras é partir da chamada equação geral da reta.

$$x_2 = ax_1 + b$$

Onde a é definido como coeficiente angular e b como coeficiente linear da reta. Tal equação pode ser reescrita como segue.

$$ax_1 - x_2 + b = 0$$

É interessante notar que ambas as variáveis x_1 e x_2 , definidas na equação, podem ser consideradas como vetores n -dimensionais, podendo desta forma assumir conjuntos de elementos (coordenadas). Desta forma, é bastante direta a extensão das definições envolvendo tal equação, para dimensões distintas quanto ao número de coordenadas. Por conveniência, neste caso, será exemplificada através de vetores bidimensionais. Consideram-se para tanto, os vetores propostos a seguir.

$$V = \begin{cases} \mathbf{x} = (x_1, x_2) \\ \mathbf{w} = (a, -1) \end{cases}$$

A equação pode ser novamente reescrita da seguinte maneira, fazendo uso das definições dos vetores propostos V .

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Esta equação pode ser então definida como a equação geral do hiperplano (LORENA; CARVALHO, 2003).

Existem no entanto casos onde não é possível se separar os dados, em qualquer que seja o espaço nas n -dimensões, apenas através do uso de um hiperplano. Assim sendo, tais casos são denominados como não linearmente separáveis (KOWALCZYK, 2017).

Formalmente, Elizondo (2006) define conjuntos linearmente separáveis da seguinte maneira: Dois conjuntos de dados distintos em algum espaço \mathbb{R}^n são ditos linearmente separáveis se existe algum hiperplano P de \mathbb{R}^n tal que os elementos de um conjunto fiquem do lado oposto aos elementos do outro conjunto. Assim sendo, há a capacidade de se distinguir entre tipos de dados fazendo o uso de tais representações, o que introduz o conceito de classificadores lineares.

2.3.2 Multiplicadores de Lagrange

Alternando o contexto para o ramo da pesquisa operacional e otimização, para compreensão de conteúdos posteriores, é necessária a introdução do conceito de multiplicadores de Lagrange. Para alguns tipos de problema de otimização, é bastante complexo se chegar a uma solução correta. Um artifício desenvolvido pelo matemático italiano Joseph Louis Lagrange é o de utilizar do gradiente da função objetivo para se encontrar pontos de máximo e mínimo. A definição deste artifício se encontra disponível no Apêndice B.

2.3.3 Estatística

Um outro ramo da matemática sobre o qual se evocaram alguns conceitos neste trabalho é o ramo da estatística. Apesar de não ser uma área de estudo recente, a mesma ganhou bastante atenção em épocas mais atuais, com o advento das técnicas e tecnologias em IA. É notável que muitas das ferramentas disponíveis no ramo da IA tem por base conceitos advindos da estatística, e além disso, a mesma também pode servir de apoio a toda uma gama de outras disciplinas.

2.3.3.1 Probabilidade

Um dos conceitos que forma a base de toda a estatística é o de probabilidades. Por probabilidade, há de se entender como a noção de chances de ocorrência de algum evento, dentro de alguma situação observada. Tais eventos são ditos eventos não-determinísticos (ou aleatórios), em contraste com os eventos determinísticos, já explorados por outras disciplinas da matemática. No estudo de probabilidades, se busca definir uma forma de se estabelecer qual a chance de que um evento venha a ocorrer.

Em uma definição formal, conforme explicitado em Carvajal et al. (2015), considerando o espaço amostral finito e uniforme Ω e um evento qualquer A condicionado a este espaço, a probabilidade de A é dada pelo cálculo da razão entre o número de resultados favoráveis a ocorrência de A e o total de resultados possíveis do experimento.

$$P(A) = \frac{R(A)}{R(\Omega)}$$

2.3.3.2 Entropia

No ramo da teoria da informação, a entropia pode ser entendida como uma forma de quantificar informação. Tal conceito, foi introduzido em Shannon (1948), representando não só uma quantificação a um acervo literal de informação, como também, a singularidade (o quão uma informação é inesperada, por assim dizer) carregada por tal informação. Na época, Shannon buscava, dentre outras coisas, uma forma de transmissão de mensagens onde não ocorresse a perda de informação. Formalmente, a entropia (no ramo da teoria da

informação) pode ser definida como a medida da incerteza que se atribui a uma variável aleatória (EDWARDS, 2008).

Considera-se um conjunto de n eventos com probabilidades de ocorrência de um determinado estado (eventos que “alcançam” determinado estado) definidas como p_i e uma variável X que é dita assumir tais eventos. É dito que a entropia de X é a quantidade de informação produzida quando a mesma assume um dos n eventos. A entropia na teoria da informação é calculada através da fórmula seguinte, considerando K uma constante para fins algébricos.

$$H(X) = -K \sum_{i=1}^n p_i \log p_i$$

Um outro conceito relacionado é o da entropia condicional. A mesma pode ser definida como uma quantificação a informação necessária para descrever o estado i assumido por uma variável Y em relação a outro estado j assumido por uma variável X (SHANNON, 1948). Para este caso, afirma-se que Y está condicionado à X . A entropia condicional pode ser calculada como se segue.

$$H_x(X|Y) = - \sum_{i,j} p(i,j) \log \frac{p(i,j)}{p(j)}$$

3 Inteligência Artificial

A IA é um ramo da computação que vêm ganhando bastante popularidade, tanto na mídia em geral quanto na própria área acadêmica e em pesquisas. Como exemplos claros deste efeito, podemos citar as ações e afirmações de líderes de nações acerca do tema, tal qual apresentado em Daws (2019) e Jing (2017). A IA se faz de diversos aparatos advindos da matemática para formulação de técnicas de processamento de informação, como vetores e conceitos de estatística, por exemplo.

3.1 *Machine Learning*

Machine learning (ou aprendizagem de máquina, em uma tradução literal) pode ser definida como o estudo e modelagem computacional de processos de aprendizagem em suas múltiplas formas de manifestação (CARBONELL; MICHALSKI; MITCHELL, 1983). O interesse na aplicação de *machine learning* a um problema se demonstra, principalmente, quando analisamos a definição do mesmo. Dentre algumas razões para a aplicação de *machine learning*, destacam-se algumas daquelas apontadas por Nilsson (1996), como a existência de algumas tarefas que sejam bem definidas através de exemplos, a possibilidade da abstração de relações, hora “invisíveis”, de conjuntos de dados e também, o caso de que há ambientes que mudam com o tempo, exigindo que a solução para o mesmo siga tal padronização. Ainda, em uma definição bastante simples, segundo Mitchell (1997, p. 3, Tradução nossa): “O campo de *machine learning* se preocupa com a questão de como construir programas de computador que melhorem automaticamente com a experiência”.

O conceito de aprendizagem é em si bastante subjetivo. No entanto, é possível defini-lo de uma forma mais sucinta para o contexto deste trabalho, de acordo com o que é exposto por Mitchell (1997, p. 14, Tradução nossa):

Um programa de computador é dito “aprender” de uma experiência E a respeito de uma classe de tarefas T e medida de performance P , se sua performance nas tarefas em T , como mensuradas em P , melhoram com a experiência E .

Em *machine learning*, se resumem em três as maneiras pelas quais um programa (agente) pode vir a aprender. Estas são definidas através do tipo de *feedback* (retorno) dado às soluções propostas pelo programa. Na aprendizagem não supervisionada, o agente aprende padrões através da entrada de dados, mesmo que nenhum *feedback* explícito se apresente. Já na aprendizagem por reforço, o agente conta com uma série de reforços para suas soluções propostas, estes representados por recompensas ou punições (NORVIG, PETER & RUSSEL, STUART J., 2010).

Na aprendizagem supervisionada, o agente trata de observar “pares” de entrada e saída e aprende de forma a efetuar o mapeamento de tais pares (NORVIG, PETER & RUSSEL, STUART J., 2010). Um exemplo desta forma de aprendizagem seria o caso de um agente “atirador”, havendo informações de ângulo e velocidade como entrada e, como saída, uma informação de erro ou acerto de um alvo qualquer. Este trabalho explora, no que tange sua própria construção, o uso de técnicas de aprendizagem supervisionada.

3.2 Problemas de classificação

Problemas de aprendizagem supervisionada em *machine learning* podem ser agrupados em duas classes distintas, os problemas de regressão e os problemas de classificação. Considerando para exemplificação, a tarefa T , representando uma resolução de um problema hipotético Pr que após a computação de alguma técnica de *machine learning* apropriada M , gera uma saída O . Quando esta saída é uma entre um conjunto finito de valores, Pr é dito como um problema de classificação. Quando a aplicação de M gera como saída O um número real, Pr é dito um problema de regressão (NORVIG, PETER & RUSSEL, STUART J., 2010).

De uma maneira mais sucinta, utilizando de exemplos, um problema como o de se definir a qual espécie um animal pertence, dentre um conjunto definido de possibilidades e utilizando informações de tal animal é considerado um problema de classificação. Em contrapartida, o problema de se prever a quantidade de animais presentes em uma determinada área, dadas informações sobre as características de tal área, se define como um problema de regressão. A tarefa de se classificar URLs entre maliciosas ou não é, inerentemente, entendida como um problema de classificação.

Para solucionar um problema de classificação, conhecendo um conjunto de dados acerca do mesmo, parece visível a necessidade da implementação de um procedimento para que tais dados venham a compor uma forma de se solucionar tal problema. Este procedimento vem a ser denominado classificador. Em uma definição formal, um classificador é uma função que mapeia uma instância não classificada para uma classificação conhecida, mediante o uso de estruturas de dados internas (KOHAVI et al., 1995).

3.2.1 Treinamento

Para operar sobre qualquer que seja o problema, no âmbito de *machine learning*, é necessária uma forma de representação dos processos que envolvem uma solução para o mesmo, matematicamente. Isto é denominado um modelo. Para efetivamente construir um modelo, utilizam-se de técnicas baseadas em algoritmos específicos (alguns dos quais elucidados a seguir), bem como, para o caso de aprendizagem supervisionada, um conjunto amostral para treinamento com soluções preconcebidas (NORVIG, PETER & RUSSEL,

STUART J., 2010).

O processo de treinamento (que representa uma forma de aprendizagem da maneira como estabelecida por NORVIG, PETER & RUSSEL, STUART J. (2010)) se define como a busca de padrões dentro do conjunto amostral, possuindo como resultado o modelo para o problema de solução pretendida (MALOOF, 2006). Tem-se como finalidade da aprendizagem supervisionada, de acordo com NORVIG, PETER & RUSSEL, STUART J. (2010, p. 695, Tradução nossa): “Dado um conjunto de treinamento composto de N exemplos de pares de entrada e saída, $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, onde cada um dos y_j foi gerado por uma função desconhecida $y = f(x)$, descobrir uma função h que se aproxime da real função f ”.

3.2.2 Novas instâncias

O processo de classificação trata justamente da aplicação do modelo desenvolvido durante o treinamento (classificador gerado), para fim de classificar o conjunto de dados pretendido. Um classificador pode ser avaliado quanto à sua precisão, sendo que comumente esta avaliação se dá calculando a razão entre classificações corretas e errôneas do mesmo (KOTSIANTIS; ZAHARAKIS; PINTELAS, 2007). É afirmado que um modelo generaliza bem quando a classificação para um número suficiente de novas instâncias, através de tal modelo é correta (NORVIG, PETER & RUSSEL, STUART J., 2010). Comumente, separa-se parte do conjunto de dados do problema em uma parte para treinamento e outra para validação do modelo.

Para uma melhor visualização das etapas de *machine learning* como um todo, bem como, o caráter de dependência entre estas, a Figura 10 descreve, de acordo com definições baseadas em Müller, Guido et al. (2016), um diagrama contendo tais etapas. É ressaltado que as definições sobre as mesmas se encontram nos textos que a antecedem e também em textos posteriores.

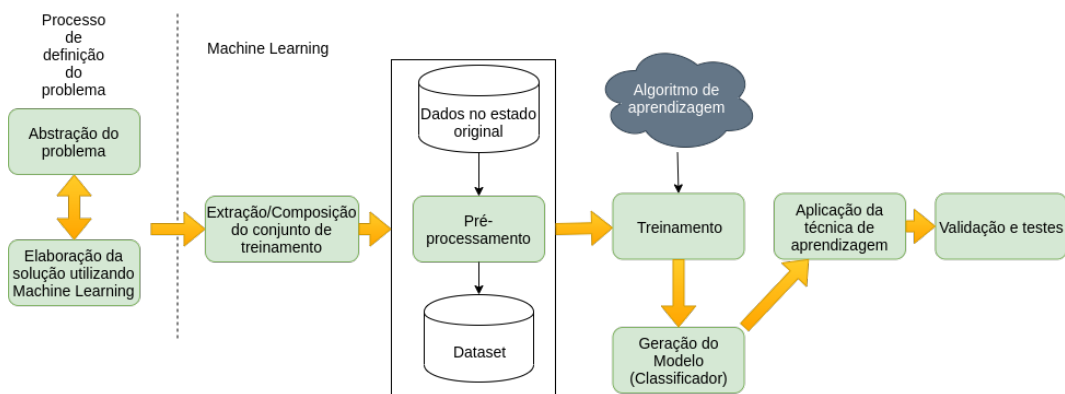


Figura 10 – Descrição das etapas de *machine learning*.

3.3 Técnicas

No desenvolvimento da aplicação, foram trabalhadas uma amplitude de técnicas de *machine learning*. Isto se deve ao caráter exploratório deste trabalho, bem como, às bases que o mesmo toma sobre trabalhos anteriormente efetuados na área.

3.3.1 Classificadores lineares

Muitas das técnicas de *machine learning* se fundamentam na separação de conjuntos de dados por meio de artifícios matemáticos. Classificadores que são capazes de separar os dados por meio de um hiperplano são denominados lineares, e podem ser definidos pela equação que segue, originária da chamada equação geral da reta (hiperplano):

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Nesta equação tem-se que \mathbf{w} é o vetor normal ao hiperplano, \mathbf{x} é um vetor suposto de objetos n-dimensionais e b é um termo compensador. Tal equação divide o espaço de entrada em duas regiões, $\mathbf{w} \cdot \mathbf{x} + b > 0$ e $\mathbf{w} \cdot \mathbf{x} + b < 0$ (LORENA; CARVALHO, 2003).

Um exemplo visual desta construção se encontra na Figura 11.

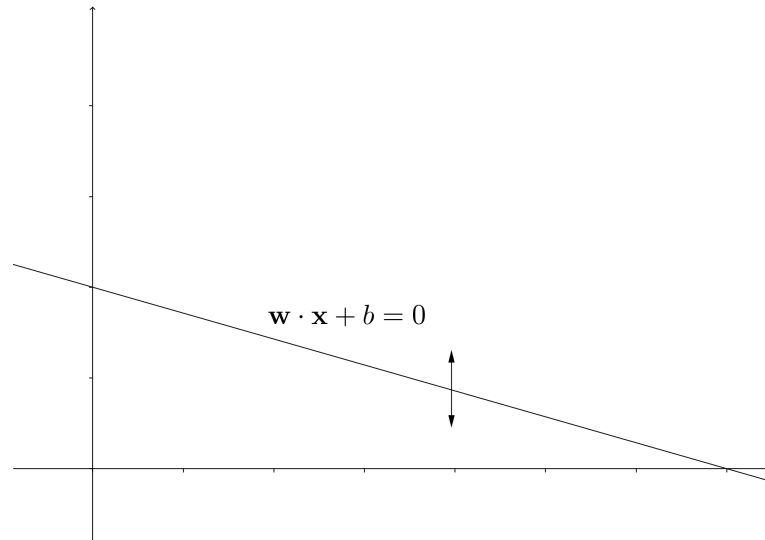


Figura 11 – Representação de um hiperplano separador.

3.3.2 SVM

As Support Vector Machines (SVM), de tradução Máquinas de Vetor Suporte, consistem em uma das técnicas mais populares para aplicação de aprendizagem supervisionada. Formuladas com base nos estudos conduzidos por Vapnik e Chervonenkis e, anos

mais tarde, ganhando um melhoramento através de Cortes e Vapnik (1995), o mesmo além de persistir como técnica, vêm recebendo crescente atenção nos últimos anos¹.

Um problema que advém do uso de classificadores lineares é que, dado um conjunto linearmente separável, podem ser gerados diversos hiperplanos separadores para o mesmo, que, embora corretos na separação dos conjuntos, podem acabar resultando em classificações que não generalizam bem para o conjunto de treinamento em questão. As SVM tratam de resolver esta situação, pelo uso do chamado separador de margens máximas (NORVIG, PETER & RUSSEL, STUART J., 2010).

Esta técnica se fundamenta sobre aquilo que é conhecido como Teoria de Aprendizado Estatístico (CORTES; VAPNIK, 1995). A mesma estabelece condições que auxiliam na escolha de um classificador a partir de um conjunto de exemplos de treinamento (LORENA; CARVALHO, 2003).

Para as SVMs, um vetor representa em si um exemplo. Cada componente do vetor pode ser entendida como uma *feature* (característica), e desta forma, seu posicionamento no espaço definido implica em sua classificação. É notável também que as SVMs operem tanto sobre atributos ditos nominais quanto atributos contínuos.

3.3.2.1 Fundamentos do Perceptron

Antes de adentrar nos conhecimentos acerca da técnica SVM em si, é interessante observar um panorama sobre uma técnica para aprendizagem mais simplificada, o chamado Perceptron. Além de simplificar a compreensão de alguns conceitos acerca das SVMs, ter uma noção sobre o funcionamento do Perceptron auxilia a ilustração das vantagens que as próprias SVMs carregam.

O Perceptron é um modelo probabilístico que foi desenvolvido por volta do ano 1957 por Frank Rosenblatt em seu trabalho no ramo de redes neurais, tendo por inspiração, o modo de operação dos neurônios (ROSENBLATT, 1958). É considerado como o bloco elementar para a construção de alguns tipos de redes neurais mais complexas. O objetivo de um perceptron é encontrar um hiperplano capaz de segregar corretamente um conjunto de dados linearmente separável, podendo assim efetuar a classificação binária de dados (KOWALCZYK, 2017).

Para tanto, o perceptron utiliza de uma função denominada “função hipótese”, que opera em vias de classificar os dados de entrada. Esta, por sua vez, toma por base a equação do hiperplano. Na função hipótese, cada vetor de entrada é associado a uma classificação que pode ter o valor +1 ou -1. Considerando um vetor de parâmetros de entrada \mathbf{x}_i , podemos definir uma função hipótese h como segue.

¹ Vide exemplo a publicação em: <https://www.theregister.co.uk/2019/06/21/alexa_heart_attack/>

$$h(\mathbf{x}_i) = \begin{cases} +1, & \text{caso } \mathbf{w} \cdot \mathbf{x}_i + b \geq 0 \\ -1, & \text{caso } \mathbf{w} \cdot \mathbf{x}_i + b < 0 \end{cases}$$

Tal função utiliza a posição do valor de entrada \mathbf{x}_i em relação ao hiperplano para gerar uma classificação y_i para tal entrada. O resultado pode ser inteligível de maneira a obter algo como a Figura 12.

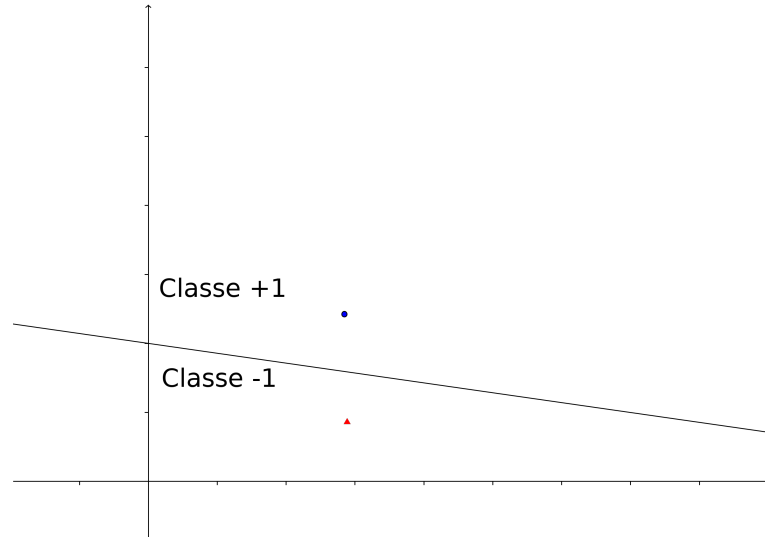


Figura 12 – Separação de conjuntos através de um hiperplano.

Utilizando a definição de uma função sinal, esta formulação será equivalente à seguinte.

$$h(\mathbf{x}_i) = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i + b)$$

A formulação de h pode ser ainda simplificada, conforme demonstrado por (KOWALCZYK, 2017), adicionando um componente $x_0 = 1$ ao vetor $\mathbf{x}_i = (x_1, x_2, \dots, x_n)$ e um componente $w_0 = b$ ao vetor $\mathbf{w} = (w_1, w_2, \dots, w_n)$. Desta forma, haverá o vetor aumentado que segue.

$$h(\mathbf{x}_i) = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i)$$

Constata-se que o formato do hiperplano depende unicamente do vetor \mathbf{w} , visto que os componentes principais da equação do hiperplano agora estão abarcados no vetor \mathbf{w} . Prosseguindo, o que resta ao Perceptron é encontrar um valor para \mathbf{w} (este que até então é desconhecido), de forma a definir o hiperplano separador que classifique os dados de entrada. Isto é efetuado por intermédio de um algoritmo conhecido como Perceptron Learning Algorithm (PLA). O PLA é a forma como o perceptron é treinado, para se referir aos conceitos anteriores. O algoritmo pode ser sumariamente definido da seguinte forma, de acordo com o que é proposto por (KOWALCZYK, 2017).

1. Iniciando com um hiperplano gerado aleatoriamente. Os exemplos de treinamento devem ser classificados;
2. Iterando sobre os exemplos, caso algum destes tenha sido classificado erroneamente, os valores do vetor \mathbf{w} devem ser atualizados (utilizando uma formulação própria para isto, denominada regra de atualização);
3. Ocorre novamente a classificação dos exemplos de treinamento;
4. Os passos 2 e 3 são repetidos até que não haja nenhum exemplo classificado incorretamente ou até que um limite máximo de iterações seja atingido.

Conforme afirmado por Smola et al. (2000, p.3, Tradução nossa): “O algoritmo Perceptron é incremental, no sentido de que pequenas mudanças são feitas no vetor de pesos em resposta a cada um dos exemplos classificados”. Nesta literatura, o vetor de pesos pode ser entendido como análogo ao vetor \mathbf{w} .

3.3.2.2 SVM de margens rígidas

SVMs, de forma similar, buscam definir um hiperplano separador que será utilizado para classificação de novas instâncias de dados. No entanto, diferentemente do Perceptron, o resultado alcançado pela aplicação de uma SVM é dito o hiperplano ótimo, que se apresenta como aquele que melhor segrega um conjunto de dados (LORENA; CARVALHO, 2003). Para simplificar o entendimento e melhorar a visualização, as definições seguintes trabalham com exemplos em planos bidimensionais (Figura 13).

Para se definir um hiperplano ótimo, antes é necessário a elaboração de um critério para que hiperplanos possam ser comparados quanto a separação. Uma ideia trivial, conforme apresentada em Kowalczyk (2017), pode advir da tentativa de compará-los utilizando a própria equação do hiperplano. Considerando um exemplo inicial qualquer (\mathbf{x}, y) e um hiperplano compactuante a tal exemplo. O vetor \mathbf{x} é aquele que contém os dados do exemplo e y é sua classificação. Em um caso de classificação binária simples, $y = \{+1, -1\}$. Considerando a equação geral do hiperplano, ao relacionar o exemplo com tal hiperplano, os resultados possíveis à aplicação de tal equação (como já observados) serão tais que $\mathbf{w} \cdot \mathbf{x} + b = 0$ ou $\mathbf{w} \cdot \mathbf{x} + b = n$, onde n será um número qualquer concebível naquela dimensão.

O resultado n corresponde a uma medida da distância entre o ponto definido pelo exemplo e o hiperplano em si. Ainda, assumindo valores com sinais distintos, n pode também representar qual a posição do exemplo (classificação) em relação a linha, pela própria definição de separação linear. Considerando a totalidade do conjunto de exemplos, é admissível calcular o valor n para cada um dos exemplos, obtendo assim um conjunto $\mathbf{n} = (n_1, n_2, \dots, n_i)$. Assim sendo, é possível obter o menor valor de tal conjunto, que

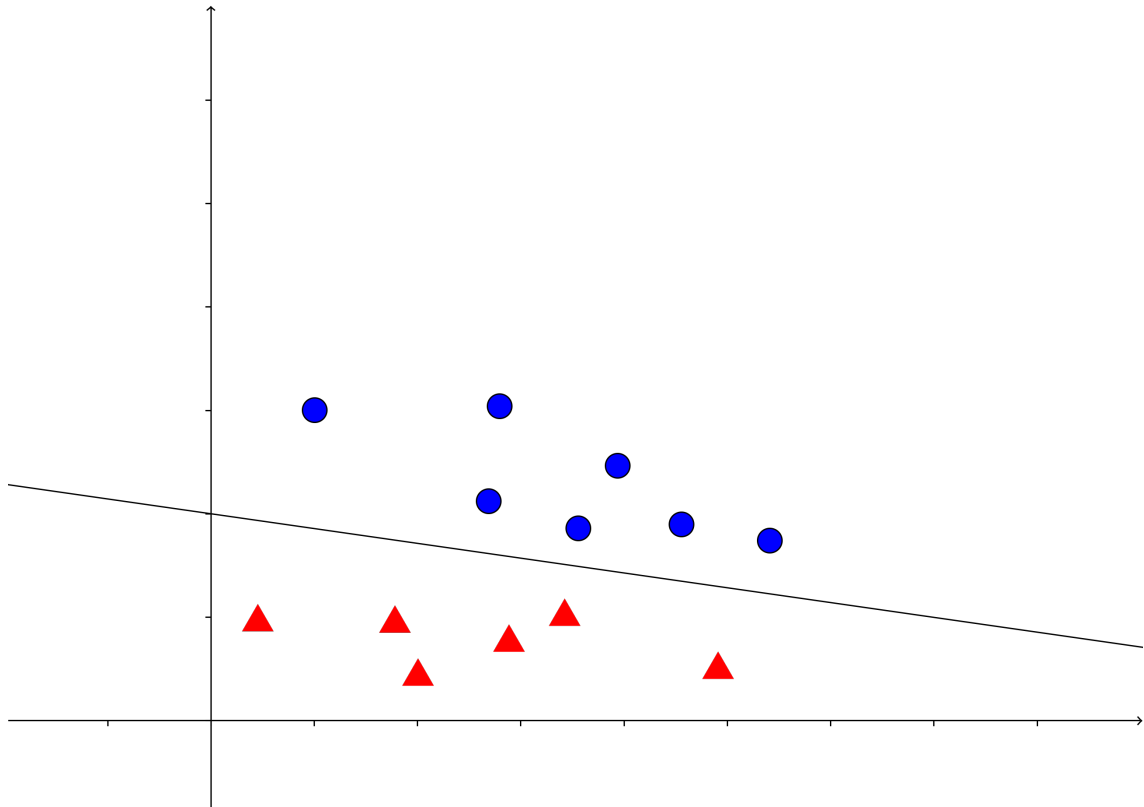


Figura 13 – Hiperplanos separador representado com exemplos de classes distintas com triângulos e círculos.

corresponde a distância de um ou mais exemplos que se encontram mais próximos do hiperplano. Determina-se tal valor como N .

Da noção de separabilidade, tem-se que quanto maior o valor N , mais interessante (no que tange a separação) tal hiperplano se demonstra. Logo, do conjunto de todos os hiperplanos passíveis de divisão do conjunto de exemplos, se deve escolher aquele com maior valor N . De uma outra exposição, a escolha é feita se baseando no maior valor dentre os menores valores resultantes do cálculo da distância entre exemplos e hiperplano. Esta distância é denominada como margem (LORENA; CARVALHO, 2003).

Os vetores que estão a uma distância mais próxima do hiperplano (e que, como afirmado anteriormente, definem a margem) são denominados vetores suporte (GONÇALVES, 2015). Vale notar também que a classificação do hiperplano em si, irá depender unicamente das características de tais vetores suporte. Como o hiperplano é completamente definido através de seus vetores suporte, a solução não deve depender de outros exemplos de entrada (CHEN; LIN; SCHÖLKOPF, 2005). Isto confere as SVMs vantagens no que tange classificações com conjuntos de treinamento pequenos, havendo uma boa demonstração deste efeito no trabalho Chi, Feng e Bruzzone (2008).

No entanto, esta formulação para comparação de hiperplanos apresenta um problema para resultados com sinal negativo. Considerando dois valores N negativos hipotético

u e v , de tal forma que $0 > v > u$. Considerando a formulação atual, o valor u será preferível ao valor v , embora v se encontre mais próximo ao hiperplano. Para superar tal limitação, utiliza-se o artifício da operação de valor absoluto (módulo), a ser aplicado no resultado da equação do hiperplano.

O próximo ajuste a ser efetuado trata-se da classificação em si. A formulação do modelo elaborada até o momento não implica em uma classificação correta. Isto pode ser demonstrado facilmente na Figura 14.

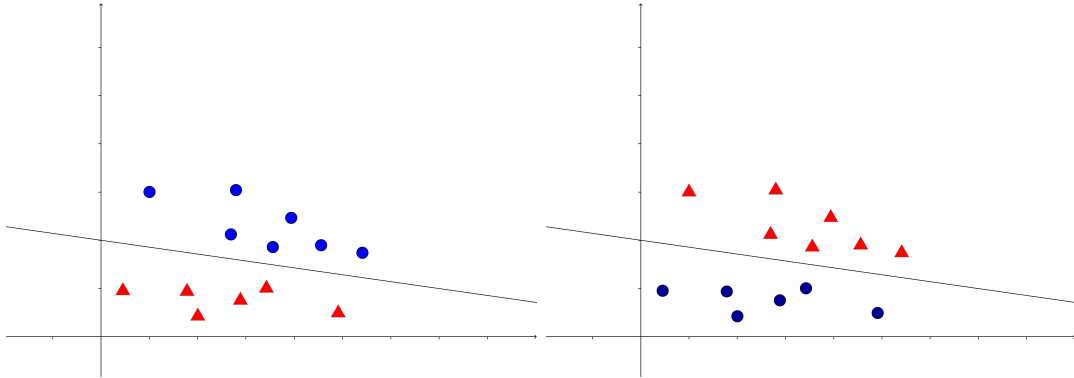


Figura 14 – Hiperplanos separadores equivalentes gerando classificações distintas.

Para a solução deste problema, utiliza-se o elemento y dos exemplos (\mathbf{x}, y) . A caráter de recordação, vale lembrar que os valores de y são tais que $y = \{+1, -1\}$. Portanto, ao multiplicar y pela equação atual, obtêm-se uma nova fórmula, na qual consta que o ponto está incorretamente classificado para resultados negativos (KOWALCZYK, 2017). Por notação, definiremos o resultado de tal fórmula como n_f . Ressalta-se que não há alteração na metodologia de escolha do hiperplano, apenas na construção da fórmula.

$$n_f = y(\mathbf{w} \cdot \mathbf{x} + b)$$

Uma outra limitação que apresenta na formulação, até o momento, é que a mesma ainda varia conforme a proporção do vetor (escala). Isto implica, por exemplo, na classificação errônea de vetores equivalentes. Para transpor tal limitação, ao invés de se manter com o vetor \mathbf{w} atual, na formulação, será empregado o vetor unitário de \mathbf{w} . Assim sendo, os termos \mathbf{w} e b (que também pode ser definido como termo bias) são divididos por $|\mathbf{w}|$. Ao se fazer tal modificação, o resultado da fórmula passa a ser definido como n_y .

$$n_y = y\left(\frac{\mathbf{w}}{|\mathbf{w}|} \cdot \mathbf{x} + \frac{b}{|\mathbf{w}|}\right)$$

Ainda, da geometria analítica, define-se formalmente o vetor \mathbf{w} como vetor normal ao hiperplano descrito. Já o termo $\frac{b}{|\mathbf{w}|}$ é aquele que corresponde a distância do hiperplano à origem (LORENA; CARVALHO, 2003).

A noção de margem pode ser demonstrada visualmente, como se segue. Vale notar que a margem estabelece uma noção de “zona de segurança”, pois por conformidade às definições anteriores, não podem existir ocorrências de exemplos de treinamento que adentrem a mesma (Figura 15).

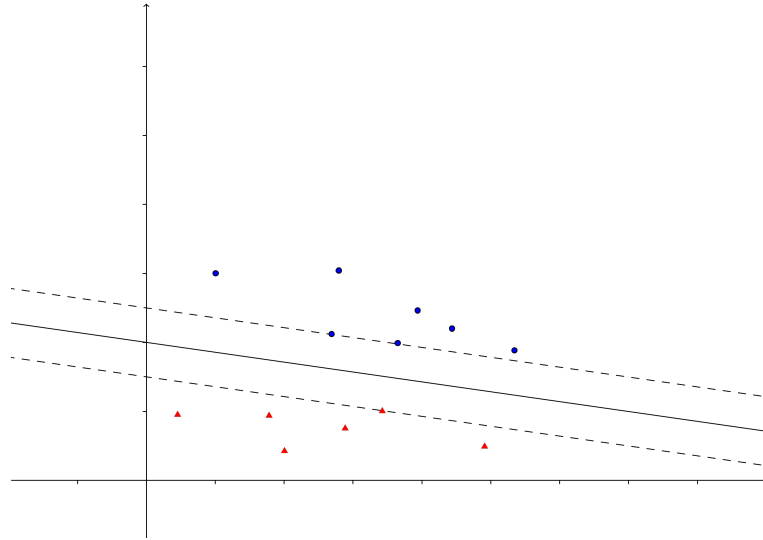


Figura 15 – Hiperplano e margens de separação dos conjuntos.

Com a formulação em seu estado atual, já existe a capacidade do cálculo do valor referente à proximidade dos exemplos para diferentes hiperplanos, resultando em uma forma de se compará-los. Desta maneira, o objetivo seguinte se define como encontrar uma forma eficiente de se efetuar a averiguação entre os elementos do conjunto de hiperplanos possíveis para um problema, a fim de se encontrar o hiperplano com maior margem. Na prática, isto se resume a encontrar os valores de \mathbf{w} e b que produzam a maior margem (KOWALCZYK, 2017). Para tanto, o artifício utilizado aqui é o da elaboração de um problema de otimização.

Para se obter os valores do vetor normal \mathbf{w} e de b , considerando um conjunto linearmente separável de m exemplos para treinamento denominado D , é desejável maximizar o valor do conjunto de menores valores resultantes do cálculo de n_y . Tal conjunto será denominado M . Portanto, o problema trabalhará com a restrição de que a margem de cada exemplo deverá ser maior ou igual a M .

$$\begin{aligned} & \underset{\mathbf{w}, b}{\text{maximizar}} && M \\ & \text{sujeito à} && n_y i \geq M, \quad i = 1, \dots, m \end{aligned}$$

Tal qual demonstrado por Kowalczyk (2017), é possível expressar facilmente a margem geométrica em termos da margem funcional. Isto serve para a simplificação posterior das restrições. Denota-se nesse caso o menor valor do conjunto de valores n_f

como F , e para esta composição, é utilizado da igualdade $M = \frac{F}{|\mathbf{w}|}$, bem como da definição de n_y .

$$\begin{aligned} & \underset{\mathbf{w}, b}{\text{maximizar}} \quad M \\ & \text{sujeito à} \quad \frac{n_f i}{|\mathbf{w}|} \geq \frac{F}{|\mathbf{w}|}, \quad i = 1, \dots, m \end{aligned}$$

A seguir, ainda no viés de simplificar as restrições, é possível remover a razão pela norma na inequação. Ainda, como pela definição, y assume valores tais que $|y| = 1$ pode-se escalar \mathbf{w} e b (o que não altera o resultado) de forma a considerar $F = 1$.

$$\begin{aligned} & \underset{\mathbf{w}, b}{\text{maximizar}} \quad M \\ & \text{sujeito à} \quad n_f i \geq 1, \quad i = 1, \dots, m \end{aligned}$$

Expressando novamente a margem geométrica em termos da margem funcional, desta vez considerando $F = 1$, subsiste a formulação do problema no formato a seguir.

$$\begin{aligned} & \underset{\mathbf{w}, b}{\text{maximizar}} \quad \frac{1}{|\mathbf{w}|} \\ & \text{sujeito à} \quad n_f i \geq 1, \quad i = 1, \dots, m \end{aligned}$$

Para alguns problemas de maximização, muitas vezes é interessante se definir um problema de minimização equivalente, em vias de economizar em recursos computacionais (ARENALES et al., 2015). É notório que a computação de tal problema de minimização equivalente seja bem mais rápida, considerando que a elaboração da equivalência seja um custo ignorável. Este problema pode então ser reescrito como segue (KOWALCZYK, 2017).

$$\begin{aligned} & \underset{\mathbf{w}, b}{\text{minimizar}} \quad |\mathbf{w}| \\ & \text{sujeito à} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, m \end{aligned}$$

Por conveniência, como nas etapas seguintes é utilizada uma solução que envolve a computação de uma derivada, para este problema em específico pode-se adicionar uma multiplicação pelo fator $\frac{1}{2}$. Também é de praxe considerar a norma do vetor da função objetivo como uma potência quadrática. Ambas modificações não alteram o conjunto de soluções ótimas, sendo efetuadas a fim de transformar o problema em um problema de otimização quadrático convexo.

$$\begin{aligned} & \underset{\mathbf{w}, b}{\text{minimizar}} \quad \frac{1}{2} |\mathbf{w}|^2 \\ & \text{sujeito à} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \quad i = 1, \dots, m \end{aligned}$$

Uma outra forma de se alcançar este mesmo formato para o problema de otimização, conforme demonstrado em Lorena e Carvalho (2003), utiliza da noção geométrica da margem (Figura 16). Para tanto, é necessário considerar novamente um hiperplano separador ótimo, definido pela equação geral do hiperplano. Considere também dois outros hiperplanos, H_1 e H_2 definidos como na figura seguinte. É de se notar que não existe nenhum exemplo na região delimitada pelos hiperplanos, haja vista a propriedade de escala dos hiperplanos.

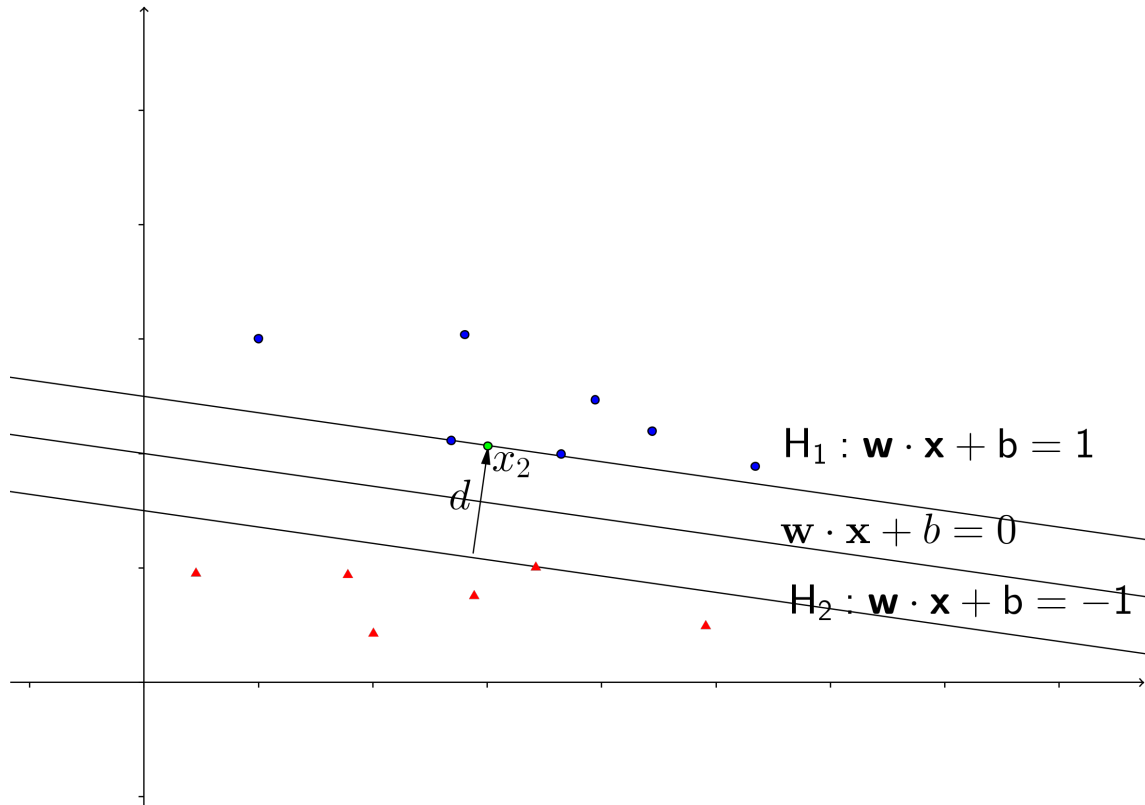


Figura 16 – Hiperplano classificador representado juntamente com H_1 e H_2 .

A tarefa seguinte consiste em encontrar uma relação para expressar a distância entre os hiperplanos (margem). Considerando um ponto qualquer x_2 definido no hiperplano H_2 , para estabelecer a distância de tal ponto à H_1 , pode-se utilizar a propriedade da perpendicularidade. Tal distância será determinada como d . Em benefício a isto, $\frac{w}{|w|}$ é considerado um vetor unitário, normal ao hiperplano H_1 . Dessa maneira, considera-se o ponto $(x_2 + d \cdot \frac{w}{|w|})$ como um ponto definido em H_1 . Logo, a expressão a seguir é válida.

$$w(x_2 + d \cdot \frac{w}{|w|}) + b = +1$$

Ao realizar a expansão desta formulação, tem-se o que segue.

$$\begin{aligned}
& \mathbf{w}\mathbf{x}_2 + d \cdot \frac{\mathbf{w} \cdot \mathbf{w}}{|\mathbf{w}|} + b = +1 \\
\Rightarrow & \mathbf{w}\mathbf{x}_2 + d \cdot |\mathbf{w}| + b = +1 \\
\Rightarrow & \mathbf{w}\mathbf{x}_2 + b = 1 - d \cdot |\mathbf{w}| \\
\Rightarrow & -1 = 1 - d \cdot |\mathbf{w}| \\
\Rightarrow & -1 - 1 = -d \cdot |\mathbf{w}| \\
\Rightarrow & 2 = d \cdot |\mathbf{w}| \\
\Rightarrow & \frac{2}{|\mathbf{w}|} = d
\end{aligned}$$

Como d foi definida como a distância de um ponto definido em H_2 até H_1 , é trivial afirmar que a distância entre estes e o hiperplano ótimo é $\frac{1}{|\mathbf{w}|}$. Para assegurar que não hajam exemplos na região delimitada, pode-se aplicar uma restrição, obtida se baseando em H_1 e H_2 . A formulação a ser alcançada será bem semelhante em estrutura à da margem funcional.

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0$$

Através de tais asserções (noção geométrica), o problema irá ser definido da mesma forma como a descrição anterior (algébrica). Para a solução deste problema, é utilizada a técnica dos multiplicadores de Lagrange (Subseção 2.3.2). A Lagrangiana para o problema pode ser escrita como segue, com seu termo multiplicador a_i .

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}|\mathbf{w}|^2 - \sum_{i=1}^m a_i y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1$$

Ao resolver o sistema de equações inerente ao método dos multiplicadores de lagrange, serão obtidos os seguintes valores.

$$\begin{cases} \sum_{i=1}^m a_i y_i = 0 \\ \mathbf{w} = \sum_{i=1}^m a_i y_i \mathbf{x}_i = 0 \end{cases}$$

Ao substituir tais valores na Lagrangiana, uma nova formulação ao problema de otimização é alcançável. Vale ressaltar que a substituição não é imediata, e sua descrição se encontra disponível em Gonçalves (2015). A esta formulação é denominada problema dual de Wolfe, em contraste com a forma primal (anterior). O interesse em se definir a mesma diz respeito à simplificação do problema para casos com vários exemplos, através da possibilidade de definir uma função objetivo que depende apenas dos multiplicadores de lagrange.

$$\begin{aligned}
& \underset{\alpha}{\text{maximizar}} && \sum_{i=1}^m a_i - \frac{1}{2} \sum_{i,j=1}^m a_i a_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\
& \text{sujeito à} && \begin{cases} a_i \geq 0, \forall i = 1, \dots, m \\ \sum_{i=1}^m a_i y_i = 0 \end{cases}
\end{aligned}$$

O problema ainda reside na busca pelos valores ótimos de \mathbf{w} e b que compõe a margem. Um novo artifício a ser utilizado nesta etapa são as condições de Karunsh-Kuhn-Tucker (KKT), que consistem em um conjunto de condições (limites) para a verificação da otimalidade de uma solução. Tais limites são necessários, devido ao fato da formulação operar sobre restrições baseadas em inequações.

Considerando o valor de \mathbf{w} obtido na solução da forma primal, é possível encontrar o valor ótimo de b tomando a solução da forma dual, juntamente com as condições Karunsh-Kuhn-Tucker.

$$b = y_i - (\mathbf{w} \cdot \mathbf{x}_i)$$

Com a solução do problema de otimização apresentada, é possível a elaboração da função hipótese. Para tanto, tal qual no Perceptron, é utilizada a definição de uma função sinal. Neste caso, considerando b e \mathbf{w} da solução.

$$h(\mathbf{x}_i) = \mathbf{sign}(\mathbf{w} \cdot \mathbf{x}_i + b)$$

Para a formulação dual, no entanto, esta função hipótese é representada da seguinte maneira, considerando uma quantidade S de vetores suporte.

$$h(\mathbf{x}_i) = \mathbf{sign}\left(\left(\sum_{i=1}^S a_i y_i \mathbf{x}_i\right) \cdot \mathbf{x} + b\right)$$

As SVMs são uma técnica verdadeiramente admiráveis, mesmo considerando sua forma mais elementar. A teoria acerca das mesmas pode ainda ser expandida em vias de incorporar a capacidade de trabalhar com casos onde os dados de entrada não sejam linearmente separáveis. No entanto, considerando que este trabalho utilizou a técnica SVM em sua forma mais simples, tais extensões à teoria não serão abordadas. A aplicação de uma SVM se dá de forma similar a do Perceptron. Os novos exemplos serão mapeados fazendo uso da função obtida como resultado.

Considerando o fato de que as SVMs são determinadas exclusivamente por seus vetores suporte, é justo afirmar que os mesmos possam ser considerados a parte mais importante do conjunto de treinamento. Porém, há de se ressaltar que dadas as características próprias da técnica, a mesma é dita ser mais efetiva para resultados com poucos

vetores suporte, onde possui a capacidade de generalizar melhor. Pode ser afirmado, intuitivamente, que para conjuntos de treinamento com poucos exemplos a aplicação do SVM tende a ter melhores resultados (BURGES, 1998).

3.3.3 Random Forest

Dentre as diversas técnicas de *machine learning*, existe uma classe específica destas que trata de combinar múltiplos modelos de aprendizagem em vias de se construir modelos mais poderosos. A estes métodos, se dá o nome de técnicas *ensemble learning* (MÜLLER; GUIDO et al., 2016). Dentre estes, um dos mais conhecidos é o chamado Random Forest (nome que numa tradução livre, significa floresta aleatória). Os primórdios da elaboração desta técnica referem-se ao trabalho de Breiman (1999), sendo aperfeiçoada por outros trabalhos posteriores.

A técnica Random Forest utiliza de árvores de decisão para seu próprio arranjo. Desta maneira, é interessante que este trabalho defina conceitos acerca de árvores de decisão antes de prosseguir.

3.3.3.1 Árvores de decisão

Uma árvore de decisão consiste em uma estrutura de classificação de exemplos, baseada em decisões referentes a parâmetros de tais exemplos. Formalmente, uma árvore de decisão vêm a representar uma função que recebe como entrada um conjunto de atributos e retorna um valor único, após executar uma série de testes. O uso de árvores de decisão é considerado uma técnica de *machine learning* simples, porém eficaz para diversos casos (NORVIG, PETER & RUSSEL, STUART J., 2010). Tais técnicas baseadas em árvores de decisão foram bastante exploradas desde os primórdios dos desenvolvimentos em *machine learning*, em especial, no trabalho de Quinlan (1986).

Árvores de decisão podem ser utilizadas tanto para problemas de regressão quanto para classificação, este último, correspondendo às demandas deste trabalho. Uma característica interessante das árvores de decisão é que a lógica gerada durante o processo de aprendizado pode ser facilmente visualizada, visto que a mesma se organiza tal qual uma estrutura em árvore. Algumas implementações permitem literalmente a exibição de uma árvore contendo o conhecimento gerado.

Esta técnica utiliza a ideia do percurso através dos nós da árvore como a metodologia de classificação. Uma árvore de decisão, estruturalmente, será composta por um conjunto de nós conexos, definidos em duas categorias. Os chamados nós de decisão serão aqueles sobre os quais recai alguma comparação binária, a ser efetuada sobre o exemplo de entrada. Estes nós representam uma decisão acerca de algum atributo do conjunto de exemplos. Já os nós folha, por sua vez, tratam de conter o objetivo à ser alcançado. Considerando

o uso de árvores em problemas de classificação, as folhas podem ser entendidas como as diferentes possibilidades de rotulação para o exemplo (Figura 17). Excepcionalmente, o primeiro nó (conhecido como raiz) será sempre um nó de decisão.

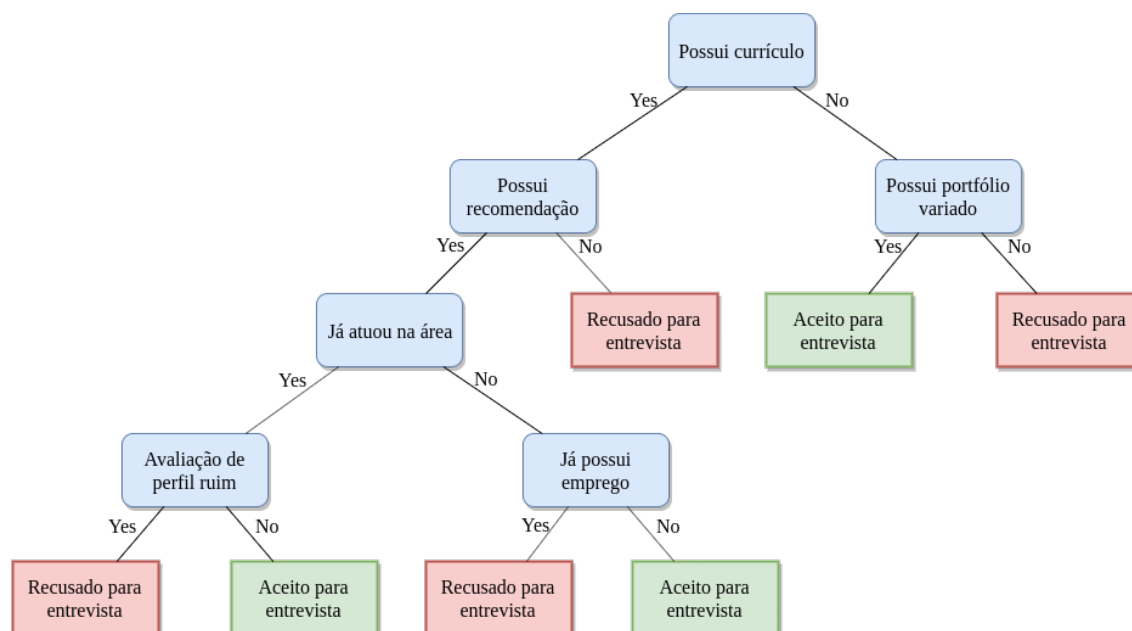


Figura 17 – Árvore de decisão simples para um problema hipotético de classificação de participação em uma entrevista de emprego.

Para que a árvore de decisão possa ser utilizada em um problema de classificação, a mesma precisa antes ser gerada ou, na terminologia comum, induzida. A indução de árvores de decisão é uma tarefa solucionável através de diversos algoritmos distintos. Será definida a seguir, para o que tange este trabalho, a indução de árvores de decisão através do algoritmo CART.

3.3.3.2 CART

O algoritmo denominado Classification and Regression Trees, (CART) foi proposto inicialmente no trabalho Breiman et al. (1984). O CART é definido como construtor *top-down* (que inicia o procedimento partindo do nó raiz) de uma árvore máxima, ou seja, uma árvore cujo crescimento se desenvolve até o alcance do critério de parada. Uma das vantagens a respeito do CART é que o mesmo opera corretamente com entradas sendo elas numéricas ou categóricas (TIMOFEEV, 2004).

A ideia por trás do CART é fazer uso de um processo iterativo para construção da árvore, onde as regras são construídas tomando por base o conjunto de exemplos de treinamento. Para a atribuição de tais regras, é necessária uma forma de mensurar o quão bem uma decisão sobre um atributo separa um conjunto de dados. Dentro da terminologia da técnica, esta forma é denominada função de impureza. Desta maneira, o algoritmo pode

ser definido da seguinte forma, tomando por base a definição de Monard e Baranauskas (2003), o uso da função de impureza e considerando um critério de parada genérico.

1. Um novo nó será criado. Caso a árvore esteja vazia, o mesmo assume a raiz.
2. A seguir, deve-se calcular o grau de impureza de cada um dos atributos do conjunto exemplo.
3. Após o cálculo é possível definir o atributo com o melhor grau de impureza. Caso a separação dos exemplos através de tal atributo resulte em melhoria (melhor separação dos conjuntos, definida através da impureza), a mesma será definida como a regra para tal nó. Caso contrário, o nó será um nó folha.
4. O conjunto de exemplos é particionado de forma a considerar a separação efetuada.
5. Repita o procedimento para cada uma das ramificações (quando houver), até atingir o critério de parada definido.

As árvores resultantes da aplicação do CART são compostas por regras que se expressam na forma *IF C THEN X ELSE Y*, direcionando a uma ou outra das ramificações possíveis. *C* é uma condição inerentemente relacionada ao atributo do nó referido. Para atributos contínuos, esta condição se encontra na forma $a_i \leq b$, com a_i sendo um representante à elementos do conjunto de treinamento e b um valor constante. Para atributos categóricos, a condição se baseia na operação de composição (STEINBERG; COLLA, 2009).

Já a escolha dos atributos é algo que inerentemente requer uma forma de se comparar os mesmos, a fim de que se assuma um atributo melhor para determinado nó. Uma forma de se fazer isso é através do grau de impureza de um atributo. Por grau de impureza, pode se entender como uma quantificação para a possibilidade de erros na classificação de novos exemplos, considerando o uso de uma regra de decisão. Desta maneira, quanto menor o grau de impureza, melhor tal regra de decisão irá operar sobre o problema em si. O CART, como proposto por Breiman et al. (1984), faz uso do chamado índice de Gini, que atribui uma medida ao grau de impureza que determinado atributo assume.

Para se obter o valor do índice, há de se considerar as probabilidades de um elemento assumir qualquer uma das classes possíveis, iniciando de um nó contendo uma regra. Desta maneira, há de se considerar a quantidade K de classes possíveis bem como a probabilidade p_i associada a cada uma (BRAMER, 2007). Como a implementação comum do CART assume a construção de regras de natureza binária, a formulação pode ser simplificada ainda mais. O índice de impureza de Gini pode então ser calculado através

da fórmula representada a seguir, considerando a quantidade C de classes. O termo $p(i)$ representa a probabilidade de escolha de um elemento da classe i .

$$G = \sum_{i=1}^C p(i) \cdot (1 - p(i))$$

A simplicidade dos modelos de árvore de decisão, como os gerados através do CART, é na maioria das vezes vista como sua maior vantagem. Porém, é fato que tais modelos são extremamente sensíveis às mudanças no conjunto de treinamento, sendo que uma pequena mudança em tal conjunto pode resultar na construção de árvores completamente distintas (FRIEDMAN; HASTIE; TIBSHIRANI, 2001).

Um outro problema que comumente ocorre neste tipo de formulação é que a mesma se encontra altamente sujeita à *overfitting*. Formalmente, *overfitting* se trata do caso onde um classificador opera com precisão sobre o conjunto de treinamento, porém, falha na generalização para novos exemplos (NORVIG, PETER & RUSSEL, STUART J., 2010). Existe uma técnica que consiste na eliminação de nós considerados indiferentes à solução, referida como “poda”, que visa tratar os casos de *overfitting* para árvores de decisão.

No entanto, uma solução mais definitiva para tais problemas consiste no uso de técnicas de *ensemble learning* ou mais precisamente, do método de classificação Random Forest.

3.3.3.3 Operações com múltiplas árvores

Uma possibilidade interessante que se apresenta através das árvores de decisão é a do uso de múltiplas árvores compondo uma única solução. Considerando um conjunto de aprendizado L composto por um conjunto de exemplos na forma (\mathbf{x}, y) e uma formulação p , que utiliza este conjunto de aprendizado para classificação, tal que $p(\mathbf{x}, L)$ prediz y . Agora, supondo uma sequência de k conjuntos de aprendizado L_k , cada qual composto de N exemplos advindos de L . Considerando o objetivo de se obter um classificador utilizando de L_k , caso y seja categórico, pode se fazer $p(L, \mathbf{x})$ compor uma predição, agregando todos os resultados para todos os L_k através de representações unitárias para cada um, e então formar $p_l(\mathbf{x})$. Este processo é nomeado como “bootstrap aggregating”, ou Bagging (BREIMAN, 1996a).

Bagging é uma técnica que pode ser utilizada com diversas formulações baseadas em árvore, para melhorar os processos tanto de regressão quanto de classificação. Bagging é melhor aproveitado quando o procedimento sobre o qual está sendo aplicado não é muito estável, ou seja, quando pequenas mudanças no conjunto de treinamento implicam em diferenças consideráveis nos resultados (SUTTON, 2005). No entanto, algumas melhorias podem ser aplicadas, em vias de sugerir mais aleatorização entre as árvores. Para tanto, pode-se fazer uso da técnica Random Forest.

Por definição, uma Random Forest é um classificador que é composto por uma coleção de classificadores árvore, definidos como $h(\mathbf{x}, \theta)$. O termo \mathbf{x} é definido como um exemplo de entrada, enquanto que θ pode ser entendido como o conjunto de vetores independentes $(\theta_1, \theta_2, \dots, \theta_k)$, cada qual representando uma das árvores geradas para classificação. Neste classificador, cada uma das árvores contribui com um único “voto” para definir a classificação mais popular considerando uma entrada \mathbf{x} (BREIMAN, 2001).

De uma forma mais sucinta, a técnica pode ser descrita da seguinte maneira, em etapas, similarmente à forma como elucidado por (BREIMAN, 2004). Primeiramente, múltiplas árvores de decisão são geradas, utilizando cada uma um conjunto de dados de autoinicialização (*bootstrap dataset*), considerado um subconjunto baseado no conjunto de exemplos original. A seguir, para o processo de classificação em si, o vetor a ser classificado é utilizado para um percurso em cada uma das árvores geradas, computando assim uma classificação por árvore. Quando esta classificação é gerada, na terminologia de árvores de decisão, é dito que a árvore votou por aquela classificação. Ocorre então a computação de todos os votos para o exemplo, e a classificação é determinada baseada na maioria.

A indução de árvores difere bem pouco da noção apresentada anteriormente. O que acontece para o Random Forest é que o conjunto de atributos considerados para compor os critérios de decisão é, na realidade, um subconjunto de todos os atributos que compõe os exemplos de treinamento. O processo é repetido para cada árvore gerada, considerando o conjunto de dados de autoinicialização e o subconjunto de atributos (o uso de tal subconjunto sendo uma “melhoria” à ideia de Bagging). Estes mecanismos servem para garantir que as árvores geradas no Random Forest sejam distintas entre si (MÜLLER; GUIDO et al., 2016). Não ocorre poda nas árvores geradas.

Para determinar uma estimativa ao erro, a técnica utilizada é conhecida como “*out of bag dataset*”. Em uma estimativa, cerca de $\frac{1}{3}$ de exemplos do conjunto de dados não é utilizado durante o treinamento, por conta dos fatores de aleatoriedade. Tais exemplos são então utilizados como entradas no conjunto de árvores gerado e é estimada uma taxa de erro. Em Breiman (1996b), se demonstrou empiricamente que tal taxa é precisa o suficiente para uso em técnicas baseadas em Bagging.

Em sua essência, o Random Forest agrega todos os benefícios das árvores de decisão, enquanto compensa por diversas de suas fraquezas (MÜLLER; GUIDO et al., 2016). Uma desvantagem, no entanto, diz respeito ao mesmo não ser tão descritivo (também denominado como *black box*) como uma árvore de decisão simples.

3.3.4 C4.5 e J48

Uma outra técnica de *machine learning* que se fundamenta sobre árvores de decisão foi proposta por Quinlan (1986), sendo conhecida como Iterative Dichotomiser 3, ou ID3.

O mesmo toma por base o conceito de entropia, introduzido por Shannon (1948). Em trabalhos mais recentes, no entanto, a técnica ID3 foi amplamente reformulada e estendida. A esta técnica nova, nomeia-se C4.5 (QUINLAN, 1993).

O C4.5, tal qual o CART, é um algoritmo que permite a indução de árvores de decisão através de um procedimento iterativo, em uma abordagem “dividir para conquistar”. O algoritmo em si pode ser descrito por etapas da maneira como segue, similarmente à forma como apresentado em Wu et al. (2008). Para tanto, considera-se um conjunto de exemplos de treinamento S .

1. Caso todos os exemplos em S pertençam a uma única classe ou o número de exemplos em S seja suficientemente pequeno, a árvore resultante será um nó folha, com classificação referente à classe de maior frequência em S .
2. Caso contrário, define-se um teste baseado em um único atributo, com dois ou mais possíveis resultados. O critério para escolha do teste é o ganho de informação proporcionado pelo mesmo, no qual se opta pelo teste com menor taxa de entropia. Também a chamada taxa de ganho (razão entre o ganho de informação e a quantidade de resultados possíveis) pode vir a ser utilizada como critério.
3. O teste será definido como raiz da árvore e uma ramificação será considerada para cada resultado possível do teste.
4. O conjunto S é particionado em diversos subconjuntos (S_1, S_2, \dots, S_n) , considerando o resultado para cada caso.
5. O procedimento é aplicado recursivamente para (S_1, S_2, \dots, S_n) .

O critério do ganho de informação é utilizado de forma a selecionar o atributo com o maior valor, o que implica numa menor taxa de entropia. Tal critério serve para mensurar o quanto de informação a consideração de um determinado atributo traz para a árvore em si. Considerando novamente o conjunto S , juntamente com as definições acerca de entropia e entropia condicional, pode-se formular o cálculo para o ganho de informação. Para tanto, efetua-se a subtração entre o valor da entropia $E(S)$ e a entropia condicional $E_c(A)$. O ganho de informação IG sobre um atributo A pode ser definido através da fórmula a seguir, conforme Quinlan (1986).

$$IG(A) = E(S) - E_c(A)$$

No entanto, podem existir casos onde existem muitas possibilidades de resultado para um único atributo, o que pode implicar em uma menor taxa de entropia e dessa forma, prejudicar o processo de construção (HARRIS, 2002). Para transpor esta limitação,

um novo critério é utilizado, a chamada taxa de ganho. Diferentemente do critério do ganho de informação, a taxa de ganho leva em consideração, a quantidade de divisões (ramificações em um nó). É notável que embora distintos, a taxa de ganho em si faz uso do ganho de informação (HSSINA et al., 2014).

Apesar de similaridades com o CART, a técnica C4.5 difere em vários aspectos do mesmo. Além da diferença no critério utilizado para escolha de atributos que irão compor os nós de decisão, o C4.5 permite a construção de árvores cujas decisões assumam graus acima do binário (WU et al., 2008). A noção de poda também existe no C4.5, sendo efetuada das folhas para a raiz, considerando comparações de estimativas de erro. A respeito de implementações, uma versão mais recente do C4.5, denominada J48, se encontra disponível na plataforma WEKA (Seção 3.4). O J48 é implementado na linguagem Java, considerando a oitava versão do C4.5. Ambos são equivalentes.

A principal desvantagem do C4.5 diz respeito ao desempenho, prejudicado tanto no quesito complexidade temporal quanto no quesito complexidade espacial. Outra desvantagem de tal técnica é que muitas vezes, as árvores geradas podem ser de difícil interpretação (WU et al., 2008).

3.3.5 Outros

Outras técnicas baseadas em IA também valem ser citadas muito brevemente aqui, considerando sua popularidade e o fato de serem também notórias as aplicações de tais técnicas na solução do problema de classificação de URLs. A começar pela Regressão logística, que consiste em uma técnica baseada no uso de modelos estatísticos para a construção de um classificador binário. Tal técnica faz uso da chamada função logística, advinda do ramo da estatística (NORVIG, PETER & RUSSEL, STUART J., 2010).

Uma outra técnica de uso notório na solução do problema em questão trata-se do Multilayer Perceptron (MLP). Os MLPs, também conhecidos como redes neurais, são um modelo baseado no uso de múltiplos perceptrons, conectados de forma a propagar saídas entre as diversas camadas que compõe a estrutura. São inspiradas no modo de operação do cérebro humano (NORVIG, PETER & RUSSEL, STUART J., 2010).

3.4 WEKA

O projeto Waikato Environment for Knowledge Analysis, de sigla WEKA trata-se de uma coleção de algoritmos de machine learning e ferramentas para processamento de dados, que surgiu em 1992 na Universidade de Waikato, Nova Zelândia (HALL et al., 2009). O mesmo é desenvolvido na linguagem Java, porém possuindo suporte integral a diversas outras linguagens através de uma API própria. O WEKA simplifica diversas

implementações que envolvem *machine learning*, visto que possui dentro de sua coleção, os algoritmos mais comuns de uso do ramo.

Para a implementação, foi decidido em favor do uso da plataforma WEKA. Considerando as necessidades e especificidades do trabalho, tal escolha se mostrou adequada. Ainda, como este trabalho foi efetuado em um período de tempo limitado e não houve financiamento, o tempo consumido com o mesmo pôde ser melhor aproveitado em razão do uso do WEKA.

4 Trabalhos relacionados

Neste capítulo tem-se a apresentação de alguns outros trabalhos na mesma temática ou que abordaram conceitos relacionados à análise de URLs maliciosas como um todo. No capítulo de resultados (Capítulo 6), há a comparação detalhada das repercussões de tais trabalhos com aquilo que fora aqui desenvolvido.

A começar, temos o trabalho de Canali et al. (2011) intitulado “Prophiler”, um projeto financiado pelo programa FP7 e fomentado pela União Européia. O objetivo do Prophiler é o de classificar páginas web, coletadas através de um webcrawler, utilizando-se de técnicas de aprendizado supervisionado. O mesmo foi empregado como uma tentativa de se reduzir a carga de links que seriam analisados por um outro sistema, e a partir deste sistema era efetuada uma análise completa da URL, agindo como uma espécie de filtro. Este outro sistema para o qual o Prophiler filtra é conhecido como “Wepawet”, e utiliza-se de uma análise profunda através de honeyclients.

Dentre as diversas características extraídas pelo Prophiler, para a URL em si, pode-se destacar: O número de padrões considerados “suspeitos” na URL, a presença de um subdomínio, a presença de um endereço IP na URL, data de registro do domínio, data de atualização, código do país de registro, o tamanho do nome do arquivo na URL, TLD da URL, informações acerca do DNS, região, zona temporal, velocidade de conexão do *host* e ainda tamanhos absoluto e relativo da URL.

Em Ma et al. (2009), a proposta é utilização de técnicas de aprendizagem estatística para a composição de um sistema de classificação de URLs. Dentre estas técnicas, destacamos especialmente o uso das SVMs. Esta abordagem serviu como base para o desenvolvimento de diversos outros projetos desta temática, incluindo o já citado Prophiler. Sobre isto, das características de uso para classificação no trabalho de Ma et al. (2009), observa-se que várias das mesmas foram utilizadas no trabalho “Prophiler”.

Já em Urcuqui et al. (2017), foi desenvolvida uma outra aplicação para análise de URLs maliciosas, pelo uso de técnicas de *machine learning*. Também foi explorada novamente o uso das SVMs, e ainda, o uso do J48 (Seção 3.3.4) e outras técnicas. Como características inerentes à URL, de forma notável, este trabalho explora o número de caracteres considerados “especiais” presentes na mesma e também informações acerca da aplicação servidor respondendo sobre tal URL.

Em Choi, Zhu e Lee (2011) utilizou-se de uma elaboração em um nível razoavelmente além dos trabalhos anteriormente citados. Neste trabalho, houve não só a busca por uma categorização de URLs maliciosas, como também, um enquadramento das mesmas nas categorias “spam”, “phishing” e “veiculação de malware”, conforme o tipo da ameaça que apresentavam.

Através do trabalho BINSPECT, de Eshete, Villaflorita e Weldemariam (2012), é apresentada uma alternativa que se define como holística e de baixo consumo de recursos. O design do BINSPECT possui uma subdivisão em três componentes majoritários: Um extrator de características e classificadores, seu componente de treinamento de múltiplos modelos e também a parte de classificação.

Vários dos trabalhos anteriormente citados culminaram na análise de características do *website* que é representado pela URL como um todo. Dentre estas, é possível destacar a análise de código JavaScript incluso na página e a análise do próprio HTML em busca de características que denotassem fraude. Na implementação inerente à este trabalho porém, foram utilizadas apenas as informações acerca da própria URL (sendo elas de origem sintática ou semântica), DNS, GEOIP e WHOIS.

4.1 Características das URLs

Nos trabalhos citados, a parte majoritária das características analisadas diz respeito a aspectos léxicos e sintáticos da URL. Vale citar ainda que, conforme apontado por Alghamdi, Watson e Xu (2016), a indisponibilidade de alguns serviços provedores de informações sobre domínios (como o WHOIS por exemplo), pode prejudicar a análise de URLs. Outro caso que pode ocorrer é que o *website* em si venha a ficar indisponível, o que também impede a coleta de diversos tipos de informações. O trabalho de Alghamdi, Watson e Xu (2016) ainda aponta um outro problema persistente acerca da análise de URLs, quanto a ofuscação.

Uma técnica que pode ser utilizada por atacantes para ofuscação das características da URL é o uso dos chamados “encurtadores de URLs”. Tais serviços proporcionam o redirecionamento de páginas, gerando URLs mais compactas que servem para acessar o mesmo conteúdo da original.

4.2 Outras ferramentas de análise de URLs

Existe uma gama de ferramentas de análise de URL disponíveis para uso atualmente, algumas inclusive, resultados de trabalhos como àqueles referidos anteriormente. Canali et al. (2011), por exemplo, foi proposto para o sistema Wepawet, que futuramente, culminou no projeto Llama. O Wepawet não é mais mantido como um projeto, embora tenha contribuído em grande escala para desenvolvimentos na área. O Google SafeBrowsing, iniciativa que mantém um sistema para detecção e bloqueio de *websites* maliciosos também é uma destas ferramentas. O mesmo existe desde o ano de 2006, possibilitando proteção contra ataques baseados em malware e phishing, porém, com seu código fonte e detalhes de implementação indisponíveis, por questões comerciais.

Uma outra ferramenta é conhecida como VirusTotal¹, que apesar de ter seu foco principal em análise e categorização de malwares, mais recentemente também passou a incorporar capacidades de análise de URLs. A mesma opera atualmente como uma subsidiária à Google.

Há outras ferramentas de software, como por exemplo o Zulu Zscaler e o Sucuri, cada qual fornecendo sua forma de análise, com suas próprias técnicas e demais especificidades. Vale aqui citar também os diversos repositórios de projetos, em plataformas como o Github.com, que servem ao mesmo propósito.

¹ Ferramenta acessível em: <<https://www.virustotal.com/gui/home/upload>>

5 Desenvolvimento

Neste capítulo são descritos os detalhes de implementação da solução de software, bem como noções operacionais sobre o mesmo. O sistema para classificação desenvolvido faz uso de diversas características, algumas das quais notoriamente exploradas pelos trabalhos anteriores.

A linguagem de programação escolhida para utilização foi o Java, por sua versatilidade, robustez e também pelo suporte oferecido pela mesma à plataforma WEKA. As capacidades multiplataforma da linguagem também foram um fator considerado. Foram utilizados alguns subprogramas mais específicos na linguagem Python, em complemento ao sistema principal.

Por boa prática, foi adotada uma nomenclatura em inglês para as classes, métodos e demais elementos do programa passíveis de nomeação. Também foi pretendido o uso das convenções de nome do Java para elementos em código.

5.1 Dependências

Para simplificação e adesão às convenções, foram utilizadas no curso do desenvolvimento da ferramenta algumas bibliotecas, artefatos e ferramentas apropriadas, generalizadas aqui como dependências. Além de promover o reaproveitamento de ferramentas, visto que tais dependências se dispõem a resolver problemas os quais admitem uma implementação geral já existente, estas também vêm a auxiliar um desenvolvimento mais ágil.

5.1.1 OpenCSV

Para algumas funcionalidades do software desenvolvido foram utilizadas técnicas de persistência de dados em arquivo. A OpenCSV é uma biblioteca para análise de arquivos no formato Comma Separated Values (CSV). A mesma conta com funções específicas para criação e tratamento deste tipo de arquivo de forma simples, além de possibilitar ser de fácil integração com a maioria dos programas.

5.1.2 GeoIP2 (Maxmind)

Considerando que o GEOIP é uma informação que é fornecida por serviços de busca específicos, geralmente atrelados a uma forma de base de dados, se mostrou necessária a inclusão de uma forma de se trabalhar com tais serviços. A API GeoIP2 da empresa Maxmind fornece capacidades de obtenção de informações GEOIP. Para que tal API opere é necessária uma base de dados de consulta local. Ressalta-se no entanto, que foi utilizada a versão gratuita de tal base de dados, conhecida como GeoLite2 Free. Para uma noção

das diferenças entre as bases de dados gratuitas e comerciais, refere-se a seguinte página da Maxmind: <<https://www.maxmind.com/en/geolite2-commercial-redistribution>> .

Um outro detalhe a ser citado é que, conforme afirmado na documentação da própria API¹ até o momento da finalização deste trabalho, as localizações são inerentemente imprecisas. As mesmas são geralmente referidas por grandes centros populacionais, mesmo porque tais serviços são baseados em parte da teoria por trás do endereçamento IP. No entanto, para os fins deste trabalho, tal ferramenta se mostrou adequada.

5.1.3 Apache Commons

As bibliotecas Apache Commons fornecem funcionalidades para a linguagem Java. As mesmas são um projeto Open Source, fomentado pela Apache Software Foundation. As bibliotecas *commons-lang* e *commons-net* foram incluídas pelo fato de serem dependências da OpenCSV.

5.1.4 dnspython

Utilizou-se, para o caso de sucessão de consultas DNS, um programa específico na linguagem Python, que recebeu a denominação de querydns. Trata-se de um programa simples para obtenção de informações de DNS na forma textual. O mesmo é invocado pelo programa principal, e opera para este em vias de retornar as informações sobre o endereço que lhe foi passado como parâmetro. O querydns utiliza de métodos do pacote *dnspython* para resolução de DNS, que se encontra disponível em: <<http://www.dnspython.org/>>.

Um dos empecilhos que vieram a se manifestar no decorrer deste trabalho foi justamente a motivação para o desenvolvimento e uso de tal programa em Python. Ocorre que as bibliotecas (incluindo de terceiros) e métodos para operação com DNS da linguagem Java apresentaram certa carência em sua documentação. De forma prática, a classe *InetAddress* (padrão do pacote de rede de Java) foi, inicialmente, cogitada como meio para a implementação de tais funcionalidades. No entanto, a mesma apresentou problemas na busca por servidores de nomes, bem como, pouca quantidade das funcionalidades requeridas.

Quando consultada, sua documentação não apresentou esclarecimentos suficientes (e até mesmo incoerências, quando se buscava em exemplos pela *web*) para a elaboração de uma solução. Uma outra solução em Java, o pacote *dnsjava*, foi considerado como substituto. Porém durante o período do desenvolvimento da aplicação ocorreu que tal pacote se encontrava em transição para um novo repositório, o que implicou em erros durante a inclusão do mesmo e, na prática, inexistência de uma documentação satisfatória.

¹ Disponível em: <<https://maxmind.github.io/GeoIP2-java/>>

5.1.5 pyasn

Para o complemento das informações acerca do DNS, no caso proposto, é necessária alguma forma de se consultar ASNs. Para isto, o querydns utiliza-se do pacote pyasn, disponível em: <<https://github.com/hadiasghari/pyasn>> . O mesmo opera nos moldes de um programa de consulta a uma base de dados, esta composta pelas definições dos ASNs. Uma das vantagens que é conferida pelo pyasn é que o mesmo possibilita consultas offline, através de uma base de dados fornecida por um utilitário próprio do pacote.

5.2 Aplicação de princípios de Engenharia de Software

Se tratando do uso de boas práticas em engenharia de software, procurou-se durante o desenvolvimento do trabalho manter as mesmas com certo pragmatismo. Técnicas como simplificação de modificações e desacoplamento de código foram prezadas durante o desenvolvimento, bem como o uso de versionamento. Houve a preferência pela objetividade e simplicidade neste período, tendo como razão para isto é que o próprio caráter deste trabalho denota a necessidade pela objetividade, bem como, o fornecimento de uma ferramenta que possa futuramente ser ampliada.

5.3 Estruturação

Nesta seção é apresentada a formulação arquitetural do software desenvolvido. Explana-se aqui a respeito de decisões de design e também demonstra-se de forma abstrata algumas das funções implementadas neste sistema de classificação de URLs. As partes componentes deste sistema podem ser categorizadas em dois escopos maiores: Extração de *features* e classificação de URLs. Vale notar que apesar de poderem ser organizadas em tais escopos, todas as partes são interdependentes.

Para este sistema, extração de *features* diz respeito à todo e qualquer trecho de código relativo ao processamento de URLs a fim de se obter delas informações específicas. Atribui-se também, como relativo à extração de *features*, os trechos de código para armazenamento destas informações antes da eventual classificação. Complementando ainda, pode-se afirmar que a finalidade deste processo é gerar saídas, as quais serão usadas como entradas para funções classificadoras (AMIRI; AKANBI; FAZELDEHKORDI, 2014).

A classificação de URLs, se define através dos trechos de código relativos à aplicação de alguma técnica de IA para que, utilizando os dados pré-processados durante a extração, uma URL venha a ser classificada. Engloba-se ainda, métodos para armazenamento e exibição de resultados. Ademais, existem também alguns métodos mais triviais, que são utilizados para ambos escopos.

Constando que Java é uma linguagem Orientada à Objetos (OO), esta implemen-

tação está fortemente embasada em conceitos como Classes e Objetos. Isto permite além da organização do código, melhor capacidade de formulação e compreensão deste.

O diagrama de classes correspondente ao sistema desenvolvido se encontra disponível no Apêndice E.

5.3.1 Classes Handler

As classes denominadas Handler tratam de operações naquilo que pode ser entendido como uma camada de baixo nível do sistema. São utilizadas para carregamento/gravação de arquivos, consultas às bases de dados, serviços em rede e chamadas de processo. A classe denominada DNSHandler, por exemplo, é responsável pela chamada ao programa querydns, atribuindo-lhe uma URL para obtenção de informações do DNS. Com o recebimento de tais informações, estas são organizadas em uma lista e repassadas a uma próxima etapa de processamento.

As classes Handler se encontram organizadas primariamente pelos tipos de entidade com o qual operam. Se relacionam diretamente com as classes *feature* extractor, na sucessão lógica do programa. Sua invocação se dá, em sua maior parte no trecho principal do programa.

5.3.2 Classes Feature Extractor

Já se tratando das classes Feature Extractor, como o nome implica, são tratadas as operações de extração de *features* a partir dos conjuntos de dados fornecidos às mesmas. Para cada *feature* específica, um método existe para sua extração. Tais métodos são combinados com *wrappers* mais gerais, que combinam chamadas à múltiplos métodos de uma classe, a fim de simplificá-las.

A forma como os dados são dispostos na saída dos *wrappers* é através de listas de literais (do tipo String, em Java). Utilizar tal formato permite manter certo padrão no processamento das saídas, dos mais variados tipos de dados. No entanto, esta forma de representação vem a tornar complexo para se operar tipos numéricos onde há a representação de ponto flutuante, como o tipo Float, por exemplo. Porém, como no domínio explorado não houve a necessidade de se trabalhar com *features* numéricas que não inteiras, deu-se permissividade ao uso de tal implementação.

Ainda sobre a extração de *features*, é interessante observar que alguns trabalhos (James, Sandhya e Thomas (2013)) constroem uma espécie de representação externa do conhecimento, armazenando os resultados de sua extração em formatos como CSV por exemplo. Para este trabalho foi adotada uma estratégia similar onde, após o processo de extração e anteriormente à classificação, as *features* são armazenadas em um arquivo CSV. Uma das vantagens conferida por essa abordagem diz respeito à portabilidade e

independência de serviços externos necessários à extração (uma vez que esta já tenha sido efetuada).

5.3.3 Utilities

Na classe Utilities foram implementados métodos de uso geral. Métodos desta classe são tão genéricos quanto possíveis, para que estes possam prover funcionalidade a uma diversidade de classes no programa. Como exemplos de métodos utilitários implementados, transparecem métodos de exibição de listas, representação de URLs como IPs de *host*, exibição de espaçamento e alguns outros.

5.3.4 Classe Intelligence

A classe Intelligence é a responsável por intermediar o programa e as chamadas à API WEKA. Para cada técnica específica, reside nesta classe um método. As chamadas a tais métodos requerem como parâmetro o necessário para efetuar etapas de treinamento e classificação, além de também ser possível utilizar de alguns parâmetros para alguma especificação para os processos. Detalhamentos acerca dos métodos serão apresentados posteriormente.

5.3.5 Programa principal

Como é de praxe em programas na linguagem Java, a classe que contém o método main é a responsável pela linha principal de execução do programa. O método main age como um controlador principal, intermediando as invocações dos demais elementos e repassando conteúdo entre as mesmas.

5.4 *Feature engineering* e modelagem

No contexto de *machine learning*, *feature engineering* ou engenharia de características em uma tradução livre, diz respeito à busca pela melhor forma de se representar seus dados a fim de sustentar uma aplicação em particular. É também uma das principais tarefas dos profissionais da área, pois a forma como os parâmetros são representados pode influenciar bastante na performance de um modelo de aprendizagem supervisionada (MÜLLER; GUIDO et al., 2016). Nesta seção são apresentados, além dos detalhamentos acerca das *features* utilizadas, a forma como as quais foram efetivamente trabalhadas no software desenvolvido.

Para este trabalho, as *features* vêm a representar aspectos de uma URL que possam denotar alguma ameaça, tal qual definido no Capítulo 2. É interessante ressaltar que os trabalhos anteriores a este já exploraram uma vasta gama de *features* distintas, o que vêm

a ser bastante proveitoso pois além de facilitar a busca por *features*, possibilita também uma forma de nortear o desenvolvimento de outras novas.

5.4.1 *Features* anteriormente exploradas

Tomando por base algumas noções de trabalhos anteriores (Capítulo 4), é possível compor um conjunto inicial de *features* à explorar, relativas ao problema. É interessante ressaltar que o conjunto de *features* exploradas em trabalhos anteriores é composto de *features* selecionadas sem qualquer favorecimento ou preferência por algum autor específico. Foram levados em consideração fatores como a forma e detalhamento com o qual se encontravam descritas em seu trabalho inicial. Também se considerou o fator da viabilidade da implementação, onde se tinha a preferência por selecionar àquelas que apresentassem uma simplicidade suficiente aliada à sua real necessidade.

Um outro fator de influência na decisão acerca do uso de uma *feature* em específico se tratou da quantidade de trabalhos que comprovadamente exploraram a mesma. Isto também possibilitou ter uma boa margem de contraste entre trabalhos, tornando possível asserções interessantes sobre os resultados à serem obtidos. Não houve qualquer repúdio às *features* pelo tipo de informação que abrigam, considerando as capacidades da API do WEKA para o trato de tal situação.

Ao todo, se tratando das *features* de trabalhos anteriores, um total de 16 foram exploradas. As mesmas se encontram descritas na Tabela 1.

Tabela 1 – *Features* anteriormente exploradas

Nomenclatura	Definição	Baseada em
Tamanho do nome de domínio	<i>Feature</i> numérica, com a quantidade de caracteres que compõe o domínio da URL.	Canali et al. (2011) ; Eshete, Villafiorita e Weldemariam (2012) ; Ma et al. (2009)
Tamanho do nome do arquivo	Também numérica, com a quantidade de caracteres que compõe o nome do recurso sendo acessado.	Canali et al. (2011) ; Eshete, Villafiorita e Weldemariam (2012)
Presença de IP	Checagem da presença de endereço(s) IP na URL. É uma <i>feature</i> booleana, onde se assume o valor 0 para a presença de IP e 1 para a sua ausência.	Canali et al. (2011)

Número de porta	Checagem da presença de números de porta de acesso na URL. <i>Feature</i> booleana, onde se assume o valor 0 para a presença de número de porta e 1 para a sua ausência.	Canali et al. (2011)
Presença de subdomínio	Verifica se há algum subdomínio distinto na URL em questão. <i>Feature</i> booleana, assumindo o valor 0 (Condicionado a True) para subdomínios “www” e 1 para qualquer outro.	Canali et al. (2011)
URL relativa	Averiguação se a URL é ou não relativa (independente da base do domínio). É uma <i>feature</i> booleana, onde 0 representa que a URL em questão é relativa e 1 que a mesma não o seja.	Canali et al. (2011)
TLD	Resultado da classificação do TLD de acordo com os testes específicos (quantidade de caracteres, por exemplo). <i>Feature</i> booleana, onde se assume o valor 0 para uma classificação que implica em suspeita e 1 caso contrário.	Canali et al. (2011)
Tamanho do caminho da URL	Outra <i>feature</i> numérica, com a quantidade de caracteres que compõe o caminho completo do recurso sendo acessado pela URL.	Canali et al. (2011) ; Eshete, Villafiorita e Weldemariam (2012)
Nome de arquivo suspeito	Verificação acerca do nome do arquivo sendo acessado. Caso apresente-se algum trecho suspeito no nome do arquivo (tais como nomes de ameaças notórias), a mesma deve ser considerada como uma URL passível de risco. É uma <i>feature</i> booleana onde o valor 0 representa uma URL suspeita e 1 alguma que não carregue este perigo.	Canali et al. (2011)
Código do país	Baseada no uso do GEOIP. Representa o código do país de origem do endereço da URL definido pela base de dados do GEOIP. É uma <i>feature</i> categórica.	Canali et al. (2011) ; Ma et al. (2009) ; Urcuqui et al. (2017) ;

Região	Baseada no uso do GEOIP. Representa o código da região de origem do endereço da URL definido pela base de dados do GEOIP. Também é uma <i>feature</i> categórica.	Canali et al. (2011) ; Ma et al. (2009) ; Urcuqui et al. (2017)
Zona Temporal	Baseada no uso do GEOIP. Representa a zona temporal, no formato utilizado pela IANA, do país de origem do endereço da URL definido pela base de dados do GEOIP. É uma <i>feature</i> categórica.	Canali et al. (2011)
A Records	Baseada no DNS. Address Records são ponteiros para IPs que representam o real endereço de um <i>host</i> a ser acessado. Esta <i>feature</i> numérica representa a quantidade destes registros para determinado <i>host</i> .	Canali et al. (2011) ; Choi, Zhu e Lee (2011)
MX Records	Baseada no DNS. MX Records registram endereços específicos para uso em serviços de email sobre o domínio em questão. Também numérica, representa a quantidade destes endereços.	Canali et al. (2011)
NS Records	Baseada no DNS. Registros acerca dos servidores de nomes responsáveis por um domínio. <i>Feature</i> numérica, que representa a quantidade de servidores.	Canali et al. (2011)
PTR Equals A	Baseada no DNS. Os registros PTR mapeiam de forma inversa o nome e o endereço IP. Esta <i>feature</i> booleana consiste no resultado da comparação do PTR com o primeiro A Record (caso seja possível sua obtenção). O valor 0 representa diferença e 1 representa igualdade.	Canali et al. (2011) ; Ma et al. (2009)

5.4.2 *Features* propostas

Além das *features* exploradas em trabalhos anteriores, houveram também *features* introduzidas neste trabalho. Tais *features* são distintas, considerando os trabalhos referidos por este, representando aspectos observados que também podem vir a indicar que uma URL possa ser maliciosa.

A princípio, uma das *features* introduzidas foi a consideração do número de divisões apresentada no domínio. Apesar de não ser uma relação de causalidade direta, tais divisões se apresentam em grande quantidade em *websites* registrados, conforme mais subdomínios estejam agregados neste ou até mesmo para serviços rodando sobre outras partes do *website*. Esta é uma *feature* numérica.

Uma outra *feature* introduzida foi a consideração da existência de palavras suspeitas no *host*. Tal qual efetuada em uma das *features* propostas, para nomes de arquivos acessados, esta *feature* se baseia na verificação da existência de trechos suspeitos no *hostname* da URL. É uma *feature* booleana, onde 0 representa uma URL com um ou mais trechos suspeitos e 1, uma que não apresente suspeita.

Também foi introduzida a consideração da quantidade de números presentes no *hostname* da URL. Uma artimanha bem conhecida dentre as técnicas de phishing diz respeito à replicação do *hostname*, porém, acrescido de um número. Desta maneira, um usuário desavisado pode vir facilmente a confundir uma URL maliciosa com uma legítima, sendo alvo de um ataque por tal motivo. Esta em si é numérica, contabilizando a quantidade de números que compõe o elemento.

5.4.3 *Features* desconsideradas

Em uma elaboração preliminar, era pretendido o uso de *features* baseadas no uso do WHOIS (Seção 2.1.6) para a composição dos modelos deste trabalho. No entanto, isto se demonstrou como extremamente complexo, por uma série de fatores. A começar, a extração das mesmas se encontra dificultada, visto que cada servidor (comumente hospedados por entidades responsáveis pelo registro de domínios em cada país) pode estruturar os resultados de uma consulta ao WHOIS de forma distinta. Isto inclui, em alguns casos, resultados com idiomas distintos ou fornecimento de informações incompletas quando comparadas com outros resultados. Assim sendo, a análise de consultas é inviável, visto que as mesmas não possuem uma estruturação padronizada.

Um outro fator que prejudica a análise do WHOIS é que a maioria das entidades que fornecem tal serviço impõe limitações à consultas a este. Isto é efetuado a fim de evitar ataques baseados em múltiplos acessos simultâneos, porém, tornando consultas em larga escala um processo bastante lento ou forçando à implementação de técnicas para intercalar os fornecedores. Este conjunto de fatores levou a decisão pela não utilização de

features baseadas em WHOIS. Implementações iniciais de consultas aos serviços WHOIS foram mantidas no sistema, para fins de referência.

5.4.4 Modelo

Sobre os modelos em si, utilizou-se a implementação padrão presente na plataforma WEKA. É esperado que modelos diferentes de *machine learning* forneçam resultados também distintos quando aplicados em uma mesma situação (mesmo problema). Isto considerando é claro casos que não sejam triviais.

5.5 Coleta de informações

A coleta de informações para a composição dos exemplos de treinamento, realizada notavelmente pelas classes Handler, se trata do processo pelo qual se compõe a base da construção de um exemplo neste trabalho. Através de uma lista inicial de URLs, disposta em um arquivo de extensão CSV, são carregadas para o programa principal todas as URLs que se pretende que componham exemplos. Para cada URL, um encadeamento de procedimentos que visa a obtenção da informação desejada é efetuado. A obtenção da informação do WHOIS, por exemplo, é uma tarefa atribuída à classe WhoisHandler.

A seguir, após a coleta de informações em sua forma “bruta”, é necessário um pré-processamento das mesmas, para que venham a compor um exemplo utilizável para a geração de classificadores com as técnicas de *machine learning*. Para tanto, as classes FeatureExtractor vêm a ser utilizadas. As tarefas atribuídas as mesmas referem-se em sua maioria: Ao processamento de string, à construção de listas e ao restante necessário ao processamento de uma informação obtida em vias de compor uma *feature*. De maneira prática, para extrair informações da presença ou não de subdomínio da URL analisada, por exemplo, é utilizada a representação da mesma no programa, bem como uma noção de expressões regulares através de um método na classe “LinkFeatureExtractor”. A saída, para esta exemplificação, é um atributo booleano.

Para ter a construção de exemplos com informações mais recentes a coleta de informações pode ser habilitada durante a execução, o que a torna um processo suficientemente dinâmico no contexto deste trabalho. Após uma primeira efetivação da coleta de informações, no entanto, a mesma já pode ser desabilitada visto que as informações já se encontram disponíveis compondo os exemplos em um arquivo CSV (Seção 5.7). Isto se dá em vias de tornar a execução mais ágil como um todo e também, para que fosse possibilitado certo nível de independência à aplicação (através da não necessidade de uma conexão efetiva com a Internet).

5.5.1 Dependência de outros serviços

A obtenção das informações do DNS e do WHOIS são dependentes de serviços externos, considerando sua natureza. Para o processo de coleta de informações, efetua-se a conexão com os servidores apropriados, para então solicitar informações acerca da URL que se encontra em processamento. Apesar de funcional, tal dependência de sistemas externos pode ocasionar certos impedimentos além de tornar a disponibilidade de acesso à Internet, de certa maneira, um requisito para a efetividade do sistema.

Aplicações que dependem de alguma forma de uma infraestrutura em rede estão inerentemente sujeitas ao caos. Mesmo quando ameaças intencionais (como um ataque malicioso a um servidor, por exemplo), não se demonstrem aparentes, ainda diversos outros tipos de problemas podem ocasionar a inexistência da usabilidade de um sistema que necessite acesso à rede e comunicação com servidores. Quedas de energia ou problemas nos equipamentos físicos de um servidor envolvido, são apenas alguns exemplos disto. Problemas mais abstratos, como erros na codificação de APIs ou a incompletude das mesmas também não são difíceis de se vislumbrar.

Uma situação bastante recorrente se apresenta quando se realiza consultas DNS para obtenção de informações acerca de *websites* que se encontrem *offline*. Comumente, para o caso de *websites* maliciosos, ocorre que os mesmos são retirados do ar por certos períodos de tempo e/ou apagados completamente. Em ambos os casos, isto torna complexa uma análise de situações mais dinâmicas do problema.

Outros problemas podem ocorrer na aplicação, quando a mesma é executada em redes cuja infraestrutura contenha algum firewall. Regras de firewall que limitem a quantidade de conexões ou que inabilitem conexões à servidores específicos, são um exemplo disto. Para mitigar tais situações, a solução mais simples seria a de atribuir uma permissividade especial dentre as regras do firewall, levando em consideração no entanto, que tal permissividade poderia implicar em problemas de segurança para a rede como um todo.

5.6 Classificação

Pelo fato do problema abordado ser considerado inerentemente um problema de classificação, é necessário atribuir aos exemplos um parâmetro de classificação. Para tanto, utilizou-se de uma categorização binária simples, com classes 1 e 0, onde a classe 1 representava um exemplo de treinamento com URL legítima e a classe 0 representa um exemplo com URL maliciosa. Tal atributo classificador é incluído nos exemplos através das classes `FeatureExtractor`.

5.7 Armazenamento

Após o processamento das informações através das classes `FeatureExtractor` e a inclusão do atributo classificador, os exemplos se encontram prontos para uso em treinamento. Contudo, antes de efetivamente virem a compor o procedimento, os mesmos são preparados para ser armazenados em um arquivo CSV. Isto tem como objetivos a simplificação de futuras classificações, o aumento da independência do programa em si, além de facilitar a visualização da representação dos exemplos, em vias de demonstrações.

Além deste armazenamento intermediário, existem também as capacidades conferidas pela API do WEKA. Outros trabalhos como James, Sandhya e Thomas (2013) e Urcuqui et al. (2017), também fazem uso de abordagens similares para o trato de seus exemplos de treinamento.

6 Resultados

Aqui são apresentados os resultados das técnicas de *machine learning* utilizadas na solução do problema, derivados dos conhecimentos desenvolvidos no decorrer do trabalho.

Tendo considerado as definições e noções elementares apresentadas nos capítulos iniciais, das referências a trabalhos anteriores no Capítulo 4 e do detalhamento a nível de implementação no Capítulo 5, agora o foco do trabalho se firma na execução da solução proposta. Além de particularidades acerca da composição dos exemplos, são elucidados os resultados para a aplicação dos diferentes modelos e também quanto a parte de avaliação de performance do sistema.

6.1 Dataset utilizado

Os exemplos de treinamento são agrupados em conjunto, de forma a compor o que na terminologia de IA é conhecido como *dataset*. O mesmo pode ser entendido como um dos pilares de qualquer sistema que se baseie no uso de *machine learning*. É interessante ressaltar que a obtenção de dados para compor *dataset* tem alta influência no fator da complexidade do desenvolvimento de um trabalho. Ainda, para este trabalho em específico, é interessante diferenciar o conjunto de URLs de entrada daquilo que realmente consiste no *dataset* em si. Considerando que se buscou alta independência e a capacidade de adaptação às novas URLs, a aplicação possui o aspecto de ser habilitada à construção do próprio *dataset*, tomando como entrada um conjunto de URLs (Seção 5.3).

Se tratando de um problema já explorado anteriormente (Capítulo 4), existe uma boa quantidade de *datasets* disponíveis para a solução do problema de classificação de URLs. Mesmo alguns trabalhos que não tornam seus *datasets* disponíveis, comumente fornecem técnicas e fontes que podem ser utilizadas para a obtenção de *datasets*. Uma observação interessante é a de que a maior complexidade na construção do *dataset* para este problema diz respeito ao problema de encontrar as URLs maliciosas em si, por conta dos riscos que isto impõe.

O *dataset* utilizado foi composto por *features* extraídas de URLs provenientes de diversas fontes. A começar, um serviço utilizado foi o Phishtank¹, que trata de reportar URLs que se apresentem como ameaças baseadas em phishing ou até mesmo outros tipos. Este serviço fornece um *dataset* constantemente atualizado, onde constam um número muito grande de URLs maliciosas. Outro conjunto de URLs que se apresentou como bastante útil foi o de Sharma e Kumawat (2016), que se encontra disponibilizado na plataforma Github. O mesmo se trata de um conjunto de URLs maliciosas e legítimas que foi utilizado em um trabalho na mesma temática de classificação de URLs. Ainda, o

¹ <<https://www.phishtank.com/>>

trabalho de Urcuqui et al. (2017) também fornece um *dataset* bastante completo para uso, disponibilizado através da plataforma Kaggle².

Também foi explorado o uso de *datasets* com diferentes quantidades de URLs em sua composição (tamanhos distintos). Os tamanhos considerados foram de 200, 1000, 2000 e 3000 exemplos. Os conjuntos foram compostos por partes divididas igualmente, entre URLs maliciosas e legítimas, tendo desta maneira, metade dos exemplos em cada classificação. Note que, por serem tomados a partir da mesma fonte de URLs, tem-se que cada conjunto de tamanho maior engloba os exemplos contidos nos de tamanho menor (em uma terminologia mais clara, utilizando de um exemplo, o conjunto com 200 URLs está contido no conjunto de 1000).

6.2 Detalhamento da execução

Nesta seção são apresentados os resultados obtidos pela aplicação das técnicas, bem como, os detalhes acerca da forma como os modelos foram treinados, executados e verificados. Visto que é pretendida uma forma de se comparar os modelos e evidenciar aquele que mais se mostrou adequado à solução do problema, uma forma de mensurar tal adequação é necessária. Para que fosse possível tal comparação, foi utilizada a precisão de um modelo sobre o conjunto de treinamento.

A precisão é medida fazendo uso da classe *Evaluation*, presente no WEKA. É definido como acerto a classificação correta de um exemplo (e erro o contrário), de forma que estes acertos possam ser contabilizados como componentes de um todo, geralmente em um valor normalizado. Considerando uma quantidade de acertos C e um número de instâncias I , a fórmula do cálculo da precisão é como segue.

$$A = (C/I) * 100$$

A primeira forma de verificação quanto à precisão de um modelo consiste na retroalimentação do classificador, com os próprios exemplos de treinamento. No entanto a validação utilizando, diretamente, os próprios exemplos de treinamento como entrada não oferece uma boa noção das vantagens conferidas por determinado modelo, em especial, acerca da capacidade de generalização. Para tanto, foi empregada a técnica de validação cruzada (do termo em inglês *cross-validation*), inerente à própria classe *Evaluation* do WEKA. Tal técnica é mais refinada e permite mensurar de forma mais adequada a generalização de um modelo, pois parte do conjunto de treinamento é retida e utilizada para a verificação em si.

² <<https://www.kaggle.com/>>

Para apresentar de forma mais objetiva os resultados acerca de um classificador, um artifício de utilização bastante comum são as chamadas matrizes de confusão. Uma matriz de confusão (também denominada matriz de erro) consiste em uma tabela onde cada uma de suas linhas representa um resultado da classificação, enquanto que suas colunas representam a classificação esperada. Desta forma, é possível a representação das classificações efetuadas, tanto as corretas (Verdadeiros) como as incorretas (Falsos). A tabela descrita a seguir (Tabela 3) permite uma melhor visualização deste conceito, para o caso onde tem-se uma classificação binária.

Tabela 3 – Matriz de confusão 2x2 exemplificada

Classificação	Class. A	Class. B
Class. A	Verdadeiros Positivos	Falsos Positivos
Class. B	Falsos Negativos	Verdadeiros Negativos

No que tange a noção de arredondamento e representação de valores, a fim de obter uma melhor visibilidade, os termos percentuais a seguir consideram 2 casas de precisão. A porcentagem máxima descrita é de 100%.

6.2.1 J48

O J48, já detalhado anteriormente (Seção 3.3.4), consiste em uma técnica baseada na geração de árvores de decisão. A mesma é implementada no WEKA no pacote `weka.classifiers.trees.J48`, onde também se encontram outros classificadores similares. A formulação do J48 utilizada no âmbito deste trabalho é a padrão do WEKA, não havendo modificação substancial descrita sobre o mesmo.

A tabela a seguir descreve os resultados obtidos com a aplicação de um modelo baseado no J48 para a solução do problema. Conforme descrito em uma seção anterior (Seção 6.1), diferentes quantidades de exemplos para treinamento foram consideradas.

Tabela 4 – Resultados do J48

Exemplos	Precisão	
	Conjunto de treinamento	Validação cruzada
200	90.5%	81.5%
1000	92.5%	89.1%
2000	93.8%	90.05%
3000	96.13%	92.8%

Na Tabela 4, é perceptível que apesar de apresentar resultados relativamente altos, a aplicação do J48 generalizou de forma intermediária para o conjunto com poucos

exemplos. Isto era esperado, considerando a natureza das técnicas baseadas em árvores de decisão simples. Vale a ressalva de que, na utilização de conjuntos de exemplos na casa dos milhares, tanto a precisão considerando todos os exemplos quanto a precisão por validação cruzada apresentaram pouca diferença (para todos os casos, pouco mais de 3%). É de se notar, porém, que a execução para 3000 exemplos, apresentou resultados consideravelmente altos.

Sobre a classificação dos exemplos em si, as matrizes de confusão, considerando a execução da validação cruzada, estão representadas a seguir. Para cada conjunto de exemplos com quantidade específica, uma representação distinta foi elaborada. A Tabela 5 representa, por exemplo, a execução para 200 exemplos.

Tabela 5 – Matriz de confusão para 200 exemplos

	0	1
0	83	17
1	20	80

A Tabela 6 representa a matriz de confusão resultante para a execução com 1000 exemplos.

Tabela 6 – Matriz de confusão para 1000 exemplos

	0	1
0	443	57
1	52	448

A Tabela 7 representa a matriz de confusão resultante para a execução com 2000 exemplos.

Tabela 7 – Matriz de confusão para 2000 exemplos

	0	1
0	894	106
1	84	916

E a Tabela 8 representa a execução para considerando 3000 exemplos.

Tabela 8 – Matriz de confusão para 3000 exemplos

	0	1
0	1403	97
1	119	1381

Por se tratar de uma técnica baseada na geração de uma árvore de decisão, é possível a representação do conhecimento gerado pela aplicação do algoritmo através da árvore em si. Novamente, a plataforma WEKA vem a auxiliar nos processos deste trabalho, tornando possível a obtenção da árvore gerada representada em forma textual simples. A árvore gerada na execução para 3000 exemplos se encontra disponível no Apêndice D.

6.2.2 SVM

Se tratando de uma SVM (Seção 3.3.2), as mesmas são consideradas uma técnica bastante efetiva, baseada na noção de separação linear de conjuntos. Por padrão, até a versão atual da plataforma WEKA (3.8.x), as SVMs não estão incluídas, considerando sua forma mais simples. No entanto, o WEKA permite uma forma simples para a sua inclusão, para isto, necessitando da extensão LibSVM (de Chang e Lin (2011)). São implementadas ligações (*bindings*) no WEKA que permitem o uso de tal extensão. A formulação de SVM utilizada neste trabalho também segue seu padrão, não possuindo modificações.

A tabela a seguir descreve os resultados obtidos com a utilização de um modelo baseado nas SVMs para a solução do problema.

Tabela 9 – Resultados do SVM

Exemplos	Precisão	
	Conjunto de treinamento	Validação cruzada
200	94%	86%
1000	91.5%	89%
2000	89.3%	87.4%
3000	90.3%	89.33%

Na Tabela 9, com os resultados do SVM, é observável que para 200 exemplos a precisão através do conjunto de treinamento mostrou bons resultados, o maior dos testes com o SVM, porém a generalização foi prejudicada visto que a precisão através da validação cruzada foi a menor descrita na tabela. Já para o caso de 2000 exemplos, ambas as formas de cálculo denotaram alta precisão, porém com valores menores quando comparados com os demais resultados. O caso de 1000 exemplos, no entanto, apresentou-se um pouco melhor.

Para 3000 exemplos, porém, a precisão através da validação cruzada atingiu um alto resultado nos testes com o SVM, sendo desta maneira, o conjunto que melhor generalizou. Tais resultados, no entanto, foram suficientemente próximos aos alcançados para o conjunto de 1000. Uma individualidade interessante do uso do SVM é perceptível, ao se observar os resultados de um panorama geral. É uma característica inerente à técnica em si operar melhor sobre conjuntos de treinamento com pouca (porém suficiente) quantidade de exemplos (Subsubseção 3.3.2.2).

Considerando os exemplos em si, para cada execução com conjunto distinto, tem-se cada uma das matrizes a seguir. A Tabela 10 contém os resultados para 200 exemplos.

A Tabela 11 representa a matriz de confusão resultante da execução do modelo baseado em SVM, considerando 1000 exemplos.

Tabela 10 – Matriz de confusão para 200 exemplos

	0	1
0	91	9
1	19	81

Tabela 11 – Matriz de confusão para 1000 exemplos

	0	1
0	428	72
1	38	462

Já a Tabela 12 representa a matriz de confusão resultante da execução considerando 2000 exemplos.

Tabela 12 – Matriz de confusão para 2000 exemplos

	0	1
0	857	143
1	109	891

E a Tabela 13 representa a execução para considerando 3000 exemplos.

Tabela 13 – Matriz de confusão para 3000 exemplos

	0	1
0	1318	182
1	138	1362

6.2.3 Random Forest

Já no que tange a aplicação da técnica Random Forest (Seção 3.3.3), foi utilizado o pacote `weka.classifiers.trees.RandomForest`. A técnica em si, baseada no conceito de Bagging, foi também mantida em sua implementação mais comum, não havendo modificações sobre as entidades necessárias à sua execução.

A Tabela 14, expressa a seguir, descreve os resultados obtidos com a aplicação do Random Forest.

Interpretando os resultados da tabela, fica explícito que o modelo baseado no Random Forest teve resultados excepcionalmente altos, considerando todos os diferentes conjuntos de exemplos. Um destaque para a validação com o conjunto de treinamento composto por 2000 exemplos, que atingiu a marca de 100% de precisão, enquanto que a validação cruzada sobre o mesmo conjunto também teve um resultado alto, classificando com 93.3% de precisão. Também os resultados para 3000 exemplos foram bastante singulares, atingindo a marca de 94.77% de precisão para a validação cruzada, e um

Tabela 14 – Resultados do Random Forest

Exemplos	Precisão	
	Conjunto de treinamento	Validação cruzada
200	99%	91%
1000	99.8%	93.1%
2000	100%	93.3%
3000	99.9%	94.77%

valor pouco menor que a marca dos 100% para a validação com o próprio conjunto de treinamento.

Considerando as características da técnica Random Forest (Subsubseção 3.3.3.3), há de se apreciar que para o domínio do problema, modelos baseados na mesma são bem interessantes. Também é de grande valia o fato da técnica operar particularmente bem sobre tarefas de classificação, e sobre conjuntos de dados com atributos que distinguem em tipo.

Os resultados acerca das classificações podem ser visualizados a seguir, através das matrizes de confusão. A Tabela 15 representa os resultados para a execução com 200 exemplos.

Tabela 15 – Matriz de confusão para 200 exemplos

	0	1
0	93	7
1	11	89

Já se tratando da execução para 1000 exemplos, a Tabela 16.

Tabela 16 – Matriz de confusão para 1000 exemplos

	0	1
0	468	32
1	37	463

Para 2000 exemplos, a tabela Tabela 17.

Tabela 17 – Matriz de confusão para 2000 exemplos

	0	1
0	946	54
1	80	920

E finalmente, a Tabela 18 representa a execução considerando 3000 exemplos.

Pelas Random Forests possuírem entre suas desvantagens o fato de serem técnicas ditas *black box*, na prática, isto significa que é inviável a visualização do conhecimento

Tabela 18 – Matriz de confusão para 3000 exemplos

	0	1
0	1425	75
1	82	1418

gerado pelas mesmas em sua forma de representação pura. Isto ocorre, pois além da metodologia de Bagging, o processo de análise iria requerer examinar cada uma das árvores, sendo que o número de árvores geradas é suficientemente grande. No entanto, isto não chega a ser um empecilho para formas mais sutis de análise dos resultados da técnica.

O conceito de importância de atributos diz respeito, para classificadores baseados em árvore, o quão efetivamente um atributo operou sobre o modelo ao qual pertence. Especificamente para o Random Forest, tal importância pode ser mensurada através do uso da média no decrescimento da impureza, sendo esta a forma implementada no WEKA. Isto vem a significar que, quanto menor o valor da média no decrescimento da impureza, maior a importância do atributo. De uma forma mais prática, a importância de atributos possui a informação de quão significativo é o impacto de um atributo sobre a decisão em si. A Tabela 19 representa os resultados do cálculo de importância dos atributos para a execução do modelo com validação cruzada e com 3000 exemplos.

6.3 Detalhamento de performance

O ambiente no qual foram sucedidos os testes da aplicação consiste em uma plataforma comum. O processador é um Intel I5 modelo 8265U, de arquitetura 64 bits. A memória consta como 8 Gigabytes de memória do tipo DDR4. Não foram efetuadas modificações no hardware que pudessem vir a causar algum efeito sobre a aplicação em si. No que diz respeito ao software, os sistemas operacionais utilizados durante o desenvolvimento foram o Linux Mint 19³ e o Ubuntu 18.04.3 LTS⁴ e durante os testes, o Ubuntu 18.04.3 LTS.

As quantidades de exemplos foram determinadas como 2000 e 3000, por serem os valores de maior demanda de processamento entre os testes efetuados. Para a reflexão acerca do consumo da aplicação, foi utilizada a plataforma de perfilamento de projeto do Apache Netbeans, que forneceu uma noção detalhada da performance acerca do consumo de memória. Ambos resultados para tempo e memória são aproximados para valores com precisão de duas casas decimais.

Por último mas não menos importante, os cálculos de performance aqui expostos consideram apenas o treinamento e a classificação de exemplos, não levando em conta a

³ <<https://linuxmint.com/>>

⁴ <<https://ubuntu.com/>>

Tabela 19 – Tabela de importância de atributos para 3000 exemplos

Atributo	Importância do atributo	Número de nós utilizando o atributo
Tamanho do nome de domínio	48%	4106
Tamanho do nome do arquivo	45%	3536
Tamanho do caminho da URL	45%	2620
Presença de subdomínio	40%	1111
Presença de IP	39%	51
MX Records	37%	1959
Região	37%	925
TLD	35%	1112
Divisões no domínio	34%	1536
NS Records	34%	1785
A Records	30%	1079
Quantidade de números no hostname	29%	822
Zona Temporal	24%	261
Código do país	23%	154
Palavras suspeitas no hostname	16%	159
Número de porta	0%	0
URL Relativa	0%	0
PTR Equals A	0%	0
Nome de arquivo suspeito	0%	0

Legenda: Em fonte normal, os atributos baseados em trabalhos anteriores e em negrito, estão representados os atributos introduzidos neste trabalho

obtenção de informações e a construção do *dataset*. A Tabela 20 apresenta as medidas em tempo e memória consumidos para a execução com 2000 exemplos.

Tabela 20 – Valores aproximados das medidas de performance para 2000 exemplos

Técnica	Tempo (segundos)	Memória (Megabytes)
J48	1.56	33.26
SVM	1.92	55.9
Random Forest	3.09	130

Já a Tabela 21 descreve as medidas para a execução com 3000 exemplos.

Tabela 21 – Valores aproximados das medidas de performance para 3000 exemplos

Técnica	Tempo (segundos)	Memória (Megabytes)
J48	2.58	49.48
SVM	4.17	84.21
Random Forest	4.85	162.99

6.4 Extração e construção dos exemplos

No que diz respeito especificamente à etapa de obtenção de informações das URLs e construção dos exemplos, para os testes efetuados, a mesma foi a que teve maior consumo de tempo na execução da aplicação. Isto era esperado, considerando as necessidades de acesso à rede e também a característica sequencial da implementação. A respeito do acesso à rede, o mesmo utilizou de uma conexão doméstica padrão, com velocidade média de 300kbps. Os resultados das medidas de tempo tomadas em uma média de 3 execuções para os casos de 200, 2000 e 3000 exemplos está representado na tabela Tabela 22.

Tabela 22 – Tempo médio de execução da extração de *features*

Quantidade de exemplos	Tempo (Horas e minutos)
200	0h e 11m
2000	0h e 59m
3000	1h e 34m

6.5 Discussão

Nesta seção, é pretendida uma apreciação dos resultados apresentados anteriormente, tanto para a solução do problema em si quanto para a performance alcançada. Vale a ressalva de que apesar de leves discrepâncias, a precisão alcançada foi relativamente alta para todas as técnicas de *machine learning* utilizadas. Mais do que isso, é notável também que até mesmo os resultados contabilizados para poucos exemplos indicaram uma precisão acima da casa dos 80%, para todas as técnicas aplicadas.

Dos 3 modelos considerados, o modelo baseado no Random Forest foi terminantemente o melhor classificador que se apresentou para este problema. Considerando a verificação da precisão através da validação cruzada, o classificador Random Forest quando fazendo uso do *dataset* com 3000 exemplos se demonstrou o mais efetivo no âmbito deste trabalho. Os demais resultados obtidos por este classificador foram igualmente notáveis (Tabela 14), considerando uma média geral, foram os de maior gradação. É reafirmado, no entanto, que também é válido debater acerca dos outros classificadores.

Considerado o modelo mais simples envolvido neste trabalho, o J48 obteve resultados surpreendentemente efetivos para classificação. O seu resultado alcançado através do treinamento com 3000 exemplos, a título de citação, foi equiparável aos resultados do Random Forest. Se tratando do SVM, apesar de seus resultados terem sido razoavelmente menores, ainda podem ser considerados bastante altos. Um ponto interessante aqui diz respeito ao fato de que para o SVM, os resultados da precisão através de validação cruzada e retroalimentação do conjunto de treinamento chegaram bem próximos uns dos outros, com exceção do caso de 200 exemplos.

Por fim, se tratando dos parâmetros de tempo e memória e considerando o estado atual da computação, faz-se a ressalva de que todas as medidas de performance indicam que as soluções poderiam ser executadas sem problemas aparentes em diversos dispositivos compatíveis. O J48 se demonstrou com a melhor performance, seguido pelo SVM com resultados intermediários e o Random Forest, com maior consumo em tempo e memória. Todos estes, no entanto, obtiveram resultados bastante próximos uns dos outros.

7 Conclusão

Finalizando este trabalho, foi produzido um levantamento geral sobre o mesmo e também sobre a elaboração de noções acerca de possibilidades para trabalhos futuros.

Na amplitude deste trabalho, interessou-se pelo problema de classificação de URLs e os diversos segmentos que o compõe. Adotou-se uma abordagem mais pragmática, alcançada através do desenvolvimento de uma solução própria para tal classificação. Além da solução para o problema em si, também foram apresentadas soluções para situações menores apresentadas ao longo do desenvolvimento da mesma, especificamente, no que tange a obtenção de informações e construção de bases para treinamento.

De início, foram apresentados os diversos conceitos e definições inerentes a tal temática, como àqueles a respeito da *web*, segurança, IA e mesmo àqueles provenientes diretamente de ramos da matemática. A seguir, os trabalhos que anteriormente exploraram este mesmo problema foram elucidados, bem como, suas contribuições para a solução do problema. Numa próxima etapa, mostrou-se os conceitos mais técnicos sobre a implementação da solução em si, bem como as metodologias utilizadas e as noções sobre a operação de tal solução. Vale citar também que neste ponto foram discutidas as *features* que vêm a compor os modelos elaborados. Finalmente, os resultados obtidos com a solução foram descritos e avaliados, após diversos testes descritos sobre circunstâncias distintas.

Os exemplos que formam o conjunto de treinamento são compostos por um conjunto de 19 *features* distintas, 16 destas advindas de trabalhos anteriores e 3 elaboradas para este trabalho. Cada *feature* corresponde à um aspecto da URL considerada, que venha a dar certa significância sobre o teste da mesma. Os modelos sobre o qual as classificações são efetuadas são baseados nas técnicas de *machine learning* J48, SVM e Random Forest, tendo esta última alcançado resultados excelentes para o problema em si.

Se tratando da implementação inerente ao trabalho, a mesma é composta pelo classificador e pelos extratores de *features*. A implementação por si só foi coerente com o que estava proposto para o trabalho. Foram utilizadas as funcionalidades da plataforma WEKA para compor o classificador, implementando o necessário para o alinhamento com as técnicas propostas. Os extratores, por sua vez, operam em vias de retirar o que é essencial sobre os dados em sua forma pura, formando exemplos. É bem interessante notar ainda que ambas as partes foram implementadas em vias de serem executadas juntas ou independentemente uma da outra.

7.1 Trabalhos futuros

Por fim, ao longo do desenvolvimento deste trabalho alguns possíveis aprimoramentos foram deixados em aberto e são melhor comentados à seguir. Devido à questão do

tempo, tais possibilidades foram conjecturadas porém não implementadas, sendo consideradas passíveis a uma possível continuação deste trabalho. A começar, uma abordagem para a composição de outro conjunto de *features* ou mesmo para servir como um sistema inteiramente novo seria a utilização de uma técnica similar à Term Frequency - Inverse Document Frequency (TF-IDF). A mesma se define como uma técnica específica para análise de termos em textos, valorando os mesmos conforme a quantidade de ocorrências, porém considerando também o quão comum um termo se apresenta sob um conjunto de múltiplos textos.

Uma variante desta técnica poderia ser implementada, de modo a considerar não os termos incomuns, mas sim, os que se apresentam mais comuns dentre textos distintos. Ao aplicar tal técnica sobre respostas obtidas no acesso a um conjunto de URLs maliciosas, há de se obter uma boa noção de terminologias comuns no léxico das mesmas. É notável que não é necessária a execução de nenhum código presente nas respostas obtidas, apenas a sua obtenção em texto puro. Isto permite manter também a segurança durante a execução desta possível forma de análise.

Outro ponto que demonstra uma possibilidade interessante para futuros desenvolvimentos diz respeito à classificação de uma URL quanto ao tipo específico de ameaça que a mesma corresponde. Dado que são diversos os tipos de ameaça que uma URL maliciosa pode vir a resguardar, é um objetivo interessante a capacidade de se classificar URLs não só acerca da possibilidade de serem ou não maliciosas, mas também a qual ameaça carregam. Além de permitir um melhor curso de ação para se tomar após uma classificação, o desenvolvimento desta abordagem permitiria asserções interessantes acerca das características das URLs. A exploração de outras técnicas de *machine learning* também se demonstram como uma possibilidade relativamente interessante. Também é possível citar neste caso, variações das técnicas já apresentadas neste trabalho.

Uma possibilidade que também foi antecipada durante o desenvolvimento foi a de analisar elementos provenientes de outras camadas da rede que não a de aplicação. De exemplo, foram pressupostas formas de se analisar elementos provenientes da camada de transporte, porém, a análise pode ser estendida à camadas de nível mais baixo. É sensato afirmar que quanto mais próxima ao nível de hardware, a análise demandaria uma quantidade consideravelmente maior de esforço para ser elaborada.

A respeito do classificador baseado em SVM, é possível também efetuar alterações no parâmetro de regularização, em vias de tentar a obtenção de uma maior precisão. Outro elemento que pode ser ajustado no mesmo classificador diz respeito à tolerância a erro, que também pode influir na precisão. Ambas as alterações, no entanto, foram suprimidas como uma possibilidade de um trabalho posterior.

A reimplementação do sistema desenvolvido e a elaboração de uma API para acesso ao mesmo também constam como possibilidades futuras. Junto a isto, uma implementação

para execução mais simplificada em servidores web também é um ponto interessante de exploração.

Ainda se tratando de trabalhos futuros, a elaboração de uma forma mais efetiva de extração de *features* ou mesmo a utilização de técnicas de computação paralela para melhor gerir tal etapa é algo que seria bastante utilizável para o viés deste trabalho. Considerando que efetuar a extração é uma tarefa bastante custosa em tempo, o proveito obtido seria considerável.

No entanto, é acrescentado que o aparato máximo para se combater qualquer ameaça, seja ela virtual ou real, é o uso do conhecimento em conjunto com a prudência. Quando se soma a estes as capacidades das ferramentas computacionais, os resultados tendem a ser superiores.

REFERÊNCIAS

- AFUAH, A. *Internet business models and strategies: Text and cases*. New York: McGraw-Hill, Inc., 2002.
- ALGHAMDI, B.; WATSON, J.; XU, Y. Toward detecting malicious links in online social networks through user behavior. In: IEEE. *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*. Omaha, 2016. p. 5–8.
- AMIRI, I. S.; AKANBI, O. A.; FAZELDEHKORDI, E. *A machine-learning approach to phishing detection and defense*. USA: Syngress, 2014.
- ARENALES, M. et al. *Pesquisa operacional: para cursos de engenharia*. Rio de Janeiro: Elsevier Brasil, 2015.
- BERGER, A. L.; PIETRA, V. J. D.; PIETRA, S. A. D. A maximum entropy approach to natural language processing. *Computational linguistics*, MIT Press, v. 22, n. 1, p. 39–71, 1996.
- BERNERS-LEE, T. Long live the web. *Scientific American*, JSTOR, v. 303, n. 6, p. 80–85, 2010.
- BERNERS-LEE, T.; MASINTER, L.; MCCAILL, M. *Uniform resource locators (URL)*. Austria, 1994.
- BILGE, L. et al. Exposure: Finding malicious domains using passive dns analysis. In: INSTITUTE EURECOM. *Ndss*. Austria, 2011. p. 1–17.
- BLUMENTHAL, M. S.; CLARK, D. D. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology (TOIT)*, ACM, v. 1, n. 1, p. 70–109, 2001.
- BOSE, I.; LEUNG, A. C. M. Unveiling the mask of phishing: Threats, preventive measures, and responsibilities. *communications of the Association for Information Systems*, v. 19, n. 1, p. 24, 2007.
- BRAMER, M. Avoiding overfitting of decision trees. *Principles of data mining*, Springer, p. 119–134, 2007.
- BREIMAN, L. Bagging predictors. *Machine learning*, Springer, v. 24, n. 2, p. 123–140, 1996.
- BREIMAN, L. Out-of-bag estimation. Citeseer, 1996.
- BREIMAN, L. Random forests. *UC Berkeley TR567*, 1999.
- BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001.
- BREIMAN, L. Consistency for a simple model of random forests. Citeseer, 2004.
- BREIMAN, L. et al. *Classification and Regression Trees (Wadsworth Statistics/Probability)*. Belmont: Chapman and Hall/CRC, 1984.

- BURGES, C. J. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, Springer, v. 2, n. 2, p. 121–167, 1998.
- BUSH, V. et al. As we may think. *The atlantic monthly*, v. 176, n. 1, p. 101–108, 1945.
- CANALI, D. et al. Prophiler: a fast filter for the large-scale detection of malicious web pages. In: ACM. *Proceedings of the 20th international conference on World wide web*. Hyderabad, 2011. p. 197–206.
- CARBONELL, J. G.; MICHALSKI, R. S.; MITCHELL, T. M. An overview of machine learning. In: *Machine learning*. Palo Alto: Elsevier, 1983. p. 3–23.
- CARRELL, J. B. Fundamentals of linear algebra. *The University of British Columbia*, 2005.
- CARVAJAL, S. et al. *Estatística Básica*. Rio de Janeiro: Elsevier Brasil, 2015.
- CASTELLS, M. The impact of the internet on society: a global perspective. *F. González, ed*, p. 132–133, 2014.
- CHANG, C.-C.; LIN, C.-J. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, Acm, v. 2, n. 3, p. 27, 2011.
- CHEN, P.-H.; LIN, C.-J.; SCHÖLKOPF, B. A tutorial on ν -support vector machines. *Applied Stochastic Models in Business and Industry*, Wiley Online Library, v. 21, n. 2, p. 111–136, 2005.
- CHI, M.; FENG, R.; BRUZZONE, L. Classification of hyperspectral remote-sensing data with primal svm for small-sized training dataset problem. *Advances in space research*, Elsevier, v. 41, n. 11, p. 1793–1799, 2008.
- CHOI, H.; ZHU, B. B.; LEE, H. Detecting malicious web links and identifying their attack types. *WebApps*, v. 11, n. 11, p. 218, 2011.
- CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995.
- CRANDALL, J. R. et al. The ecology of malware. In: ACM. *Proceedings of the 2008 New Security Paradigms Workshop*. Lake Tahoe, 2009. p. 99–106.
- CURRAN, J. Reinterpreting the internet: James curran. In: *Misunderstanding the internet*. New York: Routledge, 2012. p. 9–39.
- DAIGLE, L. *WHOIS protocol specification*. California, 2004.
- DAWS, R. *Putin outlines Russia's national AI strategy priorities*. 2019. Disponível em: <<https://www.artificialintelligence-news.com/2019/05/31/putin-russia-national-ai-strategy-priorities/>>. Acesso em: 15 jun. 2019.
- EDWARDS, S. *Elements of Information Theory, Thomas M. Cover, Joy A. Thomas, John Wiley & Sons, Inc.(2006)*. New York: Pergamon, 2008.
- EGELE, M. et al. Dynamic spyware analysis. Advanced Computing Systems Professional and Technical Association, 2007.

- EGELE, M. et al. Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks. In: SPRINGER. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Heidelberg, 2009. p. 88–106.
- ELISAN, C. C. *Advanced malware analysis*. United States: McGraw-Hill Education, 2015.
- ELIZONDO, D. The linear separability problem: Some testing methods. *IEEE Transactions on neural networks*, IEEE, v. 17, n. 2, p. 330–344, 2006.
- ESHETE, B. Effective analysis, characterization, and detection of malicious web pages. In: ACM. *Proceedings of the 22nd International Conference on World Wide Web*. Rio de Janeiro, 2013. p. 355–360.
- ESHETE, B.; VILLAFIORITA, A.; WELDEMARIAM, K. Binspect: Holistic analysis and detection of malicious web pages. In: SPRINGER. *International Conference on Security and Privacy in Communication Systems*. Italy, 2012. p. 149–166.
- FOROUZAN, B. A. *Comunicação de dados e redes de computadores*. 4. ed. Porto Alegre: AMGH, 2010.
- FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. *The elements of statistical learning*. Stanford: Springer series in statistics New York, 2001. v. 1.
- GONÇALVES, A. R. Máquina de vetores suporte. *Acesso em 06/2019*, v. 21, 2015.
- GOODMAN, M. *Future crimes: Everything is connected, everyone is vulnerable and what we can do about it*. New York: Anchor, 2015.
- GYONGYI, Z.; GARCIA-MOLINA, H. Web spam taxonomy. In: *First international workshop on adversarial information retrieval on the web (AIRWeb 2005)*. Stanford: [s.n.], 2005.
- HADNAGY, C. *Social engineering: The art of human hacking*. Indianapolis: John Wiley & Sons, 2010.
- HALL, M. et al. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, ACM, v. 11, n. 1, p. 10–18, 2009.
- HARRIS, E. Information gain versus gain ratio: A study of split method biases. In: WILLIAM & MARY. *ISAIM*. Williamsburg, 2002. p. 1–20.
- HINDEN, R. et al. Internet protocol, version 6 (ipv6) specification. 2017.
- HSSINA, B. et al. A comparative study of decision tree id3 and c4. 5. *International Journal of Advanced Computer Science and Applications*, Citeseer, v. 4, n. 2, p. 0–0, 2014.
- HUSTON, G. Exploring autonomous system numbers. *The Internet Protocol Journal*, v. 9, n. 1, p. 2–23, 2006.
- IKINCI, A.; HOLZ, T.; FREILING, F. Monkey-spider: Detecting malicious websites with low-interaction honeyclients. *SICHERHEIT 2008–Sicherheit, Schutz und Zuverlässigkeit. Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik eV (GI)*, Gesellschaft für Informatik e. V., 2008.

IKONOMAKIS, M.; KOTSIANTIS, S.; TAMPAKAS, V. Text classification using machine learning techniques. *WSEAS transactions on computers*, v. 4, n. 8, p. 966–974, 2005.

JAGATIC, T. N. et al. Social phishing. *Communications of the ACM*, v. 50, n. 10, p. 94–100, 2007.

JAMES, J.; SANDHYA, L.; THOMAS, C. Detection of phishing urls using machine learning techniques. In: IEEE. *2013 International Conference on Control Communication and Computing (ICCC)*. Trivandrum, 2013. p. 304–309.

JING, M. *China recruits Baidu, Alibaba and Tencent to AI ‘national team’*. 2017. Disponível em: <<https://www.scmp.com/tech/china-tech/article/2120913/china-recruits-baidu-alibaba-and-tencent-ai-national-team>>. Acesso em: 15 jun. 2019.

KLIMBURG, A. *National Cyber Security Framework Manual*. Tallinn: NATO, 2012.

KOHAVI, R. et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: MONTREAL, CANADA. *Ijcai*. Stanford, 1995. v. 14, n. 2, p. 1137–1145.

KOTSIANTIS, S. B.; ZAHARAKIS, I.; PINTELAS, P. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, v. 160, p. 3–24, 2007.

KOWALCZYK, A. Support vector machines succinctly. *Syncfusion Inc*, 2017.

KRAVETS, D. *U.N. REPORT DECLARES INTERNET ACCESS A HUMAN RIGHT*. 2011. Disponível em: <<https://www.wired.com/2011/06/internet-a-human-right/>>. Acesso em: 07 jun. 2019.

KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a internet uma abordagem top-down*. São Paulo: Pearson Education, 2013.

KURZWEIL, R. et al. *The age of intelligent machines*. Cambridge: MIT press Cambridge, MA, 1990. v. 579.

LISKA, A.; GALLO, T. *Ransomware: Defending against digital extortion*. Sebastopol: "O'Reilly Media, Inc.", 2016.

LORENA, A. C.; CARVALHO, A. d. Introduçaoas máquinas de vetores suporte. *Relatório Técnico do Instituto de Ciências Matemáticas e de Computação (USP/Sao Carlos)*, v. 192, 2003.

MA, J. et al. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In: ACM. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. San Diego, 2009. p. 1245–1254.

MADAKAM, S.; RAMASWAMY, R.; TRIPATHI, S. Internet of things (iot): A literature review. *Journal of Computer and Communications*, Scientific Research Publishing, v. 3, n. 05, p. 164, 2015.

MALOOF, M. A. *Machine learning and data mining for computer security: methods and applications*. Heidelberg: Springer, 2006.

- MANNING, C. et al. The stanford corenlp natural language processing toolkit. In: *ACL. Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. Baltimore, Maryland, 2014. p. 55–60.
- MARCHAND, L. *Multiplicadores de Lagrange: uma aplicação em problemas de otimização global restrita*. Trabalho de Conclusão de Curso — Universidade Federal do Rio Grande, 2016.
- MAZURCZYK, W.; CAVIGLIONE, L. Information hiding as a challenge for malware detection. *arXiv preprint arXiv:1504.04867*, 2015.
- MITCHELL, T. *Machine learning*. Buena Vista: Macgraw-Hill companies, 1997. v. 1.
- MITNICK, K. D.; SIMON, W. L. *The art of deception: Controlling the human element of security*. Indianapolis: John Wiley & Sons, 2011.
- MONARD, M. C.; BARANAUSKAS, J. A. Indução de regras e árvores de decisão. *Sistemas Inteligentes. Rezende, SO Editora Manole Ltda*, p. 115–140, 2003.
- MOSHCHUK, A. et al. A crawler-based study of spyware in the web. In: *NDSS*. Washington: [s.n.], 2006. v. 1, p. 2.
- MÜLLER, A. C.; GUIDO, S. et al. *Introduction to machine learning with Python: a guide for data scientists*. Sebastopol: "O'Reilly Media, Inc.", 2016.
- NILSSON, N. J. *Introduction to machine learning: An early draft of a proposed textbook*. Stanford: USA; Stanford University, 1996.
- NORVIG, PETER & RUSSEL, STUART J. *Artificial Intelligence: A modern approach*. New Jersey: Prentice Hall, 2010.
- PENWARDEN, R. et al. *The Rise of the Smartphone*. Palo Alto: FluidSurveys, 2014.
- PETERSON, L. L.; DAVIE, B. S. *Computer networks: a systems approach*. San Francisco: Elsevier, 2007.
- PROVOS, N. et al. The ghost in the browser: Analysis of web-based malware. *HotBots*, v. 7, p. 4–4, 2007.
- QUINLAN, J. R. Induction of decision trees. *Machine learning*, Springer, v. 1, n. 1, p. 81–106, 1986.
- QUINLAN, J. R. C4. 5: Programming for machine learning. *Morgan Kauffmann*, v. 38, p. 48, 1993.
- RESCORLA, E. Rfc 2818: Http over tls. *Internet Engineering Task Force: http://www.ietf. org*, 2000.
- ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.
- SAHOO, D.; LIU, C.; HOI, S. C. Malicious url detection using machine learning: a survey. *arXiv preprint arXiv:1701.07179*, 2017.

- SAHS, J.; KHAN, L. A machine learning approach to android malware detection. In: IEEE. *2012 European Intelligence and Security Informatics Conference*. Dallas, Texas, 2012. p. 141–147.
- SANTOS, R. J. *Vetores e geometria analítica*. Belo Horizonte: Belo Horizonte: Imprensa Universitária da UFMG, 2010.
- SHANNON, C. E. A mathematical theory of communication. *Bell system technical journal*, Wiley Online Library, v. 27, n. 3, p. 379–423, 1948.
- SHARMA, A.; KUMAWAT, N. K. *URL CLASSIFICATION SYSTEM*. 2016. Disponível em: <https://github.com/Anmol-Sharma/URL_CLASSIFICATION_SYSTEM>. Acesso em: 05 maio 2019.
- SIKORSKI, M.; HONIG, A. *Practical malware analysis: the hands-on guide to dissecting malicious software*. San Francisco: no starch press, 2012.
- SINCLAIR, C.; PIERCE, L.; MATZNER, S. An application of machine learning to network intrusion detection. In: IEEE. *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*. Austin, 1999. p. 371–377.
- SKOUDIS, E.; ZELTSER, L. *Malware: Fighting malicious code*. New Jersey: Prentice Hall Professional, 2004.
- SMITH, A. Smartphone ownership–2013 update. *Pew Research Center: Washington DC*, v. 12, p. 2013, 2013.
- SMOLA, A. J. et al. *Advances in large margin classifiers*. London: MIT press, 2000.
- STALLINGS, W.; BROWN, L. *Segurança de Computadores*. 2. ed. Rio de Janeiro: Elsevier, 2014.
- STEINBERG, D.; COLLA, P. Cart: classification and regression trees. *The top ten algorithms in data mining*, CRC Press Boca Raton, FL, v. 9, p. 179, 2009.
- SUTTON, C. D. Classification and regression trees, bagging, and boosting. *Handbook of statistics*, Elsevier, v. 24, p. 303–329, 2005.
- Tim Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. 1. ed. San Francisco: HarperBusiness, 2000.
- TIMOFEEV, R. Classification and regression trees (cart) theory and applications. *Humboldt University, Berlin*, 2004.
- URCUQUI, C. et al. Machine learning classifiers to detect malicious websites. In: CHILE SPRING SCHOOL OF NETWORKS. *CEUR Workshop Proceedings*. Pucón, 2017. v. 1950, p. 14–17.
- VIGNA, G. *From Anubis and Wepawet to Llama*. 2014. Disponível em: <<https://www.lastline.com/blog/from-anubis-and-wepawet-to-llama/>>. Acesso em: 08 jun. 2019.
- WANG, G. et al. Detecting malicious landing pages in malware distribution networks. In: IEEE. *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Redmond, 2013. p. 1–11.

WANG, Y.-M. et al. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In: CITESEER. *IN NDSS*. Redmond, 2006.

WEIDMAN, G. *Testes de invasão: Uma introdução prática ao hacking*. São Paulo: Novatec Editora, 2014.

WILLIAMS, R. *Google security expert: Facebook and others with poor security are 'tremendously disappointing'*. 2019. Disponível em: <<https://inews.co.uk/news/technology/google-security-expert-facebook-and-others-with-poor-security-are-tremendously-disappointing/>>. Acesso em: 09 jun. 2019.

WU, X. et al. Top 10 algorithms in data mining. *Knowledge and information systems*, Springer, v. 14, n. 1, p. 1–37, 2008.

A Tipos de *malware*

Os *malwares* podem se manifestar em diversas formas, dado o intuito de seu utilizador e o tipo de ameaça que carregam. A seguir, se apresentam algumas definições de categorizações comuns destes softwares. Para a obtenção de exemplos reais e executáveis de *malware*, é recomendável referir-se ao projeto “theZoo”, disponível em: <<https://github.com/ytisf/theZoo>>. Efetivamente, este trabalho não se responsabiliza por eventuais danos causados pelo mal uso de tais exemplos.

A.1 Vírus

Vírus podem ser definidos como trechos de código malicioso executável que afetam arquivos no hospedeiro. Comumente, os vírus requerem interação do usuário para serem executados. (SKOUDIS; ZELTSER, 2004). Os vírus são possivelmente à forma mais popularmente conhecida de manifestação de *malware*. Uma vez infectando a máquina de um usuário, podem infectar não somente arquivos, mas também programas em execução.

A.2 Worm

Worms tratam-se de código malicioso, auto-replicante, geralmente correlato à alguma infraestrutura em rede (SKOUDIS; ZELTSER, 2004). Diferentemente de um vírus, um *worm* pode executar por si próprio, porém, geralmente necessita de uma aplicação que o faça.

A.3 Trojan

Malwares que se escondem dentro de algum outro programa dito como legítimo, de execução normal, são denominados Trojan (WEIDMAN, 2014). Quando ocorre a execução do programa pretendido, o código malicioso é por sua vez executado, causando os mais diversos tipos de dano. Este tipo de ameaça comumente se apresenta em *websites*, através de programas oferecidos pelos mesmos para download.

A.4 Spyware

Spywares são um tipo de ameaça mais sutil, de forma que não afetam diretamente o comportamento da máquina do usuário, mas sim, coletam informações pessoais acerca do mesmo ou de seu comportamento (EGELE et al., 2007). Além de monitorar suas vítimas, um *spyware* também pode, em alguns casos, vir a ser utilizado para a instalação de outros tipos de software malicioso. Também são muito comuns páginas da web que

contém este tipo de ameaça, de forma direta (do próprio código da página) ou não (oferta de downloads).

A.5 *Ransomware*

Ransomware é um termo utilizado para se referir à uma classe de *malware* que é empregada em vias de extorquir suas vítimas, tirando proveito para isto da obtenção do controle total da máquina em questão (LISKA; GALLO, 2016). Não são uma forma de ataque dita nova, pois o primeiro registro que se tem de um ataque utilizando este tipo de *malware* data do final da década de 1980. Em tempos mais atuais, este tipo de ameaça ganhou os meios midiáticos devido à uma série de ataques envolvendo a mesma, com alvo em servidores de grande porte. Comumente, os meios de estabelecimento de um *ransomware* no sistema são correlatos aos de vírus, *worms* ou trojans.

A.6 Outros

Existem muitos outros tipos de ameaça na forma de softwares maliciosos que podem afetar usuários das mais diversas formas. Os *adwares*, por exemplo, são uma subcategoria especial de *spyware* que direciona mensagens indesejadas de propaganda aos usuários afetados. Também existem outros tipos de software que não são *malwares* na definição formal, porém, são comumente utilizados para prejudicar usuários. Um exemplo clássico deste tipo são os *keyloggers*, programas que registram todo o *input* em uma máquina.

B Definições Matemáticas

Este apêndice trata de definir de forma mais específica, conceitos apresentados nos capítulos anteriores embasados na área da matemática. Tais definições podem vir a ser úteis aos leitores que não apresentem afinidade com a área, permitindo-lhes uma melhor compreensão do que foi efetuado.

B.1 Detalhamento sobre vetores

Uma notação comum para vetores é através de um conjunto contendo coordenadas n -dimensionais dispostas em uma tupla de valores reais. Tais coordenadas referem-se ao ponto final deste vetor, partindo de uma origem predefinida (KOWALCZYK, 2017). Tipograficamente, um vetor é comumente representado por uma letra maiúscula ou em negrito.

Para os assuntos que tangem este trabalho, interessam também as definições acerca de módulo e direção de um vetor. O módulo de um vetor, em algumas literaturas também definido como norma, trata-se do tamanho que o segmento representa. Para se calcular o módulo, utiliza-se a fórmula conhecida como norma euclidiana, baseada na geometria euclidiana e na noção de um triângulo retângulo. Considerando um vetor hipotético \mathbf{x} , seu módulo pode ser calculado fazendo como segue.

$$|\mathbf{x}| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

Já a direção, pode ser definida como um novo vetor para o qual as coordenadas serão as razões entre as coordenadas do vetor original sobre seu módulo (KOWALCZYK, 2017). A direção \mathbf{d} será definida para um vetor \mathbf{x} , da seguinte maneira.

$$\mathbf{d} = \left(\frac{x_1}{|\mathbf{x}|}, \frac{x_2}{|\mathbf{x}|}, \dots, \frac{x_n}{|\mathbf{x}|} \right)$$

Um outro formalismo a ser definido sobre vetores se trata do produto escalar. O mesmo consiste em uma operação aplicável aos 2 vetores que componham um ângulo entre si. Algebricamente, o produto escalar pode ser originado como segue, utilizando da identidade do cosseno. Considere um caso geral, como o dos vetores u e v da Figura 18 a seguir.

Considerando a relação entre os cossenos da figura, é possível fazer como segue.

$$\cos(\theta) = \cos(\alpha - \beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)$$

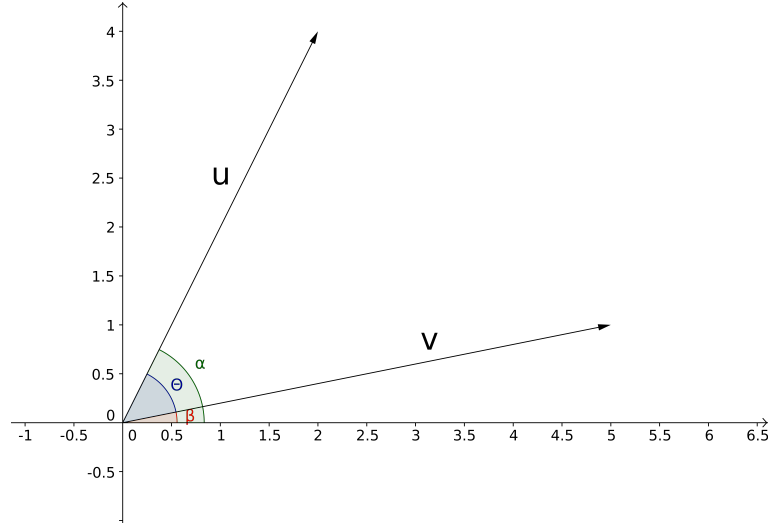


Figura 18 – Vetores e ângulos para definição do produto escalar

$$\cos(\theta) = \frac{u_1}{|\mathbf{u}|} \frac{v_1}{|\mathbf{v}|} + \frac{u_2}{|\mathbf{u}|} \frac{v_2}{|\mathbf{v}|}$$

$$|\mathbf{u}||\mathbf{v}| \cos(\theta) = u_1 v_1 + u_2 v_2$$

Por definição, teremos o produto escalar descrito diretamente com a simbologia que se apresenta a seguir.

$$\mathbf{u} \cdot \mathbf{v} = |\mathbf{u}||\mathbf{v}| \cos(\theta)$$

B.2 Detalhamento sobre Multiplicadores de Lagrange

Os multiplicadores de Lagrange são aplicáveis a uma função objetivo multivariável, que esteja sujeita a uma restrição composta de uma outra função multivariável, igual a um fator constante. Através deste método, o problema de otimização com restrição é convertido em um problema sem restrição, através da inserção do elemento conhecido como multiplicador de Lagrange (MARCHAND, 2016).

Considera-se uma função definida como $f(x, y, \dots)$ restrita a uma condição $g(x, y, \dots) = k$, com k uma constante. Considerando um multiplicador de Lagrange definido como λ , a relação a seguir é tomada.

$$L(x, y, \dots, \lambda) = f(x, y, \dots) - \lambda(g(x, y, \dots) - k)$$

O gradiente da função L , quando igualado a um vetor nulo (zero), permite a construção de um sistema de equações resolvível.

$$\nabla L(x, y, \dots, \lambda) = \mathbf{0}$$

$$\left\{ \begin{array}{l} \frac{\partial L}{\partial x} = 0 \\ \frac{\partial L}{\partial y} = 0 \\ \dots \\ \frac{\partial L}{\partial \lambda} = 0 = k \end{array} \right.$$

Dadas as soluções obtidas através do sistema, conforme é exposto por Marchand (2016, p. 25), onde é considerado um caso com três variáveis: “Os pontos (x, y, z) , onde f tem seus extremos sujeitos restrição, estão entre essas soluções”.

C Exemplo de extração de URL

Este apêndice inclui o resultado da extração de *features* de uma única URL. A Tabela 23 demonstra o mesmo, com suas 19 *features*.

A URL considerada para esta extração foi <http://www.example.com/index.html>.

Tabela 23 – Exemplo de resultado de extração

Feature	Valor extraído	Tipo
Tamanho do nome de domínio	15	Numérica
Tamanho do nome do arquivo	11	Numérica
Presença de IP	False	Booleana
Número de porta	False	Booleana
Presença de sub-domínio	True	Booleana
URL relativa	False	Booleana
TLD	False	Booleana
Tamanho do caminho da URL	11	Numérica
Nome de arquivo suspeito	False	Booleana
Divisões no domínio	2	Numérica
Palavras suspeitas no hostname	False	Booleana
Quantidade de números no hostname	0	Numérica
Código do país	US	Categórica
Região	Massachusetts	Categórica
Zona Temporal	America/New_York	Categórica
A Records	1	Numérica
MX Records	1	Numérica
NS Records	2	Numérica
PTR Equals A	False	Booleana

D Árvore de decisão gerada pelo J48 para 3000 exemplos

J48 pruned tree

Subdomain Presence <= 0

| URL Path Length <= 1

| | A Records <= 0

| | | File Name Length <= 0: 0 (5.0)

| | | File Name Length > 0: 1 (50.0/5.0)

| | A Records > 0: 1 (808.0/7.0)

| URL Path Length > 1

| | File Name Length <= 52

| | | Timezone = America/Chicago

| | | | TLD <= 0

| | | | | NS Records <= 3

| | | | | Domain Name Length <= 16: 0 (5.0)

| | | | | Domain Name Length > 16: 1 (2.0)

| | | | | NS Records > 3: 1 (2.0)

| | | | TLD > 0

| | | | | Domain Name Length <= 26: 1 (102.0/8.0)

| | | | | Domain Name Length > 26: 0 (4.0/1.0)

| | | Timezone = null

| | | | A Records <= 1

| | | | | Domain Name Length <= 22

| | | | | File Name Length <= 7: 0 (5.0/1.0)

| | | | | File Name Length > 7

| | | | | Domain Name Length <= 17: 1 (3.0)

| | | | | Domain Name Length > 17: 0 (5.0/1.0)

| | | | | Domain Name Length > 22: 0 (15.0)

| | | | A Records > 1: 1 (2.0)

| | | Timezone = Asia/Kuching: 1 (0.0)

| | | Timezone = America/Los_Angeles

| | | | NS Records <= 2

| | | | | MX Records <= 2: 1 (6.0/1.0)

| | | | | MX Records > 2: 0 (4.0)

			NS Records > 2: 1 (21.0)
			Timezone = Europe/Warsaw: 1 (0.0)
			Timezone = America/Denver
			Region = null: 0 (0.0)
			Region = California: 0 (0.0)
			Region = Utah: 0 (12.0/1.0)
			Region = Arizona: 0 (0.0)
			Region = Florida: 0 (0.0)
			Region = NewJersey: 0 (0.0)
			Region = Ohio: 0 (0.0)
			Region = England: 0 (0.0)
			Region = Texas: 0 (0.0)
			Region = Nevada: 0 (0.0)
			Region = Illinois: 0 (0.0)
			Region = Oklahoma: 0 (0.0)
			Region = NewSouthWales: 0 (0.0)
			Region = Karnataka: 0 (0.0)
			Region = Vorarlberg: 0 (0.0)
			Region = Hesse: 0 (0.0)
			Region = Ontario: 0 (0.0)
			Region = NorthRhineWestphalia: 0 (0.0)
			Region = ProvinciadeLosSantos: 0 (0.0)
			Region = NorthHolland: 0 (0.0)
			Region = Tuscany: 0 (0.0)
			Region = SaoPaulo: 0 (0.0)
			Region = Madrid: 0 (0.0)
			Region = NewYork: 0 (0.0)
			Region = Incheon: 0 (0.0)
			Region = Michigan: 0 (0.0)
			Region = Leinster: 0 (0.0)
			Region = KalakiTbilisi: 0 (0.0)
			Region = Galicia: 0 (0.0)
			Region = BuenosAires: 0 (0.0)
			Region = Georgia: 0 (0.0)
			Region = Moscow: 0 (0.0)
			Region = Virginia: 0 (0.0)
			Region = Iowa: 0 (0.0)
			Region = SouthHolland: 0 (0.0)
			Region = Quebec: 0 (0.0)

				Region = KyivCity: 0 (0.0)
				Region = HautsdeFrance: 0 (0.0)
				Region = Arusha: 0 (0.0)
				Region = TinhQuangNam: 0 (0.0)
				Region = Bavaria: 0 (0.0)
				Region = Washington: 0 (0.0)
				Region = saka: 0 (0.0)
				Region = AuvergneRhoneAlpes: 0 (0.0)
				Region = SuhbaatarAymag: 0 (0.0)
				Region = NorthCarolina: 0 (0.0)
				Region = SantaFe: 0 (0.0)
				Region = Rajasthan: 0 (0.0)
				Region = Bangkok: 0 (0.0)
				Region = Kansas: 0 (0.0)
				Region = MykolayivskaOblast: 0 (0.0)
				Region = CanaryIslands: 0 (0.0)
				Region = MatoGrossodoSul: 0 (0.0)
				Region = Oregon: 0 (0.0)
				Region = Maharashtra: 0 (0.0)
				Region = ledeFrance: 0 (0.0)
				Region = Pennsylvania: 0 (0.0)
				Region = Massachusetts: 0 (0.0)
				Region = RiodeJaneiro: 0 (0.0)
				Region = Maryland: 0 (0.0)
				Region = TelAviv: 0 (0.0)
				Region = BadenWrttembergRegion: 0 (0.0)
				Region = Maine: 0 (0.0)
				Region = WesternCape: 0 (0.0)
				Region = Saxony: 0 (0.0)
				Region = Nebraska: 0 (0.0)
				Region = Zealand: 0 (0.0)
				Region = Saitama: 0 (0.0)
				Region = Uusimaa: 0 (0.0)
				Region = LandBerlin: 0 (0.0)
				Region = Pardubickykraj: 0 (0.0)
				Region = Missouri: 0 (0.0)
				Region = SaintGallen: 0 (0.0)
				Region = MinasGerais: 0 (0.0)
				Region = Colorado: 1 (2.0)

				Region = TaipeiCity: 0 (0.0)
				Region = Fujian: 0 (0.0)
				Region = BritishColumbia: 0 (0.0)
				Region = Parana: 0 (0.0)
				Region = Beijing: 0 (0.0)
				Region = Haryana: 0 (0.0)
				Region = Zhejiang: 0 (0.0)
				Region = BaselCity: 0 (0.0)
				Region = NewProvidenceDistrict: 0 (0.0)
				Region = NewHampshire: 0 (0.0)
				Region = Minnesota: 0 (0.0)
				Region = Flanders: 0 (0.0)
				Region = SaiKungDistrict: 0 (0.0)
				Region = CentralandWesternDistrict: 0 (0.0)
				Region = Tokyo: 0 (0.0)
				Region = Seoul: 0 (0.0)
				Region = TsuenWanDistrict: 0 (0.0)
				Region = Scotland: 0 (0.0)
				Region = Istanbul: 0 (0.0)
				Region = Yerevan: 0 (0.0)
				Region = ProvincieFlevoland: 0 (0.0)
				Region = Victoria: 0 (0.0)
				Region = NovaScotia: 0 (0.0)
				Region = Saarland: 0 (0.0)
				Region = RheinlandPfalz: 0 (0.0)
				Region = Waikato: 0 (0.0)
				Region = Zurich: 0 (0.0)
				Region = NewMexico: 1 (1.0)
				Region = Wallonia: 0 (0.0)
				Region = Wyoming: 0 (0.0)
				Region = Lisbon: 0 (0.0)
				Region = Jiangsu: 0 (0.0)
				Region = Saskatchewan: 0 (0.0)
				Region = Auckland: 0 (0.0)
				Region = Jerusalem: 0 (0.0)
				Region = OsloCounty: 0 (0.0)
				Region = ProvinciadeCartago: 0 (0.0)
				Region = Wisconsin: 0 (0.0)
				Region = Indiana: 0 (0.0)

				Region = ProvincieUtrecht: 0 (0.0)
				Region = SouthDenmark: 0 (0.0)
				Region = Wales: 0 (0.0)
				Region = StPetersburg: 0 (0.0)
				Region = Liguria: 0 (0.0)
				Region = Tennessee: 0 (0.0)
				Region = RhodeIsland: 0 (0.0)
				Region = CentralJutland: 0 (0.0)
				Region = Apulia: 0 (0.0)
				Region = Piedmont: 0 (0.0)
				Region = CapitalRegion: 0 (0.0)
				Timezone = Europe/Amsterdam
				File Name Length <= 7: 0 (2.0)
				File Name Length > 7: 1 (4.0)
				Timezone = America/Phoenix
				URL Path Length <= 11: 0 (4.0)
				URL Path Length > 11: 1 (11.0/1.0)
				Timezone = Asia/Hong_Kong: 1 (1.0)
				Timezone = America/New_York
				TLD <= 0: 0 (4.0/1.0)
				TLD > 0: 1 (58.0/2.0)
				Timezone = Asia/Jakarta: 1 (0.0)
				Timezone = Europe/Vilnius: 1 (0.0)
				Timezone = Europe/London: 1 (25.0)
				Timezone = Asia/Taipei: 1 (0.0)
				Timezone = Asia/Singapore: 1 (0.0)
				Timezone = Australia/Sydney: 0 (1.0)
				Timezone = Europe/Berlin
				NS Records <= 2
				Region = null
				Domain Name Length <= 29
				File Name Length <= 20: 1 (4.0)
				File Name Length > 20: 0 (3.0/1.0)
				Domain Name Length > 29: 0 (2.0)
				Region = California: 1 (0.0)
				Region = Utah: 1 (0.0)
				Region = Arizona: 1 (0.0)
				Region = Florida: 1 (0.0)
				Region = NewJersey: 1 (0.0)

					Region = Ohio: 1 (0.0)
					Region = England: 1 (0.0)
					Region = Texas: 1 (0.0)
					Region = Nevada: 1 (0.0)
					Region = Illinois: 1 (0.0)
					Region = Oklahoma: 1 (0.0)
					Region = NewSouthWales: 1 (0.0)
					Region = Karnataka: 1 (0.0)
					Region = Vorarlberg: 1 (0.0)
					Region = Hesse: 1 (0.0)
					Region = Ontario: 1 (0.0)
					Region = NorthRhineWestphalia: 0 (3.0)
					Region = ProvinciadeLosSantos: 1 (0.0)
					Region = NorthHolland: 1 (0.0)
					Region = Tuscany: 1 (0.0)
					Region = SaoPaulo: 1 (0.0)
					Region = Madrid: 1 (0.0)
					Region = NewYork: 1 (0.0)
					Region = Incheon: 1 (0.0)
					Region = Michigan: 1 (0.0)
					Region = Leinster: 1 (0.0)
					Region = KalakiTbilisi: 1 (0.0)
					Region = Galicia: 1 (0.0)
					Region = BuenosAires: 1 (0.0)
					Region = Georgia: 1 (0.0)
					Region = Moscow: 1 (0.0)
					Region = Virginia: 1 (0.0)
					Region = Iowa: 1 (0.0)
					Region = SouthHolland: 1 (0.0)
					Region = Quebec: 1 (0.0)
					Region = KyivCity: 1 (0.0)
					Region = HautsdeFrance: 1 (0.0)
					Region = Arusha: 1 (0.0)
					Region = TinhQuangNam: 1 (0.0)
					Region = Bavaria: 1 (0.0)
					Region = Washington: 1 (0.0)
					Region = saka: 1 (0.0)
					Region = AuvergneRhôneAlpes: 1 (0.0)
					Region = SuhbaatarAymag: 1 (0.0)

					Region = NorthCarolina: 1 (0.0)
					Region = SantaFe: 1 (0.0)
					Region = Rajasthan: 1 (0.0)
					Region = Bangkok: 1 (0.0)
					Region = Kansas: 1 (0.0)
					Region = MykolayivskaOblast: 1 (0.0)
					Region = CanaryIslands: 1 (0.0)
					Region = MatoGrossodoSul: 1 (0.0)
					Region = Oregon: 1 (0.0)
					Region = Maharashtra: 1 (0.0)
					Region = ledeFrance: 1 (0.0)
					Region = Pennsylvania: 1 (0.0)
					Region = Massachusetts: 1 (0.0)
					Region = RiodeJaneiro: 1 (0.0)
					Region = Maryland: 1 (0.0)
					Region = TelAviv: 1 (0.0)
					Region = BadenWrttembergRegion: 1 (0.0)
					Region = Maine: 1 (0.0)
					Region = WesternCape: 1 (0.0)
					Region = Saxony: 1 (0.0)
					Region = Nebraska: 1 (0.0)
					Region = Zealand: 1 (0.0)
					Region = Saitama: 1 (0.0)
					Region = Uusimaa: 1 (0.0)
					Region = LandBerlin: 1 (4.0)
					Region = Pardubickykraj: 1 (0.0)
					Region = Missouri: 1 (0.0)
					Region = SaintGallen: 1 (0.0)
					Region = MinasGerais: 1 (0.0)
					Region = Colorado: 1 (0.0)
					Region = TaipeiCity: 1 (0.0)
					Region = Fujian: 1 (0.0)
					Region = BritishColumbia: 1 (0.0)
					Region = Parana: 1 (0.0)
					Region = Beijing: 1 (0.0)
					Region = Haryana: 1 (0.0)
					Region = Zhejiang: 1 (0.0)
					Region = BaselCity: 1 (0.0)
					Region = NewProvidenceDistrict: 1 (0.0)

					Region = NewHampshire: 1 (0.0)
					Region = Minnesota: 1 (0.0)
					Region = Flanders: 1 (0.0)
					Region = SaiKungDistrict: 1 (0.0)
					Region = CentralandWesternDistrict: 1 (0.0)
					Region = Tokyo: 1 (0.0)
					Region = Seoul: 1 (0.0)
					Region = TsuenWanDistrict: 1 (0.0)
					Region = Scotland: 1 (0.0)
					Region = Istanbul: 1 (0.0)
					Region = Yerevan: 1 (0.0)
					Region = ProvincieFlevoland: 1 (0.0)
					Region = Victoria: 1 (0.0)
					Region = NovaScotia: 1 (0.0)
					Region = Saarland: 1 (0.0)
					Region = RheinlandPfalz: 1 (0.0)
					Region = Waikato: 1 (0.0)
					Region = Zurich: 1 (0.0)
					Region = NewMexico: 1 (0.0)
					Region = Wallonia: 1 (0.0)
					Region = Wyoming: 1 (0.0)
					Region = Lisbon: 1 (0.0)
					Region = Jiangsu: 1 (0.0)
					Region = Saskatchewan: 1 (0.0)
					Region = Auckland: 1 (0.0)
					Region = Jerusalem: 1 (0.0)
					Region = OsloCounty: 1 (0.0)
					Region = ProvinciadeCartago: 1 (0.0)
					Region = Wisconsin: 1 (0.0)
					Region = Indiana: 1 (0.0)
					Region = ProvincieUtrecht: 1 (0.0)
					Region = SouthDenmark: 1 (0.0)
					Region = Wales: 1 (0.0)
					Region = StPetersburg: 1 (0.0)
					Region = Liguria: 1 (0.0)
					Region = Tennessee: 1 (0.0)
					Region = RhodeIsland: 1 (0.0)
					Region = CentralJutland: 1 (0.0)
					Region = Apulia: 1 (0.0)

```

|   |   |   |   |   Region = Piedmont: 1 (0.0)
|   |   |   |   |   Region = CapitalRegion: 1 (0.0)
|   |   |   |   NS Records > 2: 1 (11.0)
|   |   |   Timezone = Europe/Paris
|   |   |   |   File Name Length <= 31: 1 (4.0)
|   |   |   |   File Name Length > 31: 0 (3.0)
|   |   |   Timezone = Europe/Istanbul: 0 (1.0)
|   |   |   Timezone = Europe/Moscow
|   |   |   |   Domain Divisions <= 2: 0 (6.0)
|   |   |   |   Domain Divisions > 2: 1 (3.0)
|   |   |   Timezone = Atlantic/Cape_Verde: 0 (4.0)
|   |   |   Timezone = Asia/Kolkata: 0 (6.0)
|   |   |   Timezone = Asia/Seoul: 1 (0.0)
|   |   |   Timezone = Europe/Budapest: 1 (0.0)
|   |   |   Timezone = Asia/Dhaka: 1 (0.0)
|   |   |   Timezone = Europe/Vienna: 1 (0.0)
|   |   |   Timezone = America/Toronto: 1 (13.0/2.0)
|   |   |   Timezone = America/Santiago: 1 (0.0)
|   |   |   Timezone = Asia/Tehran: 1 (0.0)
|   |   |   Timezone = America/Panama: 1 (0.0)
|   |   |   Timezone = Europe/Madrid: 0 (1.0)
|   |   |   Timezone = Europe/Rome
|   |   |   |   NS Records <= 3: 1 (5.0)
|   |   |   |   NS Records > 3: 0 (5.0/1.0)
|   |   |   Timezone = America/Sao_Paulo
|   |   |   |   TLD <= 0: 0 (2.0)
|   |   |   |   TLD > 0: 1 (10.0)
|   |   |   Timezone = Asia/Bangkok: 0 (2.0)
|   |   |   Timezone = America/Detroit: 1 (8.0)
|   |   |   Timezone = Europe/Dublin: 1 (4.0)
|   |   |   Timezone = Asia/Tbilisi: 0 (3.0)
|   |   |   Timezone = America/Argentina/Buenos_Aires: 1 (0.0)
|   |   |   Timezone = Indian/Mahe: 1 (0.0)
|   |   |   Timezone = Europe/Helsinki: 1 (0.0)
|   |   |   Timezone = Europe/Stockholm: 1 (0.0)
|   |   |   Timezone = Europe/Kiev: 1 (0.0)
|   |   |   Timezone = Europe/Riga: 1 (1.0)
|   |   |   Timezone = Europe/Sofia: 1 (1.0)
|   |   |   Timezone = Asia/Tokyo: 1 (2.0)

```

		Timezone = Asia/Aden: 1 (0.0)
		Timezone = Africa/Dar_es_Salaam: 1 (0.0)
		Timezone = Asia/Ho_Chi_Minh: 0 (2.0)
		Timezone = Asia/Choibalsan: 1 (0.0)
		Timezone = Asia/Almaty: 1 (0.0)
		Timezone = Africa/Lagos: 1 (0.0)
		Timezone = Europe/Minsk: 1 (0.0)
		Timezone = America/Argentina/Cordoba: 1 (0.0)
		Timezone = Africa/Johannesburg: 1 (0.0)
		Timezone = Europe/Bucharest: 1 (0.0)
		Timezone = Atlantic/Canary: 1 (0.0)
		Timezone = America/Campo_Grande: 0 (5.0)
		Timezone = America/Belize: 1 (0.0)
		Timezone = America/Montevideo: 1 (0.0)
		Timezone = Asia/Jerusalem: 1 (3.0)
		Timezone = Asia/Shanghai: 1 (0.0)
		Timezone = Europe/Lisbon: 1 (0.0)
		Timezone = Europe/Vaduz: 1 (7.0)
		Timezone = Europe/Copenhagen: 1 (0.0)
		Timezone = Europe/Prague: 1 (1.0)
		Timezone = Europe/Zurich: 1 (2.0)
		Timezone = America/Vancouver: 1 (5.0)
		Timezone = America/Cayman: 1 (0.0)
		Timezone = America/Nassau: 1 (0.0)
		Timezone = Europe/Brussels: 1 (1.0)
		Timezone = Australia/Perth: 1 (0.0)
		Timezone = Asia/Yerevan: 1 (0.0)
		Timezone = Australia/Melbourne: 1 (1.0)
		Timezone = America/Halifax: 1 (1.0)
		Timezone = Pacific/Auckland: 1 (1.0)
		Timezone = America/Regina: 1 (0.0)
		Timezone = Europe/Oslo: 1 (0.0)
		Timezone = America/Costa_Rica: 1 (0.0)
		Timezone = America/Indiana/Indianapolis: 1 (0.0)
		Timezone = America/Tortola: 1 (1.0)
		File Name Length > 52
		MX Records <= 2: 0 (81.0/1.0)
		MX Records > 2
		Domain Name Length <= 17: 1 (3.0)

				Domain Name Length > 17: 0 (4.0)							
Subdomain Presence > 0											
		File Name Length <= 41									
			MX Records <= 0: 0 (496.0/20.0)								
			MX Records > 0								
				NS Records <= 2							
					File Name Length <= 4						
						File Name Length <= 0: 0 (7.0)					
						File Name Length > 0					
						Domain Divisions <= 2					
							Suspicious Words <= 0: 0 (6.0/1.0)				
							Suspicious Words > 0: 1 (72.0/10.0)				
						Domain Divisions > 2: 0 (9.0/1.0)					
					File Name Length > 4: 0 (270.0/32.0)						
				NS Records > 2							
					Numbers In Host <= 1						
					Timezone = America/Chicago						
						TLD <= 0: 0 (5.0/1.0)					
						TLD > 0					
						MX Records <= 1					
							File Name Length <= 20: 1 (13.0)				
							File Name Length > 20				
								Domain Divisions <= 1: 0 (2.0)			
								Domain Divisions > 1			
									Domain Name Length <= 20: 1 (4.0)		
									Domain Name Length > 20: 0 (2.0)		
							MX Records > 1: 1 (45.0)				
					Timezone = null: 1 (7.0/1.0)						
					Timezone = Asia/Kuching: 1 (0.0)						
					Timezone = America/Los_Angeles						
						TLD <= 0: 0 (3.0)					
						TLD > 0					
							A Records <= 0: 0 (3.0)				
							A Records > 0				
							Domain Name Length <= 12: 0 (3.0/1.0)				
							Domain Name Length > 12: 1 (30.0/1.0)				
					Timezone = Europe/Warsaw: 1 (0.0)						
					Timezone = America/Denver: 0 (4.0)						
					Timezone = Europe/Amsterdam						

						Domain Divisions <= 1: 0 (2.0)
						Domain Divisions > 1: 1 (6.0/1.0)
						Timezone = America/Phoenix: 0 (5.0/1.0)
						Timezone = Asia/Hong_Kong: 1 (0.0)
						Timezone = America/New_York: 1 (34.0/1.0)
						Timezone = Asia/Jakarta: 0 (1.0)
						Timezone = Europe/Vilnius: 1 (0.0)
						Timezone = Europe/London: 1 (15.0)
						Timezone = Asia/Taipei: 1 (0.0)
						Timezone = Asia/Singapore: 0 (1.0)
						Timezone = Australia/Sydney: 1 (1.0)
						Timezone = Europe/Berlin
						A Records <= 0: 1 (3.0)
						A Records > 0
						File Name Length <= 2: 1 (4.0)
						File Name Length > 2: 0 (15.0/1.0)
						Timezone = Europe/Paris
						Domain Divisions <= 1: 0 (3.0)
						Domain Divisions > 1: 1 (4.0)
						Timezone = Europe/Istanbul: 0 (3.0/1.0)
						Timezone = Europe/Moscow
						Domain Divisions <= 1: 0 (4.0)
						Domain Divisions > 1: 1 (3.0)
						Timezone = Atlantic/Cape_Verde: 1 (0.0)
						Timezone = Asia/Kolkata: 0 (2.0/1.0)
						Timezone = Asia/Seoul: 0 (1.0)
						Timezone = Europe/Budapest: 1 (0.0)
						Timezone = Asia/Dhaka: 1 (0.0)
						Timezone = Europe/Vienna: 1 (0.0)
						Timezone = America/Toronto: 1 (5.0)
						Timezone = America/Santiago: 1 (0.0)
						Timezone = Asia/Tehran: 0 (1.0)
						Timezone = America/Panama: 1 (0.0)
						Timezone = Europe/Madrid: 1 (0.0)
						Timezone = Europe/Rome: 1 (4.0/1.0)
						Timezone = America/Sao_Paulo
						A Records <= 2: 0 (2.0)
						A Records > 2: 1 (11.0)
						Timezone = Asia/Bangkok: 1 (0.0)

					Timezone = America/Detroit: 0 (1.0)
					Timezone = Europe/Dublin: 1 (3.0/1.0)
					Timezone = Asia/Tbilisi: 1 (0.0)
					Timezone = America/Argentina/Buenos_Aires: 0 (1.0)
					Timezone = Indian/Mahe: 1 (0.0)
					Timezone = Europe/Helsinki: 1 (0.0)
					Timezone = Europe/Stockholm: 1 (0.0)
					Timezone = Europe/Kiev: 1 (0.0)
					Timezone = Europe/Riga: 1 (0.0)
					Timezone = Europe/Sofia: 1 (1.0)
					Timezone = Asia/Tokyo: 1 (2.0)
					Timezone = Asia/Aden: 1 (0.0)
					Timezone = Africa/Dar_es_Salaam: 1 (0.0)
					Timezone = Asia/Ho_Chi_Minh: 1 (0.0)
					Timezone = Asia/Choibalsan: 1 (0.0)
					Timezone = Asia/Almaty: 0 (1.0)
					Timezone = Africa/Lagos: 1 (0.0)
					Timezone = Europe/Minsk: 1 (0.0)
					Timezone = America/Argentina/Cordoba: 1 (0.0)
					Timezone = Africa/Johannesburg: 1 (1.0)
					Timezone = Europe/Bucharest: 0 (1.0)
					Timezone = Atlantic/Canary: 1 (0.0)
					Timezone = America/Campo_Grande: 1 (0.0)
					Timezone = America/Belize: 1 (0.0)
					Timezone = America/Montevideo: 1 (0.0)
					Timezone = Asia/Jerusalem: 1 (2.0)
					Timezone = Asia/Shanghai: 1 (0.0)
					Timezone = Europe/Lisbon: 1 (1.0)
					Timezone = Europe/Vaduz: 1 (0.0)
					Timezone = Europe/Copenhagen: 1 (1.0)
					Timezone = Europe/Prague: 1 (0.0)
					Timezone = Europe/Zurich: 1 (0.0)
					Timezone = America/Vancouver: 1 (0.0)
					Timezone = America/Cayman: 1 (0.0)
					Timezone = America/Nassau: 1 (0.0)
					Timezone = Europe/Brussels: 1 (1.0)
					Timezone = Australia/Perth: 1 (0.0)
					Timezone = Asia/Yerevan: 1 (0.0)
					Timezone = Australia/Melbourne: 1 (0.0)

					Timezone = America/Halifax: 1 (0.0)
					Timezone = Pacific/Auckland: 1 (0.0)
					Timezone = America/Regina: 1 (1.0)
					Timezone = Europe/Oslo: 1 (0.0)
					Timezone = America/Costa_Rica: 1 (0.0)
					Timezone = America/Indiana/Indianapolis: 1 (0.0)
					Timezone = America/Tortola: 1 (0.0)
					Numbers In Host > 1: 0 (29.0/4.0)
					File Name Length > 41: 0 (450.0/2.0)

