TRABALHO DE CONCLUSÃO DE CURSO

UM SISTEMA INTELIGENTE PARA PREVENÇÃO DE ATAQUES PHISHING

Luís Felipe de Faria Rodrigues

Igor Augusto Mageste da Mota Bastos

Brasília, dezembro de 2018

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE CONCLUSÃO DE CURSO

UM SISTEMA INTELIGENTE PARA PREVENÇÃO DE ATAQUES PHISHING

Luís Felipe de Faria Rodrigues

Igor Augusto Mageste da Mota Bastos

Trabalho de Conclusão de Curso submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Alexandre Solon Nery, D.Sc., FT/UnB _____
Orientador

Prof. Georges Daniel Amvame Nze, D.Sc., FT/UnB _____
Examinador interno

Prof. Rafael Timóteo de Sousa Jr., Ph.D., FT/UnB _____
Examinador interno

FICHA CATALOGRÁFICA

RODRIGUES, LUÍS FELIPE DE FARIA
BASTOS, IGOR AUGUSTO MAGESTE DA MOTA
UM SISTEMA INTELIGENTE PARA PREVENÇÃODE ATAQUES PHISHING [Distrito Federal] 2018.
xvi, 70 p., 210 x 297 mm (ENE/FT/UnB, Engenheiro, Engenharia Elétrica, 2018).
Trabalho de Conclusão de Curso - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Phishing	2. WebCrawler
3. Rede neural	4. Multiprocessamento
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

RODRIGUES, L.F.F., BASTOS, I.A.M.M. (2018). *UM SISTEMA INTELIGENTE PARA PREVENÇÃODE ATAQUES PHISHING* . Trabalho de Conclusão de Curso, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 70 p.

CESSÃO DE DIREITOS

AUTOR 1: Igor Augusto Mageste da Mota Bastos
AUTOR 2: Luís Felipe de Faria Rodrigues
TÍTULO: UM SISTEMA INTELIGENTE PARA PREVENÇÃODE ATAQUES PHISHING .
GRAU: Engenheiro de Redes de Comunicação ANO: 2018

Luís Felipe de Faria Rodrigues
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Igor Augusto Mageste da Mota Bastos
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedicatórias

Dedico este trabalho, em primeiro lugar a Deus, que tem sempre me abençoado e me dado saúde para seguir firme durante essa caminhada. Em segundo lugar, agradeço a minha família e namorada que me deram apoio, carinho e que sempre acreditaram em mim e não mediram esforços para que eu chegasse até esta etapa da minha vida.

Igor Augusto Mageste da Mota Bastos

Dedico este trabalho à minha família, por sua capacidade de investir e acreditar em mim. Mãe, sua dedicação e seu cuidado foram que deram, em alguns momentos, força para seguir a caminhada. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

Luís Felipe de Faria Rodrigues

Agradecimentos

Agradeço as meus pais, irmãos, minha namorada e a toda minha família que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida. Agradeço aos meus amigos, pelas alegrias, tristezas e dores compartilhadas. Com vocês, as pausas entre um parágrafo outro melhora tudo que tenho produzido na vida. Agradeço ao professor Alexandre Nery pela paciência na orientação e incentivo que tornaram possível a conclusão deste trabalho. Agradeço à todos responsáveis pelo Curso de Engenharia de Redes de Comunicação da Universidade de Brasília, às pessoas com quem convivi nesse espaço ao longo desses anos. A experiência de uma produção compartilhada na comunhão com amigos nesses espaços foram a melhor experiência da minha formação acadêmica. Por fim agradeço a todos aqueles que de alguma forma estiveram e estão próximos de mim, fazendo esta vida valer cada vez mais a pena.

Luís Felipe de Faria Rodrigues

Agradeço a Deus por me abençoar, capacitar em mais uma etapa da minha vida e ter me dado saúde para superar as dificuldades. Agradeço a minha família, por sempre me amar e apoiar, principalmente nas dificuldades. A minha namorada, pelo carinho e amor e pelo suporte incondicional. Agradeço ao meu orientador, professor Alexandre Nery, por aceitar ser meu orientador nesse TCC e pelo suporte no pouco tempo que lhe coube, pelas suas correções e incentivos.
Agradeço a esta universidade e seu corpo docente, por proporcionar um ambiente de estudo e formação pessoal. Agradeço a todos os professores que se dedicam a ensinar e contribuir com o conhecimento e formação profissional dos alunos. Agradeço aos meus amigos de curso que fizeram parte da minha formação e que vão continuar presentes na minha vida com certeza. E a todos que direta ou indiretamente fizeram parte da minha formação, meu muito obrigado.

Igor Augusto Mageste da Mota Bastos

RESUMO

Métodos de aprendizado de máquina são usados em uma variedade de tarefas computacionais onde criar e programa algoritmos é impraticável. Na área da segurança, porém, seu uso ainda está em pleno desenvolvimento. Este trabalho apresenta uma introdução aos ataques cibernéticos mais comuns, com ênfase no ataque do tipo *phishing*, às redes neurais sem peso e à algoritmos de árvores de decisão para entendimento do desenvolvimento do sistema proposto. O objetivo é propor um sistema inteligente capaz de identificar, por meio de elementos presentes na URL e na página, se um site está contaminado pelo *phishing*. Conforme os resultados, com a escolha adequada dos parâmetros, é possível a criação de um sistema inteligente e proativo capaz de identificar e prevenir ataques do tipo *phishing*, sugerindo o potencial da aplicação do aprendizado de máquinas na área da segurança.

ABSTRACT

Machine learning methods are used in a variety of computational tasks where creating and programming algorithms is impractical. In the area of cyber security, however, its use is still in development. This work presents an introduction to the most common cyber attacks, with emphasis on *phishing* attack, at weightless neural networks and decision tree algorithms for understanding the development of the proposed system. The objective is to propose an intelligent system capable of identifying, through elements presented at the URL and page, if a site is contaminated by phishing. According to the results, with the appropriate choice of parameters, it is possible to create a intelligent and proactive system capable of identifying and preventing attacks of the type phishing, suggesting the potential of applying machine learning in the area of safety.

PALAVRAS CHAVE

Phishing, Rede Neural, WebCrawler e Multiprocessamento.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

1 INTRODUÇÃO

Muito antes do século XXI, já existia a necessidade de troca de informações entre povos de diferentes culturas e línguas, com segurança: sigilo, confidencialidade, confiabilidade, entre outros. Com o passar do tempo, avanços na tecnologia de semicondutores e, consequentemente, na tecnologia de comunicação intensificaram ainda mais a disseminação de informações e a rapidez com que as mesmas são transmitidas. A Internet conectou indivíduos e as instituições, redes de universidades, forças militares, entre outras, que por sua vez passaram a produzir cada vez mais conteúdo. Com essa grande massa de informações "valiosas" depositadas a cada dia no espaço cibernético, os *hackers* têm procurado vulnerabilidades e formas diferentes de aplicar golpes. Conforme o ambiente de ameaças evolui, a proteção contra essas ameaças também deve evoluir. Com o surgimento dos ataques direcionados e ameaças avançadas, uma nova abordagem se faz necessária em relação à segurança cibernética.

1.1 PROBLEMA

Com as facilidades de criação de sites de compras e vendas, operações bancárias, entre outros, a segurança de informações e dados bancários, tem sido uma preocupação constante de desenvolvedores e instituições que fazem uso desses. Aproveitando esse crescimento exponencial e a falta de informações claras sobre a fonte e a segurança dos *websites*, golpes estão sendo cada vez mais usados para conseguir dados e informações pessoais dos usuários. Quem nunca recebeu aquelas mensagens do tipo "atualize seus dados bancários" ou, então, "parabéns, você é o novo milionário"? Esse tipo de correio se tornou rotina em nossas caixas de e-mail, configurando um golpe muito comum na internet: o *phishing*. O termo *phishing* vem da palavra do idioma inglês, *fishing*, cujo significado em português é pescar. O funcionamento desse ataque está relacionado ao calmo esporte: os *hackers* fazem nada mais do que uma "pescaria digital", lançando centenas de "iscas" pela internet para depois esperar por uma "mordida", ou seja, que alguém acesse, por exemplo, um link malicioso. O ato de acessar o link malicioso em muitos casos levará o usuário a um sítio falso, com aspectos semelhantes ao sítio original, para que o usuário não perceba que está caindo em uma armadilha. No sítio falso, o usuário muitas vezes informa dados sensíveis, como o número de contas bancárias e até mesmo senhas, que são facilmente capturados pelo *hacker*.

Os ataques com *phishing* têm se diversificado bastante com o passar dos anos. Antes, a maioria dos golpes eram disseminados por correio eletrônico, mas hoje também chegam por SMS, sítios falsos, anúncios no *Google*, propagandas em redes sociais e, principalmente, via *WhatsApp*. Diferenciar um sítio ou e-mail verdadeiro de um falso é uma tarefa complicada, até mesmo para usuários que possuem conhecimento na área de tecnologia da informação. Por exemplo, dependendo da fonte utilizada, é fácil disfarçar a letra "l" como um "i" maiúsculo, que acabam sendo

representados de forma muito parecida pela barra de navegação dos navegadores e mensagens do WhatsApp, como mostra a Tabela 1.1, que compara diferentes tipos de fontes. Percebe-se que a fonte Helvetica pode facilmente iludir o usuário, redirecionado-o a um sítio malicioso que acredita ser sua conta de correio eletrônico.

Tabela 1.1: Disfarce da letra "l" como um "i" maiúsculo usando a fonte *Helvetica*.

Tipos de fonte Fonte	Padrão LaTeX	Helvetica	Times	Courier
URLs	gmaiI.com	gmail.com	gmaiI.com	gmaiI.com

Além disso, exemplos práticos de como criar ferramentas *phishing* estão cada vez mais comuns, como é o caso do tutorial apresentado em (3), que explica um método de criação de uma página *phishing* do *Gmail*, em apenas cinco passos. Esse método é simples e não requer nenhum conhecimento de programação, ou seja, esse método pode ser aplicado por qualquer pessoa.

A grande vantagem para quem se aproveita do modo de ataque por *phishing* é o custo relativamente barato. Por exemplo, o cadastro de domínio, o disparo de e-mails em massa e a obtenção de certificado digital, necessitam de um investimento muito baixo, ou até inexistente. Por esse motivo, o crescimento desse tipo de prática tem crescido de forma alarmante. Para se ter ideia do crescimento desse ataque, segundo um estudo da Riskified(4), o golpe de *phishing* com lojas virtuais cresceu quase 300% no ano de 2018 e o Brasil é campeão de atividades de fraude no varejo.

> "Golpes de *phishing* em lojas virtuais aumentaram 297% no último ano, segundo estudo realizado pela empresa de prevenção de fraudes Riskified e a companhia de *cyber* inteligência *IntSights Cyber Intelligence*. Nesta estratégica, *cyber* criminosos se passam por serviços de e-commerce para coletar informações dos usuários, como dados bancários. A pesquisa mostra que o Brasil é o campeão de atividades de fraude no varejo, sendo 24,9% mais perigoso para realizar transações online que a média. Em seguida, aparecem México (10,4%) e Índia (1,9%). De acordo com os analistas, esses países são conhecidos por terem empresas com medidas de segurança cibernética mais fracas, o que facilita a ação de criminosos.". Fonte: TECHTUDO (4)

1.2 REVISÃO DA LITERATURA

Esta Seção apresenta o estado da arte sobre detecção e prevenção de ataques de *phishing* para melhor conhecimento sobre o que tem sido estudado sobre este assunto.

Em um trabalho publicado pela *Google* (5) foi mostrado um estudo sobre a estrutura das URLs empregadas em vários ataques de *phishing*, no qual muitas vezes é possível dizer se uma URL pertence ou não a um ataque de *phishing* sem precisar de nenhum conhecimento dos dados da página correspondente. Foram descritos vários recursos que podem ser usados para modelar um filtro de regressão logística que seja eficiente e tenha alta precisão. Para tanto foram usados filtros

para realizar medições completas em milhões de URLs e quantificar a prevalência de *phishing* na Internet hoje.

Nesse mesmo trabalho, para o treino do modelo, criou-se uma *blacklist* e uma *whitelist* de treinamento. A *blacklist* com as URLs maliciosas foi retirada do banco de dados da própria *Google*, que é atualizada continuamente por meios de fontes comerciais e também por fontes internas da empresa. Essa *blacklist* é analisada por especialistas humanos para eliminar qualquer URL benigno. Foram usadas 1245 URLs desta lista para o treinamento. Já para a *whitelist*, usou-se uma lista dos 1000 URLs mais populares como base de treinamento. Essa lista foi coletada por vários meios, incluindo seleção manual e técnicas algorítmicas que agrupam sinais para detectar URLs de alta qualidade. Além disso, foi adicionado URLs com baixa popularidade para melhorar o conjunto de treinamento, somando um total de 1263 URLs benignas. Em conjunto, as listas maliciosas e não maliciosas, fazendo proveito da infraestrutura do *Google*, formam uma tabela de domínios no qual mantém uma *whitelist* de domínios de organizações que são conhecidas como alvos de *phishing*, tendo por exemplo, vários bancos e também o *Ebay*.

Em comparação com este trabalho, usou-se para a criação do modelo uma *blacklist* e uma *whitelist* para treinamento. A *blacklist* com as URLs maliciosas foi retirada de banco de dados online, chamado de *PhishTank*, no qual foi extraído 5426 URLs. Já para *whitelist*, usou-se um mecanismo de busca na web, chamado de *webcrawler*. Esse mecanismo tem como objetivo a captura de informações da página web, além de capturar outros links a partir da página de origem. Com o uso desse mecanismo, foi coletado 6277 links de páginas não maliciosas, para composição da *whitelist* para treinamento.

No artigo publicado pelo *Human-centric Computing and Information Sciences* (6) foi apresentado um enfoque sobre o discernimento dos recursos significativos que discriminam entre URLs legítimas e de URLs com *phishing*. Esses recursos são então submetidos à mineração de regras associativas - *apriori* e *predictive apriori*. As regras obtidas são interpretadas para enfatizar os recursos mais predominantes em URLs de *phishing*. Analisando o conhecimento acessível em URL de *phishing* e considerando a confiança como um indicador, os recursos como segurança da camada de transporte, indisponibilidade do domínio de nível superior na URL e palavra-chave na parte do caminho do URL foram considerados indicadores razoáveis de URL maliciosas. Além desses, o número de barras no URL, o ponto na parte do host do URL e o tamanho do URL também são os principais fatores para o URL de *phishing*.

Em comparação com este trabalho, usou-se para a criação das *features*, várias das regras, para classificar as URLS, citadas em (6). Como por exemplo, o tamanho do domínio, a quantidade de caracteres no domínio, o tamanho do URL, entre outros.

No artigo (7) realizado por alunos de doutorado da *College of Computer Science, Zhejiang University, Hangzhou, China* forneceu uma pesquisa abrangente e um entendimento estrutural das técnicas de Detecção de URL Maliciosa usando aprendizado de máquina. Também categorizou-se e revisou-se as contribuições de estudos da literatura que abordam diferentes dimensões desse problema (representação de recursos, design de algoritmo, etc.). Além disso, nesse mesmo artigo

foi fornecido uma pesquisa oportuna e abrangente para uma variedade de públicos diferentes, não apenas para pesquisadores e engenheiros de aprendizado de máquina na academia, mas também para profissionais da indústria de segurança cibernética, para ajudá-los a entender o estado da arte e facilitar a pesquisa própria e as aplicações práticas. Também foi discutido questões práticas em design de sistemas, desafios de pesquisa aberta e apontou-se algumas orientações importantes para futuras pesquisas.

No artigo publicado pelo *International Journal of Innovative Research in Advanced Engineering* (8) descreve-se uma nova abordagem para detectar sites de *phishing* com base na análise dos comportamentos online dos usuários, ou seja, os sites que os usuários visitaram e os dados que os usuários enviaram a esses sites. Tais comportamentos do usuário não podem ser manipulados livremente pelos invasores; a detecção baseada nesses dados pode alcançar alta precisão, sendo fundamentalmente resiliente contra a mudança de métodos de fraude. Várias pesquisas foram feitas para proteger os usuários contra os ataques de *phishing*. A necessidade de descobrir automaticamente um alvo de *phishing* é um problema importante para os esforços *anti-phishing*. Existem algumas técnicas para identificar o site de *phishing*, as quais, algumas foram apresentadas nesse artigo.

Neste artigo publicado pela *International Journal of Information and Communication Technology Research* (9) foi apresentado uma abordagem de detecção de *phishing* baseada na verificação do código fonte da página, extraindo-se algumas características de *phishing* dos padrões *W3*, *World Wide Web Consortium*, para avaliar a segurança dos sites e verificar cada caractere no código-fonte da página da web, se for encontrado caractere de *phishing*, diminui-se o peso de segurança inicial. Por fim, calcula-se a porcentagem de segurança com base no peso final, a alta porcentagem indica um site seguro e outros indicam que o site provavelmente é um site de *phishing*. Verifica-se dois códigos-fonte de páginas da web para sites legítimos e de *phishing* e compara-se os percentuais de segurança entre eles. Foi descoberto que o site de *phishing* tem menos porcentagem de segurança do que o site legítimo; Nossa abordagem pode-se detectar o site de *phishing* com base na verificação das características “ruins” no código-fonte da página da web.

1.3 MOTIVAÇÃO

O *phishing* está em constante evolução e tornando-se uma ferramenta criminal mais sofisticada para roubar informações confidenciais e cometer crimes na Internet. Devido à gravidade do problema, a comunidade da Internet investiu uma quantidade significativa de esforços nos mecanismos de defesa. Atualmente, dois dos serviços mais populares que protegem os usuários deste tipo de ataque são o serviço Google Safe Browsing (10) e o serviço *Microsoft Smart Screen* (11). Ambos fornecem navegadores com listas negras de URL. Esses, por sua vez, protegem os usuários de visitar os URLs da lista negra. O maior problema desse modelo de proteção é que ele é reativo: um URL de *phishing* só pode ser incluído na lista negra se já tiver aparecido em outro

lugar, por exemplo, em um e-mail de spam ou denunciado por um usuário. Um modelo proativo, onde novos URLs de *phishing* podem ser identificados com precisão, é altamente desejável para melhor proteger os usuários.

Há alguns anos, as pesquisas sobre *phishing* na área de segurança cibernética analisam o tempo de remoção de sites de *phishing* e o número de visitantes que o sítio atrai, e tais pesquisas usam principalmente recursos como arquivos de log, *honeypots* e redes colaborativas para coletar dados. Os dois primeiros nem sempre são fáceis de obter ou de configurar. Quanto à primeira opção de recurso, os arquivos de log estão se tornando escassos, pois as pessoas estão usando ferramentas como o *Google Analytics* para rastrear o tráfego de cliques do website. Essas análises não são informações públicas por motivos compreensíveis. Os serviços de encurtamento de URL, que fornecem aos usuários um equivalente menor de qualquer URL longo fornecido, são um exemplo de serviço que, às vezes, fornece informações analíticas como informações públicas em seus URLs encurtados. Alguns provedores de encurtamento de URL, como *bit.ly* ou goo.gl, permitem visualizar em tempo real o tráfego de cliques de um determinado URL curto, incluindo referenciadores e países que o encaminham. Infelizmente, os invasores abusam dos serviços de encurtamento de URL, talvez para mascarar o destino final onde a vítima irá parar depois de clicar no link malicioso. Os ataques de *phishing*, em geral, são sítios que induzem o usuário a fornecer informações pessoais ou financeiras fazendo-se parecer como um sítio legítimo. Este tipo de ataque é também comum fora do mundo digital: estelionatários e golpistas estão sempre em busca de pessoas leigas (ou frágeis, como idosos), fazendo-se passar por uma entidade legítima a fim de enganá-las.

Segundo pesquisa (1) feita pelo laboratório especializado em *cyber* segurança da *PSafe*, o *dfndr lab*, o número de ataques cibernéticos quase dobrou no Brasil em 2018. Ao analisar os dados do 4° Relatório de Segurança Digital no Brasil, verificou-se mais de 120 milhões de detecções de ataques cibernéticos via links maliciosos no primeiro semestre de 2018. Este número representa um crescimento de 95,9% comparado com o mesmo período de 2017. A pesquisa mostra que o número de links maliciosos teve um crescimento de 12% no segundo trimestre em comparação com o primeiro trimestre do ano, passando de 56,9 milhões para 63,8 milhões de links maliciosos. O *WhatsApp* é o campeão de links maliciosos, principalmente em assuntos relacionados a futebol, como venda de camisas oficiais da Seleção Brasileira. Esses dados são extremamente preocupantes, pois, comparado à população total do Brasil, verifica-se que um a cada três brasileiros pode ter sido vítima de ataques cibernéticos somente no segundo trimestre do ano de 2018. Aproveitando esses resultados, verificou-se que a cada segundo, oito novos links maliciosos foram criados. O principal destaque dos ataques detectados é o *phishing* via aplicativo de mensagens que chegou a marca de 36,6 milhões de detecções nos meses de Abril, Maio e Junho, a frente de publicidades suspeitas, 12,2 milhões de detecções e de notícias falsas, 4,4 milhões de detecções.

Ainda segundo a mesma pesquisa, no segundo trimestre de 2018, as pessoas do sexo masculino acessaram mais links maliciosos em comparação com as do sexo feminino: 69% das detecções foram registradas a partir de acessos masculinos, enquanto 31% foram femininos.

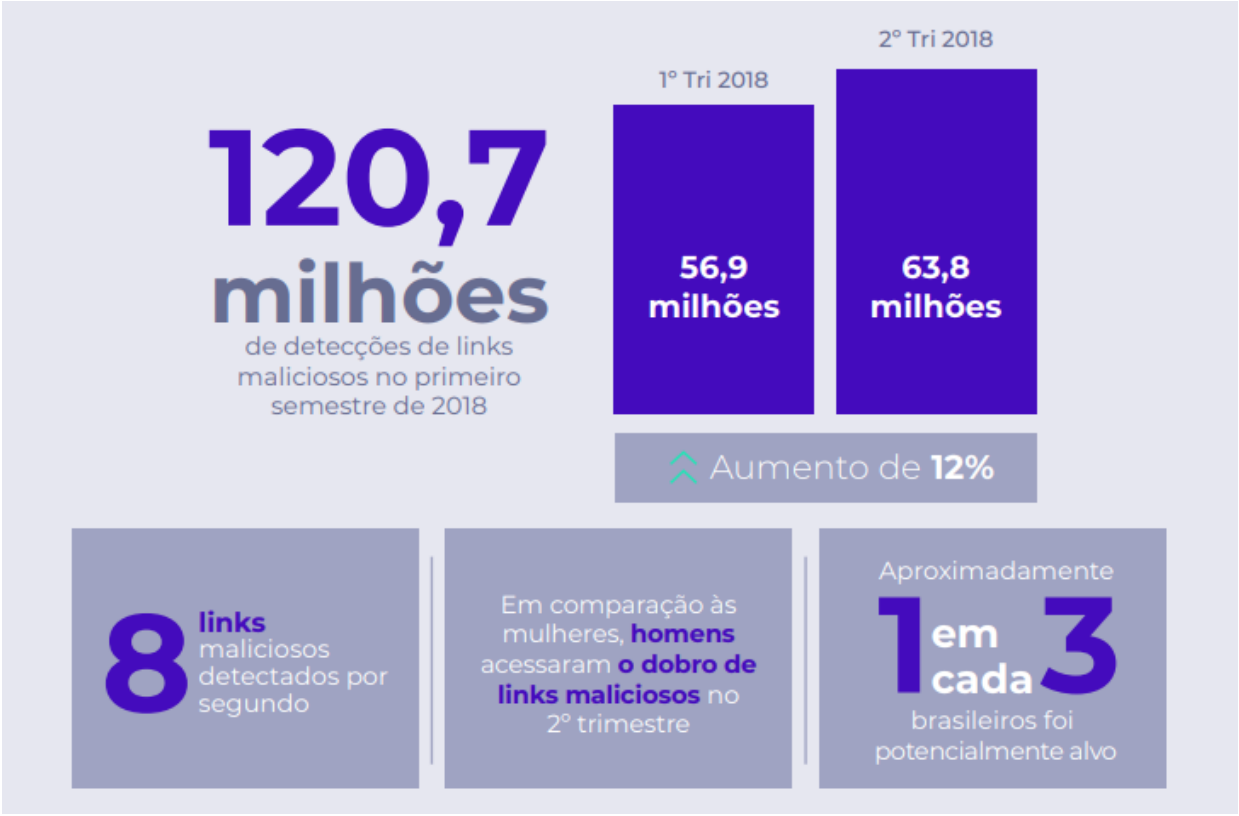


Figura 1.1: Gráfico mostra avanços dos ataques *phishing* no primeiro semestre de 2018 [Fonte: DFNDR LAB (1)].

Como forma de combate a esses ataques, decidiu-se criar uma ferramenta de detecção automática de *phishing* através de aprendizado de máquina, mais precisamente via Redes Neurais Artificiais sem Peso, como será detalhado no Capítulo 3. Logo, uma vez treinada, a rede neural é capaz de analisar a URL a ser acessada, possivelmente alertando o usuário e fazendo com que essa prática seja prevenida de forma proativa, protegendo assim os usuários.

1.4 OBJETIVOS

Nesta seção serão listados os objetivos gerais e específicos deste trabalho de conclusão de curso acerca do sistema de defesa contra ataques do tipo *phishing*.

1.4.1 Objetivo Geral

Propor uma ferramenta (serviço) que identifique automaticamente um possível ataque do tipo *phishing* através de elementos (características) básicos de uma URL e sua página correspondente, protegendo de maneira proativa os visitantes contra o roubo de informações pessoais e sensíveis.

1.4.2 Objetivos específicos

- Fazer uso de um mecanismo de pesquisa com o objetivo de encontrar, extrair, filtrar e avaliar as informações e recursos de todas as URLs associadas a um *website*.
- Extrair informações das URLs e outros elementos da página para criação de *features* objetivando a identificação de páginas maliciosas pelo sistema inteligente.
- Analisar a qualidade do *dataset* gerado por meio visual e com algoritmos disponíveis no *software weka*. Objetiva-se, pois, a validação dos mesmos para implementação na rede neural.
- Treinar a rede neural sem peso *WiSARD* com o *dataset* gerado anteriormente a fim de classificar sítios quanto à potencial presença ou não de *phishing*.

1.5 METODOLOGIA

Essa seção descreve a metodologia desenvolvida neste trabalho, a qual é composta por um conjunto de fases, métodos, técnicas e ferramentas para o apoio à detecção de *phishing* em *websites*. A metodologia deste trabalho é composta por 4 fases, sendo:

1. Obtenção de URLs a partir de um *webcrawler*: esta primeira etapa refere-se à obtenção de URLs. Para isso foi desenvolvido um *webcrawler* para obtenção de endereços não maliciosos. Esse programa, descrito em Python, percorre as páginas associadas a esses URLs e todos os *hiperlinks* contidos neles a partir de um link de partida. Por outro lado os links maliciosos foram obtidos do *PhishTank* (12), uma comunidade baseada no serviço *anti-phishing*.
2. Criação do *dataset* a partir dos dados extraídos das URL: nesta etapa escolheu-se quais elementos (*features*) presentes na URL e na página a ela associada serão avaliados, tendo como base inicial o trabalho desenvolvido por pesquisadores da *Google* (5). Em seguida realizou-se a extração e limpeza dessas informações para utilização nas etapas de análise do *dataset*.
3. Uso do *Weka* para validação do *dataset*: o *Weka* tem como objetivo agregar algoritmos provenientes de diferentes abordagens na área de estudo de aprendizagem de máquina. Esse ferramenta deve validar o *dataset* gerado na fase anterior antes da sua implementação (aprendizado) na rede neural.
4. Uso de uma rede *WiSARD* para classificação de *websites*, entre malicioso ou não: nesta última etapa é realizado o treino da rede neural *WiSARD* e sua integração ao sistema de detecção de *phishing* para efetivamente realizar-se a classificação dos endereços URL. Nesta etapa tambéms são definidas características importantes do sistema inteligente como um todo.

1.6 ESTRUTURA DO TEXTO

O restante deste trabalho está organizado em capítulos, como descrito abaixo:

- No Capítulo 2 é apresentada um referencial teórico a cerca da segurança cibernética e sobre os sistemas inteligentes, apresentando os conceitos e ideias necessários para o entendimento do projeto proposto.
- No Capítulo 3 é feito uma exposição de forma aprofundada e mais específica acerca de como as ferramentas apresentadas, no Capítulo 2, foram usadas para a obtenção de informações necessárias para o desenvolvimento do nosso projeto. Além disso realizou-se a descrição de características de URLs contaminadas por *phishings* e a descrição completa da implementação do sistema inteligente.
- No Capítulo 4 são apresentados todos os resultados e realizadas as análises que permitem a implementação de forma que o sistema possua o melhor desempenho e assertividade em cada uma de suas etapas.
- O Capítulo 5 encerra o trabalho, apresentando as principais conclusões acerca do tema e propostas de continuação e melhorias.

2 REFERENCIAL TEÓRICO

2.1 SEGURANÇA CIBERNÉTICA

A segurança cibernética é um grupo de métodos que tem como objetivo proteger as informações armazenadas nos dispositivos conectados com a Internet, incluindo a garantia de aspectos como: confidencialidade, confiabilidade e integridade. Esse processo de defesa das ameaças pode ser estruturado em quatro etapas: reconhecimento, análise, adaptação e execução.

Na etapa de reconhecimento, o *software* de segurança identifica o invasor. Quanto mais veloz for a detecção do invasor, melhor, pois os dados em atacados são protegidos mais rapidamente. Na etapa de análise, o *software* de segurança examina o que está sendo atacado e quais os melhores métodos para cada caso. A adaptação é a "nova cara" que a ferramenta irá obter, como se estivesse remontando o sistema de defesa e reforçando as "brechas" nos lugares certos. E por último, a execução que é o "combate imediato" da segurança para evitar o atacante. Essas são as quatro etapas básica para a proteção de qualquer dispositivo.

2.1.1 Táticas de ataque

Os ataques mais eficientes são os que tem alvos direcionados e específico, portanto causam mais danos. Todos os dispositivos com acesso à Internet estão sujeito a ataques. Há muitos tipos de ataques, sejam vírus, *worms*, *malwares*, *trojans*, *phishing*, etc. O lado bom é que existem várias maneiras de se evitar cada um deles. Mas, para isso, é necessário saber e entender quais os tipos de ataques cibernéticos.

2.1.1.1 Ataque DoS

O ataque DoS (sigla para *Denial of Service*), que é traduzido como "Ataques de Negação de Serviço", consiste em uma sobrecarga num servidor ou num computador fazendo com que os usuários não consigam executar as suas tarefas. Esse ataque é feito por um único computador, onde o autor faz com que o servidor ou computador atacado receba tantas requisições que não consiga realizar as suas tarefas. Em resumo, o dispositivo fica tão sobrecarregado que nega o serviço.

2.1.1.2 Ataque DDoS

O ataque DDoS (sigla para *Distributed Denial of Service*), que é traduzido como "Ataques de Negação de Serviço Distribuído", consiste em um computador mestre que escraviza outros computadores da Internet e fazem com que esses computadores ataquem uma determinada máquina.

Para que ataques como esse funcione é preciso que milhares de computadores façam parte desse "exército" que atacará o computador ou servidor alvo. Esse ataque é uma evolução do DoS.

2.1.1.3 *Backdoor*

O *Backdoor* é um tipo de ataque que permite a entrada de forma remota, através da Internet, em dispositivos, *software*, entre outras plataformas. Na maioria das vezes esse acesso não é autorizado e nem percebido pela vítima. Nesse ataque, o atacante tem acesso a várias informações confidenciais que estão contidas no computador atacado e até o controle total do ambiente. Esse tipo de ataque pode ocorrer de diversas formas, pode ser um *trojan*, que é um *software* que cria portas para outros *malware*, pode ser um código oculto em um programa instalado, através de *worms* instalados, através de sites, por conta de vulnerabilidades existentes em navegadores, entre outros.

2.1.1.4 *Spoofing*

O ataque *Spoofing* consiste em uma falsificação de endereços IP, mascarando o remetente através do pacote IP, ou seja, o computador atacado envia mensagens para o computador atacante que está usando um endereço IP que simula uma fonte confiável. Existem vários outros jeitos de usar o *spoofing*, como por exemplo, a falsificação de um e-mail ou a falsificação de um DNS.

2.1.1.5 *Phishing*

O *phishing* que corresponde a "pescaria" é um tipo de ataque que tem como objetivo a pesca de informações e dados pessoais através de armadilhas. Com isso, os atacantes conseguem dados bancários, senhas de cartões, senhas de redes sociais, entre outros.

Existem várias formas de acontecer o ataque por *phishing*. A maioria deles acontecem por meio de mensagens falsas enviadas por máquinas e e-mails com links corrompidos. Mas o modo que vem crescendo rapidamente são sites falsos, criados por especialistas, para imitar os sites de instituições, bancos e sites de vendas online, com o objetivo de roubar informações confidenciais de pessoas ou empresas.

O *phishing* acontece em seis etapas: planejamento, preparação, ataque, coleta, fraude e pós-ataque. Na etapa de planejamento, os atacantes escolhem os alvos e definem o objetivo do ataque, geralmente os alvos são usuários inexperientes que não sabem como o golpe funciona. Na etapa de preparação é onde os atacantes escolhem os tipos de dados que eles querem conseguir, se são dados bancários, dados pessoais sigilosos, entre outros. A etapa de ataque e coleta funcionam praticamente juntas, onde o atacante manda a mensagem ou o e-mail com o link malicioso e quando a vítima abre esse link é feito a coleta dos dados de interesse. E por fim, depois de feita a fraude e o roubo, os vestígios do ataque são eliminados, fazendo com que qualquer evidência seja removida, para evitar ou dificultar a identificação dos atacantes.

2.1.2 Motivações de um ataque

O avanço tecnológico tem vários pontos positivos, como o aumento da velocidade da internet, a facilidade e rapidez na comunicação entre redes de universidades ou empresas, a facilidade de negociação entre países. Mas também há alguns pontos negativos em tal avanço, como o roubo de informações pessoais ou confidenciais de forma anônima tanto para fins lucrativos ou políticos.

Os ataques cibernéticos podem ser feitos por vários motivos, alguns deles são: roubo de dados bancários, roubo de dados pessoais sigilosos, disseminação de notícias falsas, danificar a imagem de empresas, terroristas tentando infiltrar rede de computadores para obter dados de governos inimigos e furtar segredos ou sabotar equipamentos, entre outros.

Um exemplo de ataque cibernético internacional, é o roubo de informações que um país ou um conjunto de países fazem um para com os outros, com o objetivo de ter vantagens políticas, buscando a coleta, manipulação ou destruição de informações. Esse fenômeno é chamado de guerra cibernética por Steve Winterfeld e Jason Andress. (13)

Os ataques cibernéticos podem ser feitos por vários motivos, alguns deles são: roubo de dados bancários, roubo de dados pessoais sigilosos, disseminação de notícias falsas, danificar a imagem de empresas, terroristas tentando infiltrar rede de computadores para obter dados de governos inimigos e furtar segredos ou sabotar equipamentos, entre outros.

Outro exemplo, que motiva esses ataques é a disseminação de notícias falsas (*fake news*) sobre grandes eventos e datas comemorativas, como por exemplo as eleições, copa do mundo, promoções em sites comerciais (*Black Friday*), etc. Segundo pesquisa (1) feita pelo laboratório especializado em *cyber* segurança da *PSafe*, o *dfndr lab*, nos últimos meses do primeiro semestre de 2018, registrou-se que os grandes eventos são um chamariz para *cyber* criminosos, foram sessenta e nove ataques focados à copa do mundo e mais de seis milhões de detecções.

> "Faltando poucos dias para o início da Copa do Mundo, os primeiros *cyber* ataques relacionados ao evento começaram a surgir. Este comportamento já era previsto pelo time do *dfndr lab*, uma vez que os *hackers* costumam se aproveitar de eventos de grandes proporções, como os Jogos Olímpicos e outras datas comemorativas, para promover golpes. Durante a Copa, no entanto, um novo fato chamou a atenção: muitos sites falsos identificados eram, rapidamente, tirados do ar. Para driblar a situação, os criminosos utilizaram diversos links para divulgar os mesmos ataques. Foram mais de 6 milhões de detecções em 69 links maliciosos diferentes. Dentre os assuntos mais detectados pela equipe do *dfndr lab* estiveram os falsos sorteios de camisas da Seleção, responsáveis por 98,1% das mais de 6 milhões de detecções; falsos sorteios usando nomes de patrocinadores da Copa; e links de falsos sites de *e-commerce* prometendo produtos com descontos. Em todos os casos, os homens foram os principais alvos dos *hackers*." Fonte: DFNDR LAB (1).

Figura 2.1: Detecção e quantidade de golpes no primeiro semestre de 2018 relacionados a Copa do Mundo [Fonte: DFNDR LAB (1)].

Figura 2.2: Detecções por assunto de ataques nos meses finais do primeiro semestre de 2018 relacionados a Copa do Mundo [Fonte: DFNDR LAB (1)].

2.1.3 Vulnerabilidades

Os atacantes podem entrar em uma rede, sistema, *software* ou plataforma de várias maneiras diferentes. Um link malicioso aberto, um e-mail com spam, um sistema operacional ou um programa não atualizado, etc. Quando se fala de segurança das informações, tem que se levar em consideração o risco, que é uma relação entre impacto e probabilidade. O ideal é diminuir ao máximo a probabilidade de que um ataque cibernético seja bem sucedido e caso o ataque ocorra, o impacto seja o mínimo possível.

Para diminuir os riscos de ataques, deve-se conhecer as vulnerabilidades do seu ambiente. As vulnerabilidades podem ser físicas, que é tudo que está relacionado ao acesso às instalações de um ambiente. Pode ser por conta do *hardware*, que são todas as ferramentas ou itens de configuração mal instalados ou desatualizados. Pode ser por conta do *software*, pois todo programa ou sistema que está com pacotes de segurança ou versões desatualizadas são uma ótima porta de entrada para atacantes.

2.1.4 Mecanismos de defesa

A segurança de uma rede online é um assunto de extrema importância e merece a atenção dos usuários. Esse tópico é onde se encontra os maiores desafios para os profissionais de TI, que buscam reduzir as vulnerabilidades. Se manter protegido em uma rede corporativa ou até mesmo doméstica é uma tarefa árdua que pode demandar um alto custo e complexidade, exigindo investimento em ferramentas de segurança.

Algumas ferramentas de proteção contra *phishing* são oferecidas pelo mercado, como antivírus. Mas a melhor forma de se manter protegido é evitar o acesso de spams e mensagens duvidosas. Assim como fazer uso de *firewalls* e anti-spams.

Para ambientes domésticos recomenda-se:

- Tomar cuidado com a exposição excessiva, pois quanto maior a exposição em redes sociais, blogs ou sites, mais informações os criminosos terão para fazer o ataque.
- Evitar o acesso à sites de links enviados por e-mails duvidosos, pois tais links podem estar com vírus e outros agentes maliciosos. Geralmente esses e-mails ou sites falsos são repletos de erros gramaticais, promessas exageradas, solicitações de dados bancários.
- Evitar responder e-mails que pedem dados pessoais, como CPF e senha ou número de cartões de crédito. As empresas, bancos e instituições sempre avisam os canais no qual eles se comunicam com os clientes.
- Verificar o certificado de segurança do endereço do site navegado.
- Desconfiar de promoções e recompensas duvidosas, que necessite de informação sobre seus dados pessoais para ganhar algum prêmio ou beneficio.
- Manter o computador sempre atualizado, manter o sistema operacional e os aplicativos atualizados.
- Em casos de dúvida, perguntar se o e-mail pertence realmente a instituição mencionada, entrar em contato por um número confiável para esclarecer as dúvidas.

Para ambientes corporativos recomenda-se:

- Tomar cuidado com a saúde dos equipamentos e promover um ambiente seguro para os funcionários, para um melhor resultado deve-se instruir os funcionários sobre as práticas de segurança.
- Fazer uso de antivírus, *firewalls*, anti-spams, políticas de *proxy*.
- Alinhar as equipes de marketing e de segurança de TI, para proteger a empresa e os clientes de tentativa de *phishing*, para tal, é ideal que essas duas equipes trabalhem em conjunto.

A segurança em TI monitora o tráfego das atividades de rede da empresa e se mantém atualizada sobre as ameaças ou fraudes. Já o pessoal do marketing cria as campanhas de e-mail, logo estão conscientes das mensagens criadas pela empresa. Além disso, esse conjunto tem a visão geral de todos os domínios que a empresa utiliza, com isso, são capazes de identificar domínios indevidos, que podem ser provenientes de ataques, de forma rápida. Com essa junção das equipes, a empresa pode se planejar, identificar e responder aos ataques de forma mais eficiente.

2.2 SISTEMAS INTELIGENTES

Com a evolução dos sistemas computacionais e da comunicação entre eles, a necessidade de se utilizar ferramentas para tomada de decisão no menor tempo e da melhor maneira possível, se faz cada vez mais prescindível para se ter sucesso. Em meio a essas necessidades, a utilização de Sistemas inteligentes, surgiram como uma excelente opção, pois tem uma ótima capacidade de resposta aos problemas. Esses sistemas levam à automações, produtividade e sustentabilidade, esses são os três elementos fundamentais de um sistema para que ele seja equilibrado e funcional.

Criar um sistema computacional para substituir os humanos na tomada de decisões e na realização de atividades complexas é uma tarefa difícil. Então o conceito de Inteligência Artificial (IA) surgiu para procurar maneiras de tornar essa substituição possível, pois ajudaram os especialistas no desenvolvimento de sistemas computacionais com características de inteligência, que realizam tarefas que o ser humano faz só que com um melhor desempenho. Então um sistema inteligente tem as habilidades de aprender ou compreender com a experiência, entender mensagens ambíguas ou contraditórias e responder de forma rápida e correta a uma nova situação.

2.2.1 Webcrawler

A Internet é a rede de computação global compartilhada que permite comunicações globais entre todos os dispositivos conectados. Há um crescimento espetacular em fontes e serviços de informação baseados na web. Estima-se que haja aproximadamente o dobro de página web a cada ano. À medida que a web se torna maior e mais diversificada, os mecanismos de pesquisa também assumem um papel central na infraestrutura da *World Wide Web*, tornando-se cada vez mais necessário que os usuários usem essas ferramentas automatizadas para encontrar, extrair, filtrar e avaliar as informações e recursos desejados. Na Internet, os dados são altamente desestruturados, o que torna extremamente difícil pesquisar e recuperar informações valiosas. Os mecanismos de pesquisa definem o conteúdo por palavras-chave. Além disso, com a transformação da web na principal ferramenta para o comércio eletrônico, é imperativo que organizações e corporações, que investem milhões em tecnologias de Internet e intranet, acompanhem e analisem os padrões de acesso dos usuários. Esses fatores geram a necessidade de criar sistemas inteligentes do lado do servidor e do cliente que possam efetivamente explorar o conhecimento na Internet. Essas

organizações que fornecem informações e serviços, como suporte automatizado ao cliente, compras online e uma infinidade de recursos e aplicativos, educação à distância, colaboração online, noticiários, etc, estão se tornando prática comum e generalizada.

Um dos programas de pesquisa utilizados é o *webcrawler* que, de uma ou mais sementes de Dado (link de partida) URLs, percorre as páginas onlines associados a esses URLs e todos os hiperlinks contidos neles. O *webcrawler* é um componente importante dos mecanismos de pesquisa da Internet, onde são usados para coletar o texto base de páginas da web indexadas pelo mecanismo de pesquisa. Eles são programas que exploram a estrutura gráfica da web para mover de uma página para outra. O enorme tamanho e a natureza dinâmica da *World Wide Web* destacam a necessidade de suporte contínuo e atualização de sistemas de recuperação de informações baseados na web. A última dimensão fundamental refere-se às estratégias de avaliação do rastreador necessárias para fazer comparações e determinar as circunstâncias nas quais um ou outro rastreador trabalha melhor. Os rastreadores facilitam o processo seguindo os hiperlinks em sites para baixar automaticamente um instantâneo parcial da web.

2.2.2 Detecção e prevenção de phishing

Para criar mecanismos de detecção e defesa contra o *phishing*, foram criados vários métodos, que são divididos em dois tipos, um tipo voltado para vertente humanizada e o outro para uma vertente computacional.

O primeiro tipo é a conscientização do usuário, que se da única e exclusivamente a percepção que o usuário que está utilizando o serviço tem de dizer se é ou não um *phishing*. O segundo tipo é mais voltado para o lado computacional, que é a detecção por *software*, e se divide em alguns ramos. O primeiro ramo é a *Blacklist*, que é basicamente uma lista negra criada com várias URLs maliciosas que foram identificadas, por meio de análises extensivas ou *crowdsourcing*. No entanto, as *blacklists* estão sendo classificadas como uma forma fraca de detecção de *phishing*, apesar de sua simplicidade e facilidade de implementação, sofre de falsos negativos, devido à dificuldade em manter as extensas listas atualizadas.

O segundo ramo é a Heurística, esse método tem melhores recursos de generalização do que a *blacklist*, pois também tem a capacidade de detectar ameaças em novos URLs. Uma versão mais específica das abordagens heurísticas é através da análise da dinâmica de execução da página da web. Aqui também, a ideia é procurar por uma assinatura de atividade maliciosa, como criação incomum de processo, redirecionamento repetido, etc. Esses métodos geralmente exigem visitar a página da web e, assim, as URLs podem realmente fazer um ataque.

O terceiro ramo é a Similaridade visual, um recurso importante de uma página web de *phishing* é sua semelhança visual com sua página da web de destino. Assim, com o uso de ferramentas, um proprietário de uma página legítima pode detectar URLs suspeitos e comparar as páginas da web correspondentes com a verdadeira em aspectos visuais. Se a semelhança visual de uma página falsa com a página verdadeira for alta, o proprietário será alertado e poderá tomar todas as

medidas para evitar, imediatamente, possíveis ataques de *phishing* e, assim, proteger sua marca e reputação. Geralmente, a similaridade visual entre duas páginas da web é medida em três métricas: semelhança no nível de bloco, semelhança de *layout* e similaridade de estilo geral. Todas essas três métricas de similaridade visual são definidas com base na segmentação de uma páginas. A página é primeiro decomposta em um conjunto de blocos salientes. A similaridade do nível de bloco é definida como a média ponderada das semelhanças de todos os pares de blocos correspondentes. A semelhança de *layout* é definida como a proporção do número ponderado de blocos correspondidos para o número total de blocos na página verdadeira. A similaridade geral de estilo é calculada com base no histograma do recurso de estilo. O coeficiente de correlação normalizado dos histogramas das duas páginas da web é a similaridade geral de estilo.

O quarto e último ramo é o aprendizado de máquina, esta abordagem tenta analisar as informações de um URL e seus sites correspondente ou páginas, extraindo bons recursos de uma determinada URL. Com esse recursos definidos, acontece o treinamento de uma máquina, com bases nos dados extraídos da URL, com o objetivo de formar um modelo de previsão. Existem dois tipos de recursos que podem ser usados - recursos estáticos e recursos dinâmicos. Na análise estática, realiza-se a análise de uma página com base em informações disponíveis sem executar a URL (ou seja, a execução de *JavaScript*, ou outro código). Os recursos incluem recursos lexicais extraído da sequência de URL, informações sobre o anfitrião, e às vezes até mesmo conteúdo *HTML* e *JavaScript*. Como nenhuma execução é necessária, esses métodos são mais seguros do que as abordagens dinâmicas. A suposição subjacente é que a distribuição desses recursos é diferente para URLs maliciosas e benignas. Usando essas informações de distribuição, um modelo de previsão pode ser construído, o que pode fazer previsões em novos URLs. Devido ao ambiente relativamente mais seguro para extrair informações importantes e à capacidade de generalizar para todos os tipos de ameaças (não apenas as mais comuns), as técnicas de análise estática têm sido extensivamente exploradas pela aplicação de técnicas de aprendizado de máquina. As técnicas de análise dinâmica incluem o monitoramento do comportamento dos sistemas que são potenciais vítimas, para procurar qualquer anomalia. As técnicas de análise dinâmica têm riscos inerentes e são difíceis de implementar e generalizar. Portanto é mais comumente utilizado a análise estática.

2.2.3 Weka

O *Weka* (*Waikato Enviroment for Knowledge Analysis*) é um pacote de *software* que contém diversas técnicas de mineração de dados, totalmente gratuito, pois é um *software* livre e encontra-se licenciado pela *General Public License* (GPL). O principal objetivo do *Weka* é agrupar algoritmos de diferentes abordagens da Inteligência Artificial. A linguagem de escrita dele é o *Java*, mas devido a sua característica que é a sua portabilidade, ele pode ser utilizado em diferentes sistemas operacionais.

O *Weka* funciona mediante a análise, estatística ou não, de dados computacionais fornecidos, usando o princípio da mineração de dados. A partir desses dados, o *software* tenta encontrar padrões e gerar hipóteses para solucionar os problemas de forma rápida e eficiente.

A organização dos dados desse *software* possui um formato *ARFF*, no qual várias informações devem estar presentes para o seu pleno funcionamento, como por exemplo, o domínio do atributo, valores que os atributos podem representar e atributo de classe.

2.2.3.1 Método *"cross-validation"*

A validação cruzada tornou-se um método padrão na análise de desempenho de algoritmos e modelos em *machine learning* e reconhecimento de padrões. Para começar, divide-se, de forma aleatória, o conjunto de informações ou dados em três partes: conjunto de exemplos de treinamento, conjunto de exemplos de validação e conjunto de exemplos de teste. O conjunto de exemplos de treinamento é usado para treinar o algoritmo. O conjunto de exemplos de validação é usado para verificar a generalização do algoritmo e ajustar parâmetros. Depois do algoritmo treinado, o conjunto de exemplos de teste é usado para testar o algoritmo, avaliando sua generalização sobre o conjunto de testes. Deve-se tomar o cuidado de não utilizar o conjunto de testes para ajustar parâmetros.

A taxa de acerto, em porcentagem, é calculada da seguinte maneira: número de exemplos classificados corretamente dividido pelo número total de exemplos. E a taxa de erro, em porcentagem, é o número de exemplos classificados incorretamente dividido pelo número total de exemplos.

Para calcular o desempenho médio e variância, respectivamente. Deve-se treinar o algoritmo n vezes com parâmetros diferentes e avaliar os n algoritmos treinados, que será a taxa de acerto. A partir dessa avaliação calcula-se a taxa de acerto médio e a variância.

2.2.3.2 Método "percentage split"

O método *Percentage Split* ou *Holdout* consiste na divisão dos exemplos de treinamento em dois subconjuntos distintos, um para treinamento, outro para teste. O valor das porcentagens de cada um dos conjuntos é geralmente expresso numa única porcentagem maior que 50%, estando subentendido que o conjunto menor é o complemento para 100%. O valor da divisão mais comum é 66% para treinamento e 34% para teste, embora haja evidências empíricas que justifiquem essa escolha de 2/3 e 1/3.

A grande vantagem desse método é a sua simplicidade, mas dependendo da composição obtida, as classes dos exemplos podem não estar igualmente representadas nos dois conjuntos. Outra limitação desse método está no fato de que menos exemplos são usados no treinamento, podendo ter um impacto no desempenho do modelo induzido.

Para conjuntos de teste muito pequenos, dividir o já escasso número de exemplos de teste pode ter um efeito terrível ou na geração do modelo ou na sua avaliação de desempenho.

2.2.3.3 Algoritmo J48

O algoritmo J48 foi criado depois que surgiu a ideia de reformular o algoritmo C4.5, que na sua forma original, é escrito na linguagem de programação C, para a linguagem *Java* (14). Baseado em um conjunto de dados de treinamento, ele tem gera uma árvore de decisão, este modelo é utilizado para a classificação das instâncias do conjunto de teste.

Esse algoritmo é bastante usado pelo fato de se mostra conveniente para os processos envolvendo as variáveis qualitativas contínuas e discretas encontradas nas bases de dados. Esse algoritmo foi apresentado por QUINLAN, em 1993, (15) e é tido como o que apresenta melhor qualidade de montagem da árvore de decisão, com base no conjunto de dados de treinamento. O J48 faz uso do método de dividir para conquistar, na montagem da árvore, onde um programa complexo é dividido em subprogramas mais simples, aplicando, de modo recíproco, a mesma técnica a cada subprograma, decompondo o espaço determinado pelos atributos em subespaços, associando-se a eles uma classe.

2.2.3.4 Algoritmo RandomTree

O *RandomTree* é uma ferramenta de suporte à decisão que usa um modelo de decisões em forma de árvore e suas possíveis consequências, incluindo resultados de eventos aleatórios , custos de recursos e utilidade . É uma maneira de exibir um algoritmo que contém apenas instruções de controle condicional. As árvores de decisão são construídas analisando um conjunto de exemplos de treinamento para os quais os rótulos de classe são conhecidos. Eles são então aplicados para classificar exemplos não vistos anteriormente. Se treinados em dados de alta qualidade, as árvores de decisão podem fazer previsões muito precisas.

Uma árvore de decisão classifica os itens de dados colocando uma série de perguntas sobre os recursos associados aos itens. Cada questão está contida em um nó e cada nó interno aponta para um nó filho, para cada resposta possível à sua pergunta. As perguntas formam, assim, uma hierarquia, codificada como uma árvore.

As árvores de decisão são, às vezes, mais interpretáveis do que outros classificadores, como redes neurais e máquinas de vetores de suporte, porque combinam questões simples sobre os dados de maneira compreensível. Abordagens para extrair regras de decisão de árvores de decisão também foram bem sucedidas. Infelizmente, pequenas alterações nos dados de entrada podem levar a grandes mudanças na árvore construída. As árvores de decisão são flexíveis o suficiente para manipular itens com uma mistura de recursos reais e categóricos, bem como itens com alguns recursos ausentes. Eles são expressivos o suficiente para modelar muitas partições dos dados que não são tão facilmente alcançadas com classificadores que dependem de um único limite de decisão (como regressão logística ou máquinas de vetor de suporte). No entanto, até mesmo dados que podem ser perfeitamente divididos em classes por um hiperplano podem exigir uma grande árvore de decisão se apenas testes de limite simples forem usados. As árvores de decisão suportam naturalmente problemas de classificação com mais de duas classes e podem

ser modificadas para lidar com problemas de regressão. Finalmente, uma vez construídos, eles classificam novos itens rapidamente.

2.2.4 Redes neurais artificiais

Todas as células dos seres vivos são capazes de reagir a estímulos químicos. Essas reações aos estímulos químicos, são os responsáveis pela comunicação entre as células, para seres formados por mais de uma célula (seres multicelulares), que permite ao organismo realizar as mais diferentes funções em vários níveis de complexidade diferentes, como por exemplo, o metabolismo e a respiração, algumas vezes envolvendo vários tecidos vivo no processo. As células responsáveis por essa comunicação intercelular são as células neuronais, também chamados de neurônios. Os neurônios fazem essa atividade de uma forma tão sofisticada que vai muito além do que qualquer comparação com os outros tipos de células, sendo indispensável e a mais importante célula do Sistema Nervoso.

A primeira proposta de simulação do comportamento de células neurais dos seres vivos por computadores foi feita por McCulloch e Pitts (16), e foi bem aceita no campo da Inteligência Artificial. O modelo apresentado é um modelo extremamente simplificado do neurônio. Da mesma forma que um neurônio real tem seu comportamento alterado de acordo com o estímulo recebido, assim o neurônio artificial também o tem, isso faz com que ele responda de maneira semelhante ao neurônio real quando lhes são apresentados ao mesmo estímulo. Ou seja, ao ser submetido a treinamentos, o neurônio artificial é capaz de aprender como se deve agir a determinados estímulos.

2.2.4.1 O Método N-tuple

Todos os modelos baseados em RAM são fundamentados pelo método N-tuple (N-tuplas). Esse método projeta que a aprendizagem e o reconhecimento de uma imagem tem que ser elaborado através de funções lógicas. Esse mecanismo foi proposto pela primeira vez por Bledsoe e Browning (17). A imagem é dividida em várias partes de tamanhos iguais e fixos de pixel, essas partes são chamadas de Upla, e o aprendizado é dado ao criar fórmulas lógicas conjuntivas para cada uma dessas tuplas, de forma que essas fórmulas, retornem verdadeiro para cada uma das tuplas. Pegou-se como ilustração o exemplo do livro (18). A Figura 2.3b apresenta o mapeamento dos bits para a letra “I” e a Figura 2.3c para a letra “T”. A Figura 2.3a mostra as variáveis de cada um dos pixel divididos da imagem.

Para esse exemplo, é empregado um tamanho fixo de 3 pixel por tupla para o Classificador N-tuple, formando um 3-tuple. Então ao fazer o treinamento da imagem referente à letra “I”, obtém-se a seguinte fórmula:

$$[RI = A.B.C + \bar{D}.E.\bar{F} + G.H.I] \tag{2.1}$$

A	B	C
D	E	F
G	H	I

(a) Mapeamento das variáveis.

1	1	1
0	1	0
1	1	1

(b) Binarização da letra *I*.

1	1	1
0	1	0
0	1	0

(c) Binarização da letra *T*.

Figura 2.3: Mapa de bits para diferentes cenários.

Já para a letra “T”, obtém-se a seguinte fórmula:

$$[RT = A.B.C + \bar{D}.E.\bar{F} + \bar{G}.H.\bar{I}] \tag{2.2}$$

Ao submeter-se a um outro classificador de letras “T”, uma vez que o classificador N-tuple tenha sido treinado, as fórmulas assimiladas podem ser utilizadas no reconhecimento de imagens para atribuir valores numéricos inteiros para cada um dos padrões mostrados. Esses valores serão apresentados a outros classificadores, para decidir-se de à qual classe pertence o atributo apresentado na imagem. Em um exemplos simples, utilizou-se as fórmulas apresentadas para as letras “T” e “I” para decidir a qual classe pertence o padrão exibido na Figura 2.4.

1	1	1
0	1	0
1	1	1

Figura 2.4: Padrão a ser classificado.

Mantém-se os mesmos mapeamentos utilizados no treinamento, fazendo com que as fórmulas dos classificadores de “T” e “I” também se mantenham.

Então os valores das fórmulas dos classificadores são comparados entre si e o maior valor retornado pela equação é o da classe vencedora. Nesse exemplo, o maior valor retornado é o da classe “I”, que é RI = 2, o que corresponde que o padrão pertence à classe “I”.

2.2.4.2 Redes Neurais sem peso

As Redes Neurais Artificiais Sem Peso (RNSP) apresentam um treinamento mais rápido e sem necessidade de um alto poder computacional se comparado com às Redes Neurais Artificiais Com Peso, tornando-se excelente alternativa. As RNSP são baseadas em neurônios artificiais que são compostos de entradas e saídas binárias e não possuem pesos entre suas conexões neurais.

Os neurônios de uma RNSP armazenam seu conhecimento em uma tabela verdade, no qual pode ser implementada usando memórias de acesso aleatório (*Random Acess Memory* – RAM). Deste modo, o processamento de informações desses neurônios se dá pelo endereçamento de sua tabela verdade segundo o padrão de entrada, já a saída do neurônio é baseada no respectivo conteúdo da posição da tabela verdade acessada (19).

Algumas características principais que difere uma Rede Neural Artificial Sem Peso das demais redes neurais são onde as informações aprendidas ficam localizadas. Como viu-se as RNSPs armazenam as informações em tabelas verdade. Já as RNAs convencionais armazenam as informações nos pesos das conexões. Outra característica é que nas RNSPs as entradas são sempre discretas e os seus neurônios são capazes de computar todas as funções booleanas de suas entradas digitalizadas e nas RNAs com peso as entradas e pesos podem assumir quaisquer valores reais.

2.2.4.3 Neurônio artificial sem peso ou RAM

Os neurônios artificiais com peso calculam sua saída com base na entrada e nos pesos entre suas conexões. De modo diferente, nos neurônios sem peso o processamento da saída do neurônio depende apenas de sua entrada e dos valores armazenados em sua RAM.

A Figura 2.5 mostra o funcionamento de um neurônio baseado em RAM de N bits de entrada que são os que endereçam a memória RAM do neurônio. O neurônio obrigatoriamente terá que possuir uma memória de 2^N posições endereçáveis, para que todas as combinações dos N bits de entrada possuam uma posição de memória. Como mostrado na Figura 2.5, a entrada do neurônio recebe um valor binário I = I1, I2, I3, ..., IN que endereçará as posições de memória RAM do neurônio, M[0], M[1] ... M[$2^N - 1$]. A função de ativação do neurônio RAM então processa o conteúdo da memória acessada e produz uma saída.

A função de saída S da RNSP geralmente utilizada é a função identidade, podendo ser definida na Equação 2.3:

$$S = \begin{cases} 0 & se \quad M[I] \ = \ 0 \\ 1 & se \quad M[I] \ = \ 1 \end{cases} \tag{2.3}$$

Então, o bit M[I] armazenado na posição de memória endereçada por I representa a saída ou resposta do neurônio à entrada I, isto é, S = M[I]. Ou seja, o conteúdo da memória RAM determina a função executada pelo neurônio.

O neurônio baseado em RAM pode computar todas as funções binárias de suas entradas, mas é importante salientar que um único neurônio sem peso não tem capacidade de generalização. A generalização seria a capacidade de prover a saída correta para um padrão de entrada não conhecido com base nos padrões já conhecidos e utilizados para treinamento da rede. Então os neurônios sem peso só conseguem produzir a saída correta para padrões de entradas armazenados

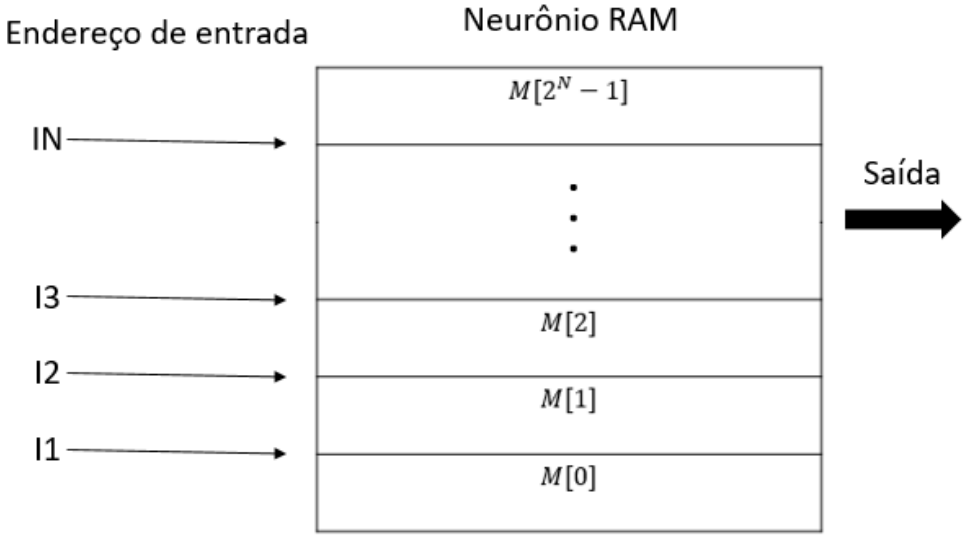


Figura 2.5: Modelo de um neurônio artificial sem peso.

na fase de treinamento. Mas as RNSPs formada por vários neurônios RAM tem a capacidade de generalização (19).

Uma RNSP com vários neurônios na camada de entrada, tem a entrada da rede separada em várias partes, que são chamadas de tuplas, no qual são usadas para endereçar cada um dos neurônios da rede.

2.2.4.4 Fase de treinamento de uma RNSP

O treinamento supervisionado é o tipo mais usado para as RNSPs. Funcionando da seguinte maneira, para cada parte do treinamento, tem uma saída pretendida para aquela parte, e os parâmetros da rede são modificados com base na saída pretendida. O treinamento das RNSPs tradicionais é formado basicamente em substituir os valores armazenados inicialmente na tabela verdade pelos valores equivalentes à saída pretendida para o neurônio.

Antes do treinamento da rede, o seu conteúdo de memória possuirá um valor inicial, que geralmente é aleatório, mas também pode ser predeterminado. Nos casos onde o neurônio tem apenas um bit para ocupar as posições da memória, isto é, a palavra da memória possui apenas um bit, esse valor terá que ser inicializado com o valor ‘0’. No decorrer do treinamento, para cada amostra entregue à rede, uma tupla desta amostra será empregada como entrada de cada um dos neurônios da camada de entrada da rede. Será de responsabilidade da tupla o endereçamento do neurônio ao qual a tupla foi apresentado, de forma a acessar o conteúdo de memória do neurônio equivalente ao endereço dado pela tupla.

Então, a saída do neurônio será determinada, para cada amostra de treinamento apresentada à rede, que condiz ao valor armazenado na posição da RAM endereçada pela entrada. A partir,

desse valor esperado para aquela determinada amostra de treinamento, o valor salvo na referida posição da memória será regulado.

Para arquitetura de camada única, os neurônios são preparados para retornar um valor de saída igual a '1' para os padrões de conjunto de treinamento, e somente para aqueles padrões. No caso em que mais de duas categorias sejam impreteríveis, várias redes RAM devem ser usadas em paralelo, sendo que cada uma das redes responderá ao treinamento de uma classe do padrão com o nível lógico '1', e para as outras classes com o nível lógico '0'.

O treinamento da RNA só acaba quando todas as amostras tenham sido treinadas e concedida à rede, e a mesma seja definida com base nos valores desejados para cada uma das amostras. O treinamento da rede se encerra quando for atingido um erro abaixo de um valor desejado predeterminado. Já no caso de uma RNSP convencional, o treinamento baseia-se em apresenta todas as amostras de treinamento à rede, trocando as posições de memória acessadas de '0' para '1'. Esse treinamento é conhecido como "*one shot learning*", que é traduzida como "um tiro de aprendizagem".

Treinar uma RNSP tem várias vantagens com relação aos algoritmos de treinamento das RNAs convencionais, pois são muito mais rápidos e flexíveis. A velocidade do processo de treinamento nas RNSPs é devido à existência de mútua independência entre os nós quando as entradas da rede são alteradas, o que contrasta com as RNAs com peso (19). Os treinamentos nos modelos com peso são mais complexos, pois quando os valores dos pesos são alterados, a rede, tendo em vista um determinado padrão de entrada-saída, irá mudar o comportamento do neurônio para outros padrões anteriormente aprendidos.

2.2.4.5 Fase de teste de uma RNSP

A fase de teste consiste em apresentar um padrão desconhecido à rede e esta deverá classificar o padrão como pertencente ou não à classe no qual foi treinada. Se a rede tiver várias camadas paralelas para identificar várias classes, o padrão desconhecido será mostrado a todas as camadas, e é esperado que apenas a camada que foi treinada para reconhecer a classe correta, saiba identificar o padrão apresentado. Se o padrão mostrado fizer parte do conjunto de treinamento, todos os neurônios da Rede Neural Artificial Sem Peso identificarão as tuplas do padrão e transmitirão saídas lógicas '1'. Mas se o padrão apresentado for semelhante aos padrões do treinamento, só parte dos neurônios identificarão as tuplas do padrão e transmitirão saídas lógicas '1'. Desse modo, a saída de todos os neurônios da rede é somada por um integrador, com o objetivo de tornar a RNSP capaz de generalizar. Então com base na quantidade de neurônios que transmitiram saída '1', estipula se o padrão desconhecido deve ser ou não reconhecido pela rede. Este tipo de Rede Neural Artificial Sem Peso modificada foi utilizada para o desenvolvimento da rede WiSARD (20), uma das aplicações mais conhecidas de RNSP.

2.2.5 A rede WiSARD

Na época em que o método n-tuple foi desenvolvido, o processo de registrar as operações conjuntivas extraídas, durante o treinamento, das várias tuplas da imagem, e depois recuperar essas fórmulas de maneira rápida e eficiente para a classificação de outra imagem era um desafio tecnológico para ser usado com o poder computacional, em situações reais. Então tempos depois, foi criada a rede WiSARD (Wilkie, Stonham and Aleksander's Recognition Device) (20) que é a Rede Neural Artificial Sem Peso que tem o maior destaque na ciência. Essa rede traz uma solução mais simples e eficiente, que viabilizou as aplicações práticas desse método, ainda por cima, pode ser implementada a partir de componentes de baixo custo e mesmo assim sendo capaz de operar em tempo real. É uma rede de reconhecimento de padrões e de um único estágio de neurônios (19). A rede WiSARD modifica o conteúdo das memórias RAM com o intuito de acumular aprendizado.

Um neurônio RAM sozinho é capaz de identificar um padrão para o qual ele foi treinado, mas não de generalização. Mas uma rede implementada com por vários desses neurônios seria capaz de generalizar. Então a rede WiSARD sugere que os neurônios sejam arrumados em uma estrutura que se chama discriminador.

Um discriminador nada mais é que uma rede de camada única com A neurônios RAM de B entradas cada. Então esse discriminador será capaz de aprender e identificar um subconjunto de um padrão com tamanho AxB. Cada um dos neurônios é ligado à entrada, que são as tuplas, fazendo com que aprenda apenas uma parte dela na fase de treinamento. Na etapa de teste, a saída de cada um dos discriminadores para um padrão é um número binário com A bits, o qual equivale à saída dos k neurônios da rede. Esses A bits são processados por um somador, resultando como saída o número dos A neurônios RAM que identificaram a tupla correspondente àquele padrão de entrada. A Figura 2.6 apresenta a estrutura básica de uma rede WiSARD.

> "Na Figura 2.6, é possível exemplificar o endereçamento dos neurônios na camada de entrada da rede. Observa-se uma imagem 8x8 pixel, os quais devem ser pixel binários, ou então deverão ser pré-processados antes de endereçar a rede. Cada um dos M neurônios da rede será endereçado por N pixel da imagem. Essa palavra de N pixel que endereça os neurônios é a tupla da rede, a qual deve possuir o mesmo tamanho para todos os neurônios ((19); (21)). Os pixel da imagem que formarão as tuplas podem ser escolhidos randomicamente ou de modo que cada neurônio seja responsável por identificar uma parte determinada da imagem. Por exemplo, para uma imagem 8x8 pixel e uma tupla de 8 bits, cada neurônio poderia ser responsável por identificar uma linha da matriz 8x8 de pixel". [Fonte: MACHADO, 2017, pgs.30 e 31 (2)]

Uma rede do tipo WiSARD com apenas um discriminador é capaz de classificar somente duas classes, ou seja, reconhecer se o padrão pertence ou não à classe à qual foi treinada. Para solucionar questões envolvendo mais de dias classes, a rede necessita ser replicada em paralelo

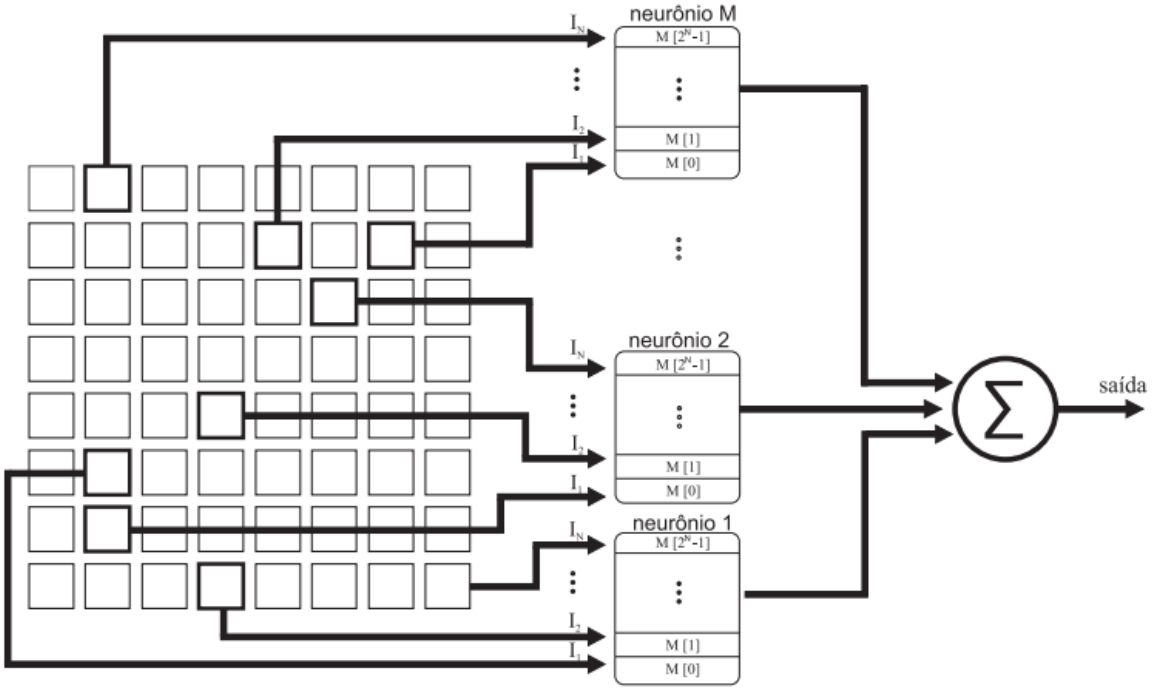


Figura 2.6: Estrutura da WiSARD. [Fonte: MACHADO, 2017, p.31 (2)].

em várias camadas, no qual é chamado de multi-discriminador (20). Um multi-discriminador nada mais é que diversos discriminadores dispostos em paralelo, sendo que cada um é treinado para identificar uma classe de padrões. Então, para uma rede WiSARD ser capaz de reconhecer X classes de padrões, ela terá que ser formada por X discriminadores em paralelo.

2.2.5.1 Treinamento da rede WiSARD

O treinamento de uma rede WiSARD acontece do mesmo modo de uma RNSP comum. Inicialmente todo o conteúdo das posições de memória está com o nível '0', porque a rede ainda não foi treinada. Para o treinamento deve-se basicamente modificar o conteúdo de memória RAM de todos os neurônios da rede, de acordo com o que o neurônio vai aprender. Para designar qual o endereço da RAM que será alterado para o treinamento, o padrão de entrada é fracionado de forma randômica em M grupo de N bits, o que são chamados de tuplas. As tuplas da rede são apontadas como as entradas dos M neurônios da rede. O valor do conteúdo de cada um dos M neurônios serão modificados para '1', quando forem endereçados pelos N bits, de modo que o treinamento para o padrão descrito está sendo descoberto. A escolha aleatória das tuplas de entrada deve sempre ser a mesma, para todos os discriminadores, da mesma maneira que para todos os padrões de entrada, para que a integridade do treinamento seja garantida. A rede WiSARD tem como característica o treinamento supervisionado, então, é preciso informar à qual classe um determinado padrão de treinamento faz parte. Para o caso de uma rede com multi-discriminador, somente a camada responsável pela identificação do padrão da amostra de treinamento terá sua memória arrumada. Dessa maneira, cada amostra de treinamento tem um valor desejado que é usado para

definir qual a camada da rede que será modificada para aprender o padrão. O Algoritmo 1 mostra o treinamento de uma rede WiSARD:

Algoritmo 1: Algoritmo de treinamento de uma rede WiSARD.

```
1 Iniciar com valor lógico '0' a memória de todos os M neurônios;
2 para cada amostra (k) de treinamento faça
3 |   Dividir a amostra em M tuplas de N bits;
4 |   Endereçar os M neurônios da camada com as M tuplas, com base em d(k);
5 |   Escrever o valor lógico '1' na memória de cada um dos M neurônios endereçados pelas tuplas;
6 fim
```

Quanto maior for a tupla da rede, menos neurônios irão compor os discriminadores. Então a quantidade de neurônios que compõem os discriminadores está diretamente relacionada à quantidade de bits que vão endereçar cada neurônio. A relação entre o tamanho da tupla e o número de neurônios no estágio de entrada da rede WiSARD é definida conforme a Equação 2.4 abaixo:

$$W = M * N \tag{2.4}$$

onde W representa o tamanho em bits de um padrão de entrada, M representa o número de neurônios da rede e N representa o tamanho da tupla para cada neurônio da rede.

2.2.5.2 Fase de teste da rede WiSARD

Ao fim da etapa de treino, é esperado que a rede tenha aprendido a identificar o problema e seja capaz de generalizá-lo para padrões não conhecidos. Durante a fase de teste da rede WiSARD, um padrão não conhecido é mostrado à rede, mantendo a mesma subdivisão do padrão em tuplas usado durante o treinamento. Então, cada tupla do padrão desconhecido irá endereçar os k neurônios da rede, sendo que cada um vai apresentar uma saída baseado no valor da memória RAM endereçada. Caso o neurônio identifique a tupla daquele padrão como pertencente à classe treinada, ele irá produzir uma saída com nível lógico '1'. E caso não identifique a tupla como pertencente à classe, a saída terá nível lógico '0'.

Para saber se o padrão de entrada foi identificado pela rede, o discriminador contará, quantos dos k neurônios apresentaram saída de nível lógico '1', que nesse caso são os que identificaram a rede como pertencente à classe, essa contagem é feita através de um somador, que é localizado no final dos neurônios, como pode ser observado na Figura 2.6 . Se a quantidade de neurônios com saída igual a '1', for maior que um limiar predeterminado, significa que o padrão desconhecido é pertencente à classe identificada pela rede. Caso contrário, número de neurônios com saída '1' menor que o limiar predeterminado, então o padrão desconhecido não pertence à classe identificada pela rede.

Para a rede WiSARD com múltiplas camadas paralelas, um padrão desconhecido será apresentado igualitariamente para todas as camadas da rede. O discriminador irá contar a quantidade

de neurônios com saída de nível lógico ‘1’ para as tuplas do padrão a ser reconhecido. Então as respostas de cada um dos discriminadores é analisada e comparada, de forma que a que apresentar o maior número de neurônios de saída ‘1’ é a escolhida. Logo a classe do padrão desconhecido será determinada como pertencente à classe do discriminador com a maior quantidade de neurônios com saída ‘1’. Essa configuração é chamada de multi-discriminador (21); (22).

3 O SISTEMA INTELIGENTE

Nesse Capítulo é apresentada a proposta de sistema inteligente para detecção automática e prevenção de ataques do tipo *phishing*. Será apresentada a metodologia para o desenvolvimento deste trabalho e também os aspectos arquiteturais do sistema inteligente e de sua Rede Neural Artificial subjacente.

3.1 CARACTERÍSTICAS DE URLS CONTAMINADAS POR *PHISHING*

A classificação de um sítio e sua respectiva URL como maliciosos (*phishing*) ou confiáveis não é uma tarefa trivial, em especial por conta da subjetividade que também existe na análise de um determinado sítio. Ou seja, um ataque *phishing* bem feito pode levar até mesmo usuários experientes a seguir uma URL "falsa" e informar dados sensíveis no respectivo sítio que o próprio usuário, visualmente, considera legítimo, ainda que ele não seja. Por outro lado, estudos recentes (5), (23), (7), (24) indicam que é possível realizar a análise de *phishing* através de características da URL e sua respectiva página. Logo, neste trabalho, objetiva-se, a caracterização do ataque por meio da análise da URL da página e de parte do seu conteúdo. Informações baseadas na análise sintática (7), do cabeçalho http (23) e provenientes do WhoIs fornecem informações importantes para classificação dos links. Por exemplo, endereços que contenham muitos digitos (números), palavras-chave como "security"e "pay", além de muitos caracteres especiais, têm maior probabilidade de serem maliciosos. No Capítulo 4 é demonstrada a relevância das principais características por meio de análise visual utilizando o *Weka*. Determinou-se, assim, algumas características de URLs para identificação de páginas contaminadas por *phishing*. Para melhor visualização destas, a tabela abaixo lista as *features* extraídas dos endereços das páginas e utilizadas na tomada de decisão dos algoritmos.

As palavras consideradas suspeitas contabilizadas na feature nomeada "NUM_KEYWORDS" da Tabela 3.1 foram: "confirm", "account", "banking", "secure", "ebayisapi", "webscr", "login", "signin", "pay", "free", "senha", "security", pois elas, frequentemente, estão presentes no endereço de páginas maliciosas conforme (5). Por outro lado as features "WHOIS_REGDATE" e "WHOIS_UPDATE" foram obtidas através do *WhoIs*. Este é um protocolo TCP específico para consultar informações de contato e DNS sobre um nome de domínio, endereço IP ou um sistema autônomo. Mais informações sobre cada *feature* encontram-se na Seção 3.2.2.1.

Tabela 3.1: Características (*features*) extraídas de URLs.

Origem	Nome	Descrição
Análise sintática	URL_LGTH	Quantidade de caracteres antes da primeira barra
	NUM_SLASHES	Quantidade de barras
	NUM_DOTS	Quantidade de pontos
	NUM_ESP_CAR	Quantidade de caracteres especiais subtraído as barras e pontos
	NUM_KEYWORDS	Quantidade de palavras consideradas suspeitas
	DOM_LGTH	Tamanho do domínio
	PRE_DOM_LGTH	Quantidade de caracteres antes do domínio
	NUM_DIR	Quantidade de diretórios
	LGTH_DIR	Tamanho do último subdiretório
	BIG_SUBDIR	Tamanho do maior diretório
	NUM_NUM_PRE_DOM	Quantidade de números antes do domínio
	NUM_NUM_POS_DOM	Quantidade de números após o domínio
	NUM_NUM_DOM	Quantidade de números no domínio
WhoIs	WHOIS_REGDATE	Tempo decorrido, em semanas, desde o registro do domínio
	WHOIS_UPDATE	Tempo decorrido, em semana, desde a última atualização da página
Cabeçalho HTTP	CONTENT_LENGTH	Tamanho da página em bytes

3.2 ARQUITETURA DO SISTEMA

O sistema inteligente é composto por vários módulos, apresentados no fluxograma da Figura 3.1. Em síntese, são quatro etapas principais que, juntas, formam o sistema. A primeira, detalhada na Seção 3.2.1, é responsável pela obtenção das URLs maliciosas e não-maliciosas. Trata-se de um *webcrawler* capaz de buscar os vários links (URLs) existentes a partir de um domínio inicial. Posteriormente, uma outra etapa gera o *dataset* por meio da extração de características mencionadas na Tabela 3.1 via análise sintática, *WhoIs* e HTTP. Esta etapa é explanada em detalhes na Seção 3.2.2. A terceira etapa consiste na análise e validação do *dataset* por meio do *Weka* para uso do conjunto de dados no treino e teste da rede neural sem peso. Por fim, a última etapa consiste no uso da rede neural WiSARD em si, considerando a representação do *dataset* em binário e o resultado correspondente da classificação.

3.2.1 Obtenção de URLs

Para o melhor desempenho e acurácia de algoritmos baseados em decisões é necessário que os mesmos sejam treinados de forma a conhecer o maior número possível de dados. Por isso o *dataset* de treino deve ser diversificado, contendo endereçoes maliciosos e não maliciosos. Por outro lado a lista utilizada para gerar o *dataset* de teste deve ser diferente do rol de treino, mas também deve ser diversificada, contendo URLs de páginas maliciosas e não maliciosas, diferentes

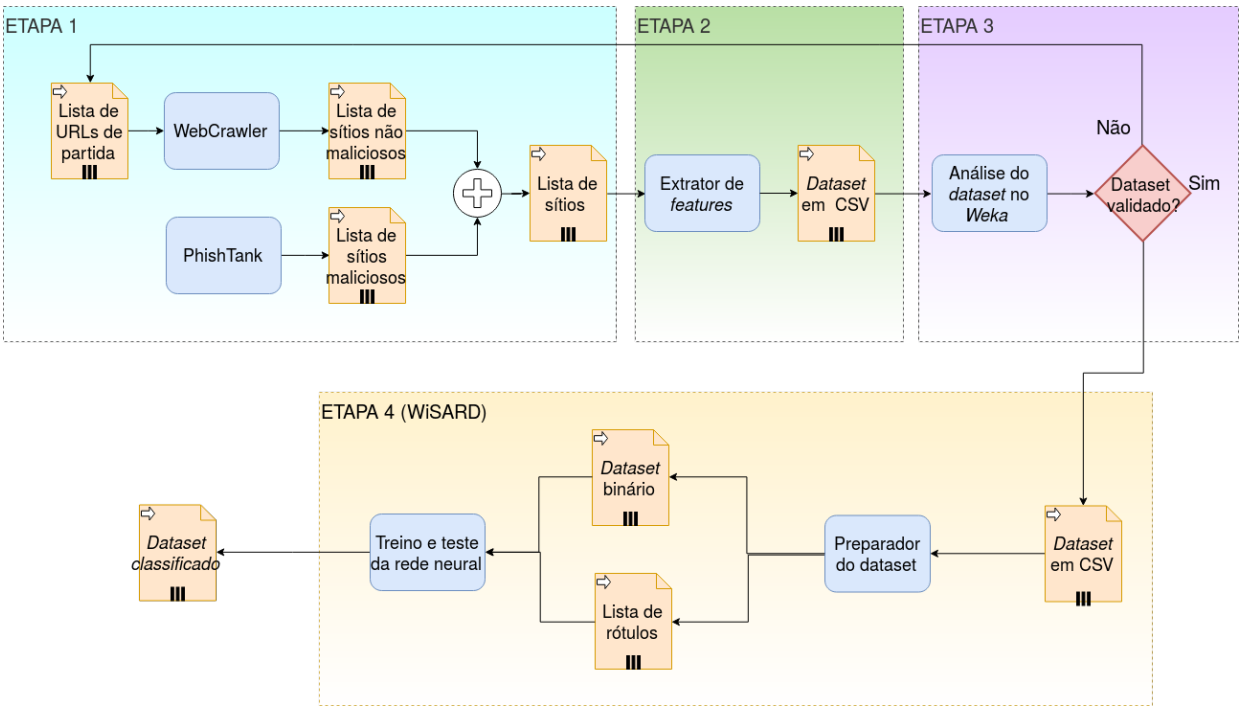


Figura 3.1: Representação diagramática do sistema inteligente *WiSARD*

daquelas usadas para treino. Dessa forma é possível verificar se o algoritmo é capaz de classificar páginas inéditas e averiguar a qualidade do *dataset*.

3.2.1.1 Obtenção de URLs não maliciosas

Para obter as URLs não maliciosas foi desenvolvido um programa *webcrawler* em Python. Esta ferramenta permite a obtenção de um número muito grande de endereços únicos a depender do processamento e do tempo. O programa busca as URLs referenciadas na página por meio do atributo "href" dentro da *tag* "$< a >$" do código HTML, como exemplifica a Figura 3.2. Este endereço inicial deve garantir a não obtenção de links maliciosos. Iniciou-se, para tanto, a busca por meio de uma página sabidamente não maliciosa, doravante referida como "URL de partida".

```
<a href="http://www.extrawatch.com">
```

Figura 3.2: Exemplo de site referenciado encontrado na página http://unb.br/

Todos os endereços encontrados na página são adicionados ao fim de uma lista na memória. Estes poderão ser também visitados pelo *webcrawler* a depender da definição de uma variável limite, chamada de *NumeroSaltos*. Tal variável define até quantas páginas serão inspecionadas pelo *crawler* seguindo a ordem da lista gerada por ele. Fica claro que definido um número grande de saltos, o *webcrawler* é capaz de obter uma quantidade maior de URLs não maliciosas (vide Capítulo 4) e, garantindo a não repetição de endereços na lista, gerar o rol desejado. Entretanto, decorrem alguns problemas sanáveis nesta implementação do *crawler*.

Por exemplo, caso a URL de partida não possua pelo menos um link referenciado por meio do "<a> href", a execução do *webcrawler* será interrompida, pois a lista estará vazia. Como solução foi garantida a presença de pelo menos um link referenciado dentro da URL de partida mediante teste preliminar com a variável *NumeroSaltos* definida como 1. Outro ponto de importante destaque é a lentidão na execução do *webcrawler* quando ocorrem repetidos *TimeOuts*: situação em que a página apontada pelo URL não se encontra disponível. Todavia, não é possível solucionar este ponto, pois a indisponibilidade de páginas é característica inerente à internet.

O maior problema, no entanto, é a piora da qualidade do *dataset* decorrente da busca em profundidade realizada pelo *crawler* quando o número de saltos é muito alto. Para que os algoritmos de aprendizado sejam capazes de classificar corretamente as páginas é necessário o treinamento com links das mais diversas origens. É exatamente o oposto disso que ocorre quando realizada a busca em profundidade. Nesta será gerada uma grande lista, porém a maior parte dela será de endereços parecidos.

```
1  https://stackoverflow.com/
2  https://stackoverflow.com
3  https://stackexchange.com/users/?tab=inbox
4  https://stackexchange.com/users/?tab=reputation
5  https://stackexchange.com
6  https://stackoverflow.com/users/login?ssrc=head&returnurl=https%3a%2f%2fstackoverflow.com%2f
7  https://stackoverflow.com/users/signup?ssrc=head&returnurl=%2fusers%2fstory%2fcurrent
8  https://stackoverflow.com/help
9  https://chat.stackoverflow.com
10 https://meta.stackoverflow.com
11 https://stackoverflow.com/users/signup?ssrc=site_switcher&returnurl=%2fusers%2fstory%2fcurrent
12 https://stackoverflow.com/users/login?ssrc=site_switcher&returnurl=https%3a%2f%2fstackoverflow.com%2f
13 https://stackexchange.com/sites
14 https://stackoverflow.blog
15 https://stackoverflow.com/tour
16 https://stackoverflow.com/company/about
17 https://www.stackoverflowbusiness.com/?ref=topbar_help
18 https://stackoverflow.com/legal/cookie-policy
19 https://stackoverflow.com/legal/privacy-policy
20 https://stackoverflow.com/legal/terms-of-service/public
```

Figura 3.3: Parte de lista gerada com URL de partida https://stackoverflow.com/

No exemplo da Figura 3.3 foi executado o *webcrawler* com a URL de partida do popular sítio *stackoverflow*. Neste recorte, as vinte primeiras páginas pertencem ao próprio *stackoverflow*, porém o primeiro link de outro domínio é o de número 495 da lista. Ou seja, para o *crawler* visitar uma página de outro domínio o número de saltos deveria ser pelo menos 495. Como solução adotou-se a busca horizontal ao executar-se o *webcrawler* em paralelo utilizando o multiprocessamento implementado em Python, através do módulo *Multiprocessing*. Dessa forma é possível determinar diversas URLs de partida, abrangendo um maior número de domínios e gerando, assim, um *dataset* mais diversificado e apropriado para o treino. Neste caso o número de saltos deve ser pequeno, pois não é desejada a busca vertical.

Para gerar a lista de links foram escolhidas 75 URLs de diferentes países e conteúdos, como : "https://www.bestbuy.com/", "https://www.justice.gov/"e "https://www.cifraclub.com.br/". O número de saltos foi definido como cinco. Os Algoritmos 2 e 3 representam as classes mais importantes da implementação do *webcrawler*. O primeiro ilustra a classe principal, na qual é

iniciada os 75 processos responsáveis pela busca horizontal.

Algoritmo 2: Pseudo-código da implementação da classe principal do *webcrawler*

```
   Entrada: Lista de URLs de partida
   Saída: Lista de sítios não maliciosos
1  início
2      Definir o número de saltos como "5";
3      para cada URL na lista de sítios não maliciosos faça
4          Criar processo paralelo que executará objeto da classe spider para presente URL;
5          Iniciar processo paralelo criado;
6      fim
7      Esperar o final da execução de todos os processos;
8      Juntar todos os arquivos de saída no mesmo arquivo txt;
9  fim
```

A classe *spider*, representada no Algoritmo 3, é capaz de buscar URLs na página por meio do atributo "<a> href" no código HTML e, assim, gerar uma lista com diversos endereços. Também é essa classe que acessa as URLs dessa lista até o máximo de "5" saltos (número definido para busca horizontal). O código do *webcrawler* em Python encontra-se no Anexo 6.1.1.

Algoritmo 3: Pseudo código da implementação da classe "spider" *webcrawler*

```
1  Classe spider, responsável por analisar páginas:
   Entrada: URL proveniente da classe principal
   Saída: Arquivo TXT com lista de URLs encontradas
2  início
3      Adicionar URL proveniente da classe principal na lista local de URLs;
4      enquanto acessadas menos de 5 páginas faça
5          Realizar acesso de página na lista local de URLs;
6          Buscar URLs na página por meio do atributo "<a> href" no código HTML;
7          Adicionar todos os links encontrados ao final da lista local de URLs;
8          Acessar próximo sítio da lista local de URLs;
9      fim
10     Escrever arquivo TXT com a lista local de URLs;
11 fim
```

3.2.1.2 Obtenção de URLs maliciosas

A obtenção de URLs maliciosas deve ser realizada com extrema cautela, pois é por meio destas que o sistema aprenderá as características de um endereço malicioso. A princípio, tentou-se construir esta lista por meio do *webcrawler* com início em páginas já classificadas como *phishing* por outras ferramentas. Entretanto não é possível garantir que os links obtidos pelo *crawler* serão de páginas maliciosas, pois estas podem referenciar outras não maliciosas e, consequentemente, ocasionar o treinamento incorreto da rede neural.

Para garantir a má índole das URLs, optou-se por utilizar um conjunto classificado por outro meio já consolidado: o *PhishTank* (12). A plataforma que possui ferramentas para desenvolvedo-

res mantém uma lista de páginas contaminadas por *Phishing*. Este rol é colaborativo, ou seja, é mantido pela comunidade que possui as funções de registrar, verificar, rastrear e compartilhar informações sobre páginas infectadas. Para assegurar a qualidade do *dataset*, o catálogo escolhido possui apenas links cadastrados por um usuário e confirmada a presença do ataque por outrem. A lista de URLs maliciosas utilizadas na contrução do *dataset* de treino possui 5426 links e o de teste 997 endereços obtidos no Phishtank em consultas realizadas em dias diferentes e garantida a não repetição entre as listas. O diagrama da Figura 3.4 representa como deu-se a obtenção da lista de sítios utilizados para construção do *dataset*.

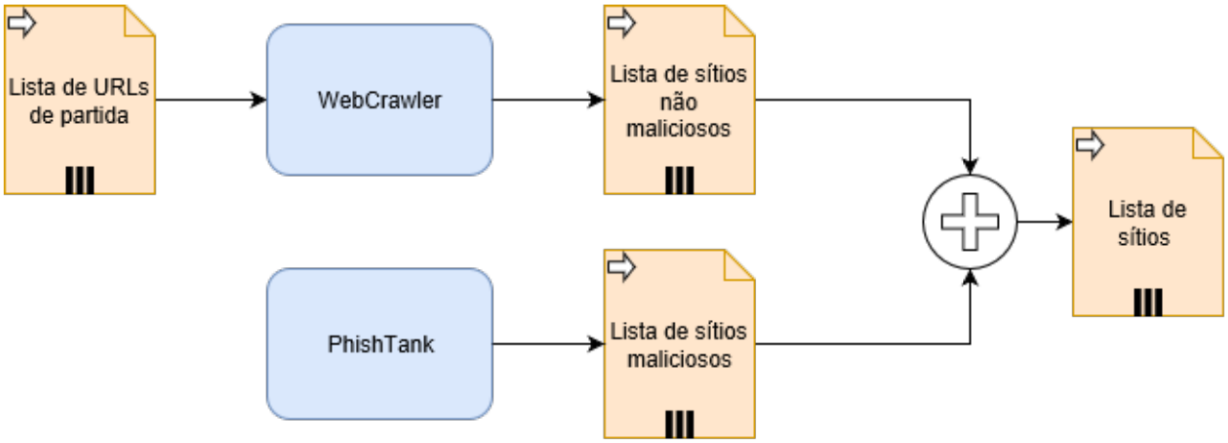


Figura 3.4: Representação diagramática da obtenção da lista de sítios

A Tabela 3.2 resume a lista de URLs utilizada para treino e para teste da rede neural. Tal lista representa o *dataset* de características (*features*) coletadas a respeito de cada uma das URLs e suas respectivas páginas, buscadas no *phishtank* e pelo *webcrawler*.

Tabela 3.2: Resumo da lista de URLs utilizados para gerar datasets

Destinação do *dataset*	Número de URLs maliciosas	Número de URLs não maliciosas	Total
Treino	5426	6277	11703
Teste	997	1026	2023

3.2.2 Construção do *dataset*

Após o processo para obtenção das URLs é necessária a extração de características de cada uma delas, denominadas *features*. Para isso realizou-se a análise sintática e a extração de dados proveniente dos protocolos *WhoIS* e HTTP. Posteriormente o *dataset* é avaliado manualmente e são realizadas eventuais alterações para melhor desempenho dos algoritmos de classificação. Estes são os procedimentos detalhados nessa Seção.

3.2.2.1 Extração de features das URLs

A extração das *features* foi realizada por meio de código em Python em três etapas: análise sintática, *WhoIs* e HTTP. A entrada é a lista de sites gerada via *webcrawler* e *PhishTank*. A saída é um arquivo CSV (*Comma-Separated Values*) contendo todas as características das URLs com adicional de um campo informando se a página é maliciosa. Esta última informação é usada durante o treino da Rede Neural WiSARD, ou seja, para selecionar qual classificador (malicioso ou não-malicioso) será treinado. A análise sintática da URL é, por sua vez, responsável por extrair as características presentes no próprio texto da URL, sem a necessidade de consulta à meio externo. Neste sentido foram extraídas sintaticamente as seguintes informações:

1. *URL_LGTH*: representa a quantidade de caracteres presente na URL antes da primeira barra.
2. *NUM_SLASHES*: representa a quantidade de barras "/" presentes no link. Descontam-se as barras presentes em "http://" e "https://".
3. *NUM_DOT*: representa o número de pontos presentes no link. Juntamente com as features responsáveis por representar a quantidade de números é capaz de identificar se o domínio é um IP, característica comum entre sites maliciosos.
4. *NUM_ESP_CAR*: representa a quantidade de caracteres especiais presentes no link. Descontam-se o número de barras e pontos.
5. *NUM_KEYWORDS*: representa a quantidade de palavras chaves presentes no link. São elas: "confirm", "account", "banking", "secure", "ebayisapi", "webscr", "login", "signin", "pay", "free', "senha" e "security".
6. *DOM_LGTH*: representa a quantidade de caracteres presentes no domínio.
7. *PRE_DOM_LGTH*: representa a quantidade de caracteres presentes antes do domínio.
8. *NUM_DIR*: representa a quantidade de diretórios presentes no link.
9. *LGTH_DIR*: representa a quantidade de caracteres presentes no último subdiretório.
10. *BIG_SUBDIR*: representa a quantidade de caracteres presentes no maior subdiretório.
11. *NUM_NUM_PRE_DOM*: representa a quantidade de números presentes antes do domínio.
12. *NUM_NUM_POS_DOM*: representa a quantidade de números presentes nos subdiretórios.
13. *NUM_NUM_DOM*: representa a quantidade de números presentes no domínio.

A análise utilizando o protocolo *WhoIs* elenca duas das principais features do *dataset*. Essas foram obtidas por meio da biblioteca "whois" implementada em Python.

1. *WHO_IS_REGDATE*: representa o tempo decorrido, em semanas, desde o registro do domínio na entidade responsável. O protocolo *whois* informa a data de criação, para calcular a idade, em semanas, foi utilizado o *Excel*.

2. *WHO_IS_UPDATE*: representa o tempo decorrido, em semanas, desde a última atualização da página. O protocolo *whois* informa a data da última atualização, para calcular a idade, em semanas, foi utilizado o *Excel*.

O cabeçalho do protocolo HTTP foi utilizado apenas para extração da *feature* que representa o tamanho da página acessada "CONTENT_LENGTH". A implementação deu-se por meio da função "urlopen" presente na biblioteca Python "urllib.request".

1. *CONTENT_LENGTH*: representa o tamanho da página acessada em *bytes*.

Por fim é incluída a informação "*IS_MALICIOUS*" responsável pela seleção do classificador da rede neural. Essa é obtida de acordo com a origem da entrada: se foi do *webcrawler* é "0"(indicando ser não malicioso), caso contrário "1"(proveniente do Phistank e, portanto, malicioso). Observe que esta informação não representa uma característica em si e, portanto, não será usada na formação do *dataset* usado para treino/teste da rede neural. Ela apenas será usada durante o treino para indicar qual classificador da rede neural deverá ser treinado, pois afinal de contas ela já representa o resultado que se deseja obter naturalmente da rede neural a partir de qualquer conjunto de URLs novas.

Tabela 3.3: Características presentes no arquivo CSV de saída do extrator de *features*

Origem	Tipo	Nome	Descrição
Análise sintática	Inteiro	NUM_SLASHES	Quantidade de barras
	Inteiro	NUM_DOTS	Quantidade de pontos
	Inteiro	NUM_ESP_CAR	Quantidade de caracteres especiais subtraído as barras e pontos
	Inteiro	NUM_KEYWORDS	Quantidade de palavras consideradas suspeitas
	Inteiro	DOM_LGTH	Tamanho do domínio
	Inteiro	PRE_DOM_LGTH	Quantidade de caracteres antes do domínio
	Inteiro	NUM_DIR	Quantidade de diretórios
	Inteiro	LGTH_DIR	Tamanho do último subdiretório
	Inteiro	BIG_SUBDIR	Tamanho do maior diretório
	Inteiro	NUM_NUM_PRE_DOM	Quantidade de números antes do domínio
	Inteiro	NUM_NUM_POS_DOM	Quantidade de números após o domínio
	Inteiro	NUM_NUM_DOM	Quantidade de números no domínio
WhoIs	Inteiro	WHOIS_REGDATE	Tempo decorrido, em semanas, desde o registro do domínio
	Inteiro	WHOIS_UPDATE	Tempo decorrido, em semana, desde a última atualização da página
Cabeçalho	Inteiro	HTTP	Tamanho da página em bytes
Arquivo de entrada	Binário	IS_MALICIOUS	Reprenta URL maliciosa caso "1" e não maliciosa se "0"

O diagrama da Figura 3.5 esquematiza o extrator de *features*.

Figura 3.5: Representação diagramática do extrator de *features*

3.2.3 Análise do *dataset*

Com o arquivo CSV gerado resta aferir os dados para checar a qualidade do *dataset* e classificar as URLs de teste. Para isso utilizou-se duas ferramentas: o *Weka* e a rede neural sem peso *WiSARD*.

3.2.3.1 Weka

Para checar a qualidade do *dataset* iniciou-se os testes por meio do software *Weka* na sua versão 3.8, a mais recente, pois nesta estão disponíveis diversos algoritmos de aprendizado de máquina (ex: clusterização) já consolidados. Optou-se por usar o J48 e o RandomTree para validação dos dados obtidos por meio do *webcrawler* e daqueles advindos do *PhishTank*. Além disso realizou-se a gráfica correlacional entre as *features* consideradas de maior relevância.

O *J48* foi utilizado para gerar uma árvore de decisão baseada no conjunto de dados de treinamento, sendo este modelo usado para classificar as instâncias do conjunto de teste. Esse algoritmo se mostra adequado para os procedimentos envolvendo as variáveis qualitativas contínuas e discretas presentes na base de dados do projeto. O *RandomTree* foi utilizado pois faz uso de um modelo de decisões em forma de árvore e suas possíveis consequências, incluindo resultados de eventos aleatórios, custos de recursos e utilidade. As árvores de decisão são mais interpretáveis do que outros classificadores, como redes neurais e máquinas de vetores de suporte, porque combinam questões simples sobre os dados de maneira compreensível. Incluiu-se na análise de resultados (Capítulo 4) uma árvore gerada pelo *RandomTree* para melhor entendimento deste conceito.

Para os testes com os algoritmos *J48* e *RandomTree* é necessário realizar a divisão do *dataset*, sendo parte destinada para treino e outro para teste. O *Weka* possui dois métodos para realizar tal tarefa: *cross-validation* e o *percentage split*. Optou-se pela utilização de ambos os métodos para validação do *dataset*. O *cross-validation* foi usado por ser um método padrão na análise de desempenho de algoritmos, modelos de aprendizado de máquina e reconhecimento de padrões. Ele permite que o *Weka* construa um modelo baseado em subconjuntos dos dados fornecidos para então calcular sua média e criar um modelo final. No *percentage split*, o *software* toma um subconjunto percentual dos dados fornecidos para construir um modelo final. Nesse caso usou-se 70% do total para treino e 30% para teste.

A análise do *dataset* por meio dos algoritmos *J48* e *RandomTree* deu-se em dois momentos.

No primeiro realizou-se o teste de *cross-validation* utilizando 10 pastas para ambos algoritmos e comparados os resultados. Em sequência dividiu-se o *dataset* por meio do *percentage split*, sendo 70% utilizado para treino e 30% para teste. Os resultados analisados foram: instâncias classificadas corretamente, instâncias classificados incorretamente, taxa de verdadeiro positivo e a taxa de falso positivo. Todos são detalhados na Seção 4.2.1. Para realizar a análise correlacional das *features*, identificou-se quais as mais relevantes por meio dos resultados obtidos pelos algoritmos e pela árvore de decisão gerada pelo *RandomTree*. Dessa forma foram escolhidas as *features*: "NUM_NUM_POS_DOM", "WHOIS_REGDATE", "DOM_LGTH" e "NUM_KEYWORDS". Os gráficos são exibidos na Seção 4.2.2.

3.2.3.2 Uso da rede neural sem peso WISARD

Apesar do *Weka* fornecer dados importantes para verificar a qualidade do *dataset* não é possível a sua aplicação em um sistema integrado. Por exemplo, não é possível saber quais as URLs foram classificas de forma incorreta ou, até mesmo, testar um endereço avulso não incluído previamente no *dataset* de teste. Utilizou-se, para possibilitar a integração com o sistema, a rede neural sem peso *WiSARD* (2). Para tanto, utilizou-se uma implementação da rede em Python (25) num ambiente Windows 10 Pro. São duas as entradas aplicadas à rede neural. A primeira são as *features*, em formato binário, organizadas em uma lista de bits, conforme mostra a Figura 3.6. A segunda são os rótulos, como mostra a Figura 3.7, que devem estar na mesma ordem das *features* previamente fornecidas. Por exemplo, caso os rótulos da Figura 3.7 refiram-se as entradas da Figura 3.6, as duas primeiras entradas serão rotuladas como "Malicioso" e as últimas como "Não-Malicioso".

```
Features = [
            [1,1,1,1,1,1,0,0],
            [1,1,1,1,1,0,1,0],
            [0,0,0,1,0,0,1,0],
            [1,0,1,0,0,0,0,0],
           ]
```

Figura 3.6: Exemplo de entrada binária aceita pela rede neural utilizada

```
Rotulos = [
            "Malicioso"
            "Malicioso"
            "Não-Malicioso"
            "Não-Malicioso"
           ]
```

Figura 3.7: Exemplo de rótulos aceitos pela rede neural utilizada

As entradas devem respeitar algumas características. Como no exemplo da Figura 3.6, as *features* inseridas devem ser binárias e todos os membros da lista devem conter exatamente o mesmo

número de componentes. É necessário, ainda, que o tamanho da lista de rótulos seja igual ao da lista de *features*. Surge então um problema de representação binária de praticamente todas as características levantadas. Como representar uma *feature* literal (ex: data) em um conjunto de bits? Conforme Tabela 3.3 as features são do tipo numérica (inteiro), literal (caracteres) ou binário. Deve-se, então, realizar a representação binária de todas aquelas não binárias: comprimento do URL, número de barras, números de pontos, etc.

Logo, realizou-se a representação binária de cada *feature* separadamente concatenando os valores convertidos em um mesmo elemento da lista, a fim de formar a lista de bits para entrada da rede neural. Supondo, por exemplo, um *dataset* com apenas duas características, uma de valor 7 e outra de valor 8, suas respectivas representações binárias são 0111 e 1000. Portanto, a lista correspondente à URL com tais características seria [0,1,1,1,1,0,0,0]. No entanto, a representação numérica binária (tradicional) destes números leva a perda de precisão da rede, pois note que 7 e 8 são valores muitos próximos em sua representação decimal, porém muito distantes (diferentes) em sua representação binária. Portanto, o classificador pode julgar que os mesmos devem ser classificados em grupos diferentes, o que possivelmente não é verdade.

Sendo assim, a fim de obter melhor acurácia na classificação das URLs foi experimentado dois esquemas de representação binária. No primeiro teste os valores inteiros foram convertidas em binários por meio da representação numérica binária tradicional, já presente na linguagem Python. Percebeu-se que um dos principais problemas desta representação é a imprevisibilidade do número de bits necessários pra representação do número inteiro já que todos os membros da lista devem possuir o mesmo tamanho. Essa questão foi solucionada ao verificar-se quantos bits seriam necessários para a representação em binário do maior elemento de cada *feature*. Dessa forma, aquelas menores foram preenchidas com o valor "0" adicionado a esquerda até sua representação atingir o mesmo tamanho da maior. Por exemplo, se o maior valor da *feature* "NUM_KEYWORDS" for 4, representado por três bits (100), e outra URL possuir apenas uma palavra chave, representada por apenas um bit (1), a última deverá ser acrescida de dois zeros a esquerda para adquirir o mesmo tamanho daquela. Sendo assim a representação do número 1 será 001.

O segundo esquema de binarização proposto é baseado na criação de oito faixas de representação. Ou seja, divide-se o maior valor encontrado em cada *feature* por oito a fim de dividi-las em faixas de mesmo tamanho. Cada uma desses intervalos será representado por combinações binárias previamente definidas. A rede neural deve ser capaz de identificar o padrão dessa representação, por isso escolheu-se os padrões da Tabela 3.4.

Por exemplo, supondo uma *feature* na qual o maior valor seja 80 a representação será realizada conforme a Tabela 3.5

Na representação binária por faixas da *feature* "CONTENT_LENGTH" aplicou-se a função do logaritmo natural em todos seus componente para diminuir o tamanho das faixas. Como essa representa a quantidade de bytes da página acessada, a diferença entre o maior e menor valor é muito grande. A aplicação dessa função aproxima os valores aumentando a significância de cada

Tabela 3.4: Representação binária por faixas

Faixa	Representação binária
0 =<Valor feature <Valor maior feature /8	11001100
Valor maior feature /8 <= Valor feature <Valor maior feature*2 /8	10101010
Valor maior feature *2/8 <= Valor feature <Valor maior feature*3/8	11110000
Valor maior feature *3/8 <= Valor feature <Valor maior feature*4/8	11000011
Valor maior feature *4/8 <= Valor feature <Valor maior feature*5/8	00110011
Valor maior feature *5/8 <= Valor feature <Valor maior feature*6/8	01010101
Valor maior feature *6/8 <=Valor feature <Valor maior feature*7/8	00001111
Valor maior feature *7/8 <= Valor feature <=Valor maior feature	00111100

Tabela 3.5: Exemplo da representação binária por faixas para feature com maior valor 80

Faixa	Representação binária
0 =<Valor feature <10	11001100
10 <= Valor feature <20	10101010
20 <= Valor feature <30	11110000
30 <= Valor feature <40	11000011
40 <= Valor feature <50	00110011
50 <= Valor feature <60	01010101
60 <=Valor feature <70	00001111
70 <= Valor feature <= 80	00111100

faixa. As Equações 3.1 e 3.2 representam as operações realizadas.

$$ValorReduzido = ln(ValorFeature) \tag{3.1}$$

$$MaiorValorReduzido = ln(MaiorValorFeature) \tag{3.2}$$

A construção da lista de rótulos deu-se de maneira simples. Uma função checa se *feature* "IS_MALICIOUS" de cada um dos componentes da lista é "0" ou "1". Acrescentando as string "Not Malicious", no primeiro caso, ou "Malicious", no segundo, ao fim do rol de rótulos. Dessa forma garante-se o mesmo tamanho de ambas as listas e a correta classificação de cada endereço. O módulo responsável pela representação binária do *dataset* e por gerar a lista de rótulos recebeu o nome de "Preparador do dataset", pois esse é responsável por preparar os dados para serem utilizados pela rede neural.

Realizada a representação binária procedeu-se o treinamento da rede e o teste, conforme mostra a Figura 3.8. Essas operações foram realizadas pelo mesmo programa. A rede *WiSARD*, utilizada por meio da biblioteca "wisardpkg", necessita de três parâmetros além das *features* e rótulos. São eles: *addressSize*, responsável por definir o tamanho da tupla da rede, *ignoreZero*, que indicará se a rede deve ignorar as entradas definidas como "0" e verbose, o qual indica se o progresso de treino e teste será mostrado na tela. Definiu-se, em todos os testes, os dois últimos parâmetros como "False". Deseja-se, pois, que os "0" não sejam ignorados, já que apresentam

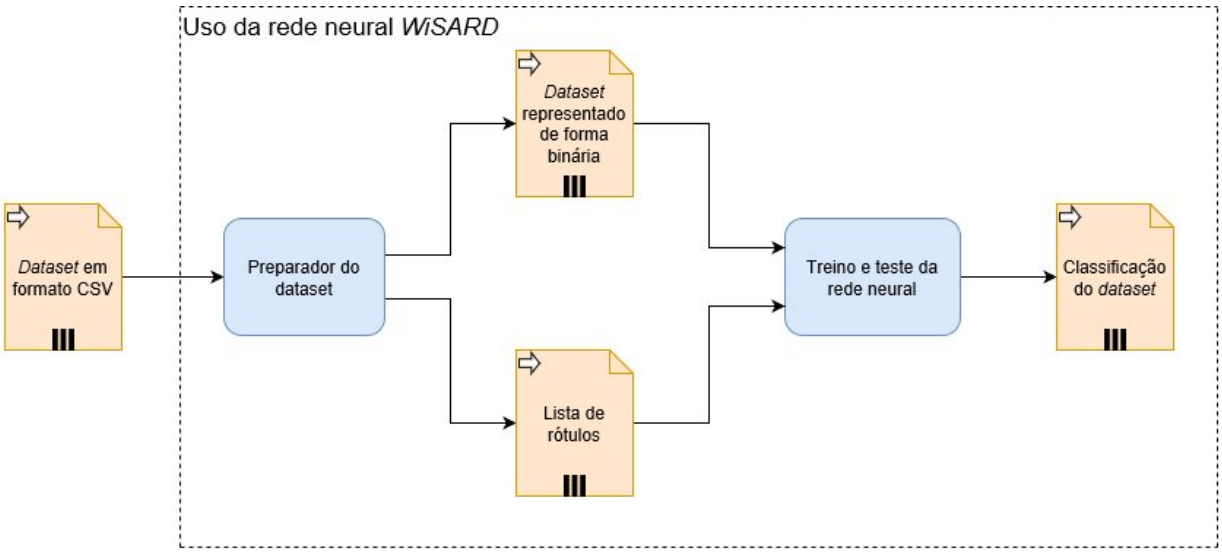


Figura 3.8: Fluxograma de uso da rede neural *WiSARD*

relevância no *dataset*, e que não seja mostrado o progresso do treino e teste devido a lentidão causada pela função. Por outro lado, o parâmetro *addressSize*, que pode variar entre 2 e 64, têm impacto direto na eficiência da rede e, efetuou-se, portanto, testes para definir qual valor apresenta melhor assertividade.

As redes neurais são algoritmos não-determinísticos, ou seja, nem sempre produzirão a mesma saída dada uma entrada. Portanto, para aferir a assertividade da rede realizou-se dez treinos seguidos de testes, utilizando o mesmo *dataset* para cada etapa. Sendo assim é possível a definição da assertividade da rede de duas formas: o primeiro resultado do teste (representado pela média das 10 tentativas) ou a maior taxa de acerto obtida. A escolha dessa é detalhada no Capítulo 4. Para a definição do *adressSize*, variou-se o entre 2 e 64, todos os valores possíveis, em 10 treinos e testes a fim de checar qual apresenta melhor desempenho. Este teste foi executado para os dois esquemas de representação binária propostos para identificar qual o mais assertivo para implementação no sistema inteligente.

O Algoritmo 4 explicita os principais passos realizado pelo sistema de detecção de *phishing*, no que diz respeito ao uso da rede neural, e deve ser executado uma vez para cada esquema de representação binária proposto. Ao final dos testes deve-se decidir qual esquema de representação binária e valor de “adressSize” devem ser utilizados para melhor desempenho do sistema. Além disso deve ser definido se é possível a utilização da maior taxa de acerto em 10 tentativas de teste e treino.

Algoritmo 4: Uso da rede neural

Entrada: Arquivo CSV contendo *dataset* de treino e arquivo CSV contendo *dataset* de teste
Saída: Taxa média de acerto, maior acerto e velocidade de execução para 10 tentativas com todos adressSize possíveis

```
1  início
2      Representar de forma binária o dataset de treino;
3      Reapresentar de forma binária o dataset de teste;
4      Gerar a lista de rótulos, confome algoritmo 2;
5      Definir verbose e ignoreZero como "false";
6      para cada k adressSize entre 2 e 64 faça
7          para cada uma das 10 tentativas faça
8              Criar objeto WiSARD com adressSize k;
9              Treinar rede neural com o dataset de treino e lista de rótulos;
10             Classificar o dataset de teste;
11             Contabilizar o número de acertos e erros conforme algoritmo 3; se taxa de acerto for a
                maior até o respectivo teste então
12                 Salvar taxa de acerto;
13             fim
14         fim
15         Calcular e salvar a taxa média de acertos;
16         Salvar o tempo gasto para execução das 10 tentativas;
17     fim
18     Exibir a taxa média para os 10 testes e treinos de cada adressSize;
19     Exibir a taxa máxima para os 10 testes e treinos de cada adressSize;
20     Exibir o tempo gasto para os 10 testes e treinos de cada adressSize;
21 fim
```

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Este Capítulo apresenta uma análise detalhada de cada programa que forma o sistema inteligente, incluindo resultados de tempo de execução e acurácia da rede neural.

4.1 WEBCRAWLER

Com objetivo de realizar uma busca horizontal e, assim, gerar um *dataset* diversificado, o *webcrawler* foi implementado com 75 URLs de partida e com o parâmetro *NumeroSaltos* definido como 5. A princípio desejava-se obter, pelo menos, 5426 endereços. Esse, pois, era o número de URLs maliciosas disponíveis no *PhishTank*. Os primeiros testes foram implementados de forma sequencial, ou seja, a verificação da segunda URL de partida ocorreria somente após o término dos cinco saltos da primeira, e assim sucessivamente. O processo dessa forma é muito lento e ainda é agravado caso páginas visitadas sofram *TimeOut*. Como solução adotou-se o multiprocessamento para realizar a inspeção de cada uma das URLs de partidas. Ou seja, foram iniciados 75 processos diferentes, sendo que cada um realizou a busca na URL de partida e nos 4 saltos restantes.

Os resultados do *webcrawler* foram analisados quanto ao tempo de execução do programa e quantidade de links não maliciosos obtidos. Decidiu-se por não avaliar o estresse do sistema; não ocorreram, pois, limite de desempenho decorrente desse. Todos os testes foram realizados em ambiente Windows 10 Pro com 8 GB de memória RAM e processador hexa-core AMD FX(tm)-6300.

Tabela 4.1: Resultados obtidos através do webcrawler

Modo de execução	Tempo de execução	Quantidade de URLs obtidas	Número de URLs obtidas por segundo
Sequencial	624,8 segundos	20455	32,73
Concorrente	76.3 segundos	20455	268

Conforme a Tabela 4.1 houve diferença significativa no tempo de execução. Sequencialmente levou-se 624,8 segundos, ou 10,4 minutos, enquanto concorrentemente o mesmo processo foi executado em 76,3 segundos, ou 1,2 minutos, o que representa uma aceleração de 8 vezes em relação à estratégia sequencial. Em um primeiro momento pode-se acreditar que, por existir setenta e cinco processos ao invés de apenas um, a velocidade de execução simultânea deveria aumentar na mesma proporção. Teoricamente o aumento de velocidade com o paralelismo deveria ser linear, entretanto, segundo a Lei de Amhahl, isto não ocorre devido ao tempo necessário para a fração sequencial dos diversos processos paralelos (26).

A quantidade de URLs obtidas em ambos modos de execução foi o mesmo. Esperava-se esse resultado, pois o número de URLs independe do modo de execução. Foram obtidos uma média

de 272,73 endereços para cada URL de partida e 56 para cada salto do *webcrawler*.

4.1.1 Desempenho na extração de *features*

Dadas as URLs obtidas pelo *webcrawler* realizou-se a extração das *features*. Escolheu-se aleatoriamente 6277 URLs não maliciosas e todas as 5426 maliciosas presentes no *PhishTank* gerando o dataset com 11703 elementos utilizado para treino do sistema. O parâmetro avaliado será somente a velocidade de execução paralela, para 20 processos, e serial. Diferentemente do *webcrawler* não foi possível extrair todas as URLs de forma serial devido à longa demora. Realizou-se, então, o processo para 500 endereços e estimou-se o tempo de execução para todo *dataset*.

Tabela 4.2: Tempo para extração de *features*

Modo de execução	Tempo de execução	Número de URLs processadas por segundo
Sequencial	529,8 minutos	0,37
Concorrente	88,3 minutos	3,87

Mesmo para a execução concorrente é evidente que esta fase do sistema inteligente é a mais lenta, sendo o gargalo de todo o sistema. Assim como no *webcrawler*, devido à lei de Amhahl (26), não há um crescimento linear na velocidade de implementação com o uso de processos em paralelo. A Tabela 4.3 compara a velocidade de execução para cada URL entre os processos, a extração de *features* e o *webcrawler*.

Tabela 4.3: Comparação da velocidade de execução para cada URL entre a extração de *features* e o *webcrawler*

Modo de execução	Número de URLs encontradas por segundo no *webcrawler*	Número de URLs processadas por segundo pelo removedor de *features*
Sequencial	32,73	0,37
Concorrente	268	3,87

Para melhor compreensão da demora na execução desta tarefa, calculou-se a porcentagem de tempo gasto em cada umas das funções. A responsável pela análise sintática foi a mais rápida ocupando 0,7% do tempo total, seguida pela função “WhoIs” com 41,1% sendo a função responsável pela obtenção dos dados do cabeçalho HTTP a mais lenta com 58,2% do tempo gasto. Conlcuiu-se que a demora ocorreu por conta das análises “WhoIs” e “HTTP”. O motivo são diversos “TimeOuts” nas consultas ao protocolo, principalmente nos sites considerados maliciosos visto que muitos são retirados do ar rapidamente. Por outro lado, o processo de extração das *features* do *dataset* de treino será executada apenas uma vez, sendo assim o longo tempo gasto nessa função não gera grande impacto na usabilidade do sistema.

4.2 WEKA

Esta Seção destaca o uso do programa *Weka* para auxílio da seleção das características (*features*) extraídas de cada URL e seus respectivos sítios, que juntos formam os *datasets* de treino e teste. Os resultados são apresentados segundo Tabelas de Classificação, popularmente conhecidas como Tabelas de Confusão. Elas representam um *layout* de tabela específico que permite a visualização do desempenho de um algoritmo de classificação, tipicamente um aprendizado supervisionado.

4.2.1 Teste do DataSet com algoritmos J48 e RandomTree

Por ser um *software* que não permite a integração com outros, o *Weka* foi utilizado para averiguação e validação da qualidade do *dataset* de treino gerado pelo *webcrawler* e *Phishtank*. Realizou-se a primeira análise por meio dos algoritmos J48 e RandomTree. Utilizou-se, para isso, os 11703 endereços qualificados na Tabela 3.2. Para cada algoritmo os testes foram executados com "cross-validation" para 10 pastas e "percentage split" no qual 70% dos endereços foi utilizado para treino e o restante pra teste. Como resultado foram analisados os seguintes parâmetros fornecidos pelo *Weka*:

- Instâncias classificadas corretamente: representa a quantidade de URLs classificas corretamente.
- Instâncias classificadas incorretamente: representa a quantidade de URLs classificas incorretamente.
- Taxa de verdadeiro positivo (VP): representa a taxa de URLs classificas corretamente conforme a classe fornecida (maliciosa ou não).
- Taxa de falso positivo (FP): representa a taxa de URLs classificas erroneamente conforme a classe fornecida (maliciosa ou não).

Iniciou-se a implementação por meio do método "cross-validation'. Esse é de fundamental importância devido a sua capacidade de avaliar a eficiência de generalização de um modelo a partir de um conjunto de dados. A taxa de acerto superior a 93% para ambos os algoritmos utilizando o método "cross-validation", conforme Tabela 4.4, é o primeiro indício de que o *dataset* pode ser utilizado no sistema inteligente. Para melhor entendimento dos resultados deve-se observar outros parâmetros.

A Tabela 4.5 apresenta os resultados de forma mais detalhada. É importante a interpretação destes dados, pois observando apenas a acurácia do modelo não é possível saber se esse está classificando corretamente ambas as instâncias ou apenas uma delas. As taxas de verdadeiros positivos mantiveram-se acima de 90% e a de falsos positivos abaixo de 9%. Portanto, assim

Tabela 4.4: Taxa de acerto para método “cross-validation”

Algoritmo	Instâncias classificas corretamente	Instâncias classificas incorretamente	Taxa de acerto
J48	11021	681	94.18%
RandomTree	10908	794	93.21%

Tabela 4.5: Taxa de verdadeiro positivo e falso positivo para método “cross-validation”

Algoritmo	Classe da URL	Taxa de verdadeiro positivo	Taxa de falso positivo
J48	Maliciosa	93,6%	5,3%
	Não maliciosa	94,7%	6,4%
RandomTree	Maliciosa	91,6%	5,4%
	Não maliciosa	94,6%	8,4%

como na acurácia, os resultados apresentados nas tabelas supracitadas oferecem indícios de que o *dataset* é adequado para o treinamento da rede neural que compõe o sistema inteligente.

A execução do método “percentage split” assemelha-se mais com a implementação do sistema inteligente do que o “cross-validation”, pois utiliza parte do *dataset* para treino e o restante para teste. A diferença da implementação do método de divisão percentual para o sistema proposto é que todo o *dataset* utilizado agora para treino e teste será utilizado exclusivamente para a primeira função, enquanto a segunda será exercida por um conjunto de dados inédito. Os resultados para o método “percentage split” foram obtidos para 3511 URLs que representam 30% do dataset dividido pelo Weka para realização dos testes.

Tabela 4.6: Taxa de acerto para método “percentage-split”

Algoritmo	Instâncias classificas corretamente	Instâncias classificas incorretamente	Taxa de acerto
J48	3302	209	94.04%
RandomTree	3291	220	93.73%

Assim como a taxa de acerto para o “cross-validation” ambos os algoritmos atingiram taxa superior a 93%, conforme Tabela 4.6. Além desse valor relevante, a semelhança entre a assertividade dos métodos reforça a hipótese de que o *dataset* gerado pelo *webcrawler* em conjunto com o *PhishTank* é suficiente para uso na rede neural. A Tabela 4.7 confirma a validade do *dataset*. As taxas de verdadeiros positivos acima de 90% e a de falsos positivos abaixo de 8% possibilitam afirmar que o modelo consegue classificar corretamente ambas as classes.

4.2.2 Visualização com Weka

O *Weka* também permite a visualização de diversos aspectos do *dataset*, entre eles a árvore de decisão utilizada no algoritmo RandomTree e a correlação de diferentes *features* por meio de gráficos. A árvore de decisão gerada a partir dos resultados apresentados no Capítulo 3 é muito complexa. Assim, não é possível a sua representação neste trabalho. Como solução alterou-se

Tabela 4.7: Taxa de verdadeiro positivo e falso positivo para método “percentage-split”

Algoritmo	Classe da URL	Taxa de verdadeiro positivo	Taxa de falso positivo
J48	Maliciosa	93,7%	5,7%
	Não maliciosa	94,3%	6,3%
RandomTree	Maliciosa	92,5%	5,2%
	Não maliciosa	94,8%	7,5%

o parâmetro “minNum” do algoritmo RandomTree de 1 para 1000. Este representa o peso total mínimo da instância presente em cada folha da árvore. Com o aumento da variável diminuiu-se o número de folhas e, consequentemente, a taxa de acurácia; permitiu-se, porém, a exibição da árvore para ilustração. A árvore de decisão é a mesma para ambos os métodos “percentage split” e “cross-validation”. A Tabela 4.8 representa os resultados obtidos no teste que deu origem à Figura 4.1.

Tabela 4.8: Resultados do teste para algoritmo RandomTree utilizando o método “percentage split” e parâmetro “minNum” definido como 1000

	Classe da URL	Resultado
Taxa de acerto	Ambas	77,95%
Taxa de verdadeiro positivo	Maliciosa	82,3%
	Não maliciosa	74,1%
Taxa de falso positivo	Maliciosa	25,9%
	Não maliciosa	17,7%

Conforme esperado o resultado das taxas de acerto, verdadeiro positivo e falso positivo pioraram. Todavia esses não serão levados em consideração, pois visa-se apenas que o algoritmo construa uma árvore de decisão mais simples e, portanto, de possível exibição.

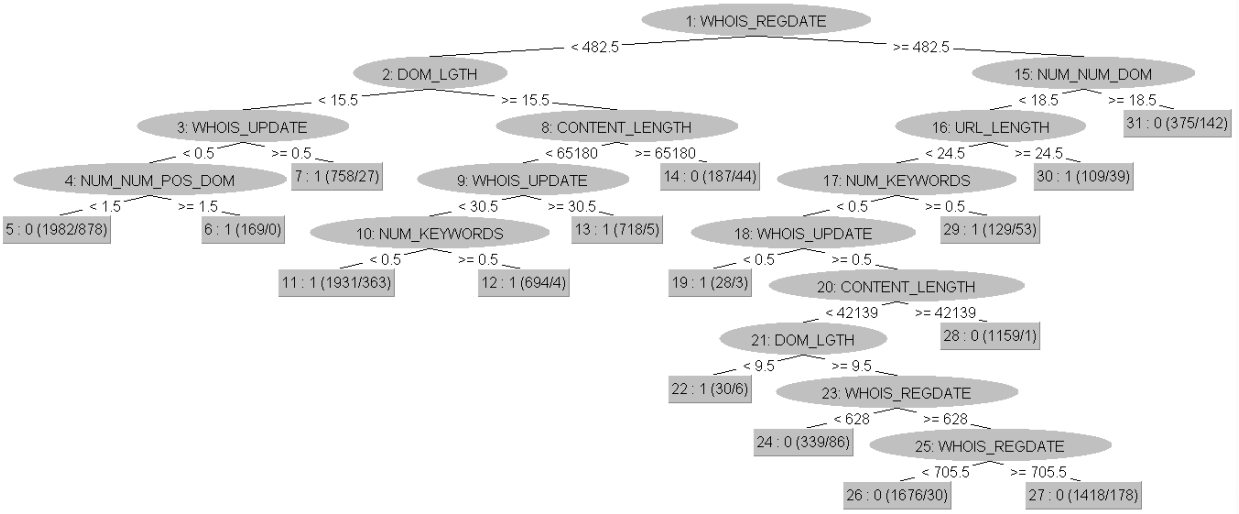
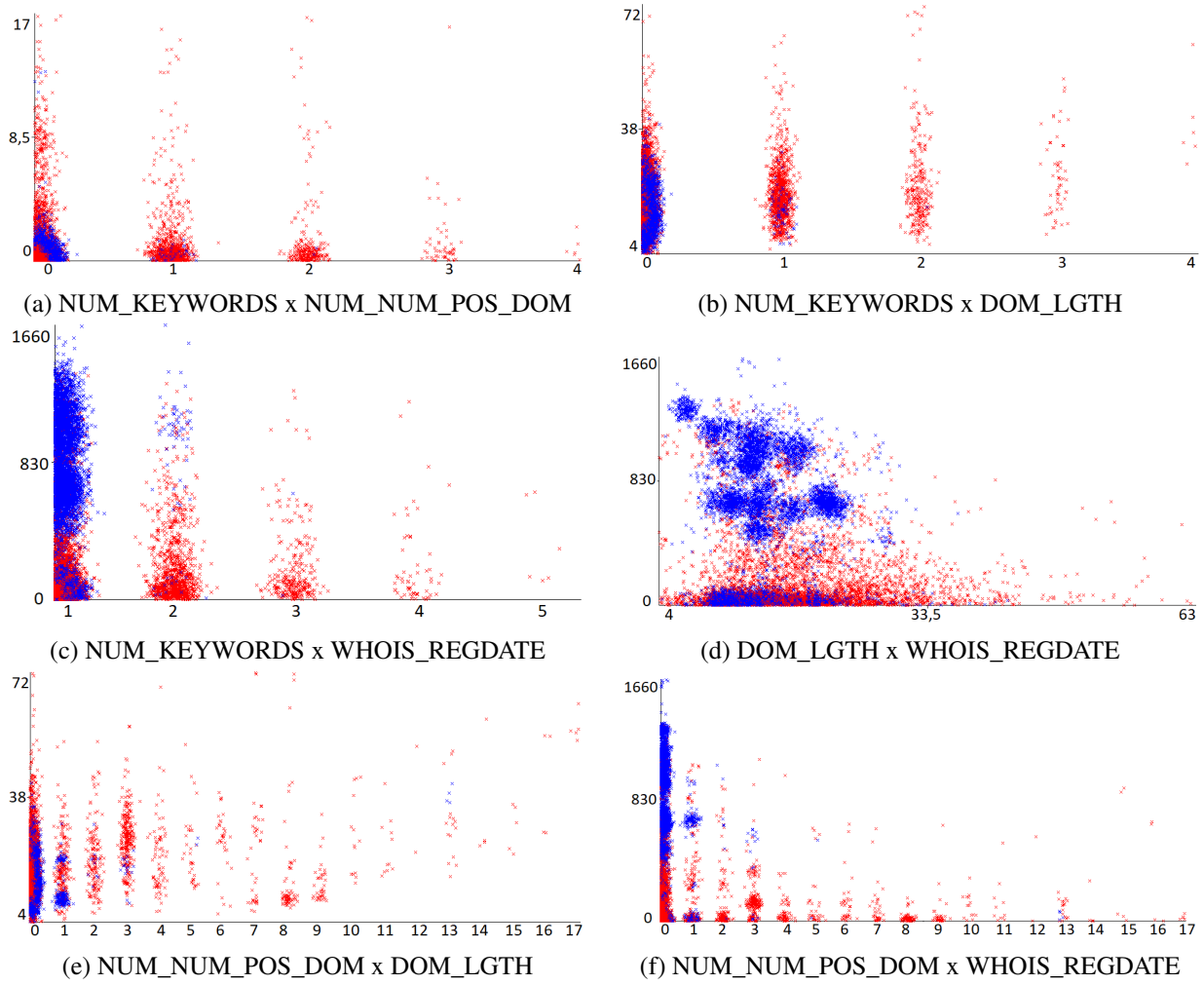


Figura 4.1: Árvore de decisão gerada pelo algoritmo RandomTree

A interpretação da árvore de decisão da Figura 4.1 é simples. Os elementos representados



(a) NUM_KEYWORDS x NUM_NUM_POS_DOM

(b) NUM_KEYWORDS x DOM_LGTH

(c) NUM_KEYWORDS x WHOIS_REGDATE

(d) DOM_LGTH x WHOIS_REGDATE

(e) NUM_NUM_POS_DOM x DOM_LGTH

(f) NUM_NUM_POS_DOM x WHOIS_REGDATE

Figura 4.2: Visualização da correlação entre as principais características (*features*) no Weka.

por folha de formato oval são de decisão, essa será baseada nos números presentes nas linha adjacentes a folha. Por exemplo, se a URL possuir a idade maior que 482,5 deve-se seguir para a análise da *feature* "NUM_NUM_DOM", caso contrário será avaliado o tamanho do domínio. Para classificação do endereço deve-se realizar este procedimento até alcançar-se a folha de formato retangular. No interior desta estão presentes número organizados no seguinte formato: "X: Y (A/B)'. No qual o "X" representa um número sequencial atribuído pelo *Weka* para cada folha, o "Y" é a classe (maliciosa ou não maliciosa) que a URL deve ser classificada, o "A" refere-se ao número de endereços classificas pela folha e o "B" representa a quantidade de links classificados incorretamente por essa, o último pode ser omitido caso zero.

Para a correlação de *features* por meio de gráficos escolheu-se aquelas que possuem maior relevância. Para isso, optou-se pelas folhas do topo da árvore gerada no treino com o *dataset* completo, e não a demonstrada na Figura 4.1, são elas: "WHOIS_REGDATE", "DOM_LENGTH" , "NUM_NUM_POS_DOM" e "NUM_KEYWORDS". Para todos os gráficos a cor vermelha representa URLs maliciosas e azul não maliciosos. Realizou-se a correlação entre todos os atributos supracitados.

Da Figura 4.2 pode-se chegar a diversas conclusões. A Figura 4.2a mostra que URLs que possuem mais de uma palavra chave têm grande probabilidade de serem maliciosas, assim como os endereços que possuem mais de quatro números nos subdiretórios. Ainda na análise do gráfico é possível afirmar que as URLs que possuem as duas características supracitadas simultaneamente, neste *dataset*, são maliciosas. Assim como na Figura 4.2b pode-se concluir que URLs que possuem mais de uma palavra chave têm grande probabilidade de serem maliciosas. Além disso, domínios com menos de 38 caracteres tendem a não ser maliciosos. Uma característica interessante é a ausência de URLs não maliciosas mesmo quando o tamanho do domínio é pequeno e há a presença de mais de duas palavras chave. A Figura 4.2c correlaciona as duas principais *features* do *dataset*. Além das conclusões já apresentadas nos gráficos anteriores para o número de palavras chave, esse gráfico apresenta a relação da idade da página com sua índole. Há grande concentração de páginas não maliciosas na região onde não há palavras chave e a idade de registro é superior a, aproximadamente, 400 semanas. Também é notável a baixa densidade de URLs maliciosas com idade avançada. Isto ocorre pois sites maliciosos tendem a ser registrados e retirados do ar rapidamente. Quanto a idade dos endereços a Figura 4.2d revela característica interessante: a baixa quantidade de páginas com idade entre, aproximadamente, 100 e 400 semanas. A *feature* "DOM_LGTH" mostrou-se útil para classificar sites com mais de 100 caracteres como maliciosos, pois abaixo dessa quantidade há alta densidade de URLs maliciosas ou não. O gráfico representado na Figura 4.2e permite inferir que páginas com mais de 3 números nos subdiretórios e antes da primeira barra possivelmente será malicioso. Há, também, grande concentração de URLs não maliciosas na região que representa a presença de 1 número nos subdiretórios naquelas com o domínio de tamanho entre, aproximadamente, 6 e 12. A Figura 4.2f apresenta, além das características já citadas sobre a idade e quantidade de números nos subdiretórios, algumas características relevantes. Por exemplo, a concentração de páginas não maliciosas que possuem 1 número nos subdiretórios e idade próxima de 780 semanas e a presença de algumas páginas de pouca idade com 13 números nos subdiretórios.

4.3 REDE NEURAL WISARD

Com a validação do *dataset* e o melhor entendimento de cada *feature* realizados com auxílio do *Weka*, utilizou-se a rede neural *WiSARD* por meio da biblioteca "wisardpkg". Realizou-se, então, a representação dos *datasets* de treino e teste por meio dos dois esquemas propostos: representação numérica binária tradicional e a representação binária por faixas. Deseja-se que o sistema inteligente possua a maior acurácia possível e, sem alterar os conjuntos de dados utilizados para teste e treino, apenas o método binário de representação e o parâmetro *adressSize* têm impacto na taxa de acerto da rede neural. Para definir o método e o valor que otimizam o desempenho do sistema foram realizados diversos testes. Em todos eles utilizou-se o mesmo conjunto de dados para treino e teste da rede neural.

Para definir qual o valor do *adressSize* que otimiza o desempenho da rede realizou-se o treino

Tabela 4.9: Resultado do teste da rede neural para binarização tradicional

Tamanho do AddressSize	Média de acerto (%)	Maior acerto (%)
2	75,7	81,8
3	75,9	81,4
4	72,1	75,5
5	72,8	75,6
6	66,5	68,9
7	69,0	71,6
8	68,8	75,1
9	71,3	72,1
10	70,4	76,5
11	70,2	71,2
12	64,7	64,9
13	60,3	65,1
14	66,2	66,6
15	63,7	64,3
16	55,5	56,6
17	68,8	69,7
18	57,2	57,3
19	68,7	72,0
20	61,1	61,8
21	60,3	61,8
22	60,6	62,0
23	58,8	67,1
24	61,1	66,6
25	62,3	63,8
26	61,0	66,5
27	61,2	64,3
28	52,3	53,7
29	61,3	61,8
30	59,7	64,3
31	56,9	58,2
32	60,6	63,6
33	51,0	55,9

Tamanho do AddressSize	Média de acerto (%)	Maior acerto (%)
34	56,8	57,6
35	54,2	54,4
36	54,0	54,7
37	53,7	54,0
38	43,9	47,8
39	52,7	53,8
40	51,1	51,4
41	55,3	59,1
42	48,8	51,6
43	54,9	59,5
44	55,4	56,6
45	58,3	59,8
46	54,3	54,5
47	53,9	54,3
48	53,7	55,0
49	55,4	55,4
50	51,4	51,4
51	48,8	48,8
52	53,4	53,4
53	52,3	52,3
54	52,7	52,7
55	52,2	52,2
56	52,2	54,1
57	51,7	55,7
58	49,6	50,5
59	49,0	50,0
60	49,9	54,0
61	50,1	52,1
62	50,2	53,7
63	51,1	55,1
64	52,1	53,0
64	52,1	53,0

e teste da rede neural para os valores entre 2 e 64 do parâmetro; são, pois, todos os possíveis. Além disso realizou-se o teste supracitado dez vezes para cada um dos valores, pois o algoritmo possui natureza não determinística. Já para decidir qual esquema de representação binária é mais efetivo efetuou-se o teste acima para para ambos. Como parâmetro de verificação de eficácia da RNSP utilizou-se a média e a maior taxa de acerto. A Tabela 4.9 apresenta o resultado dos testes utilizando o esquema de representação binária tradicional. A coluna "Média de acerto (%)" apresenta a taxa de média de acerto para os dez treinos e testes realizados, já a o campo "Maior acerto (%)" demonstra qual a maior taxa de acerto atingida durante o mesmo procedimento.

Conforme Tabela 4.9 a taxa de assertividade da rede diminui com o aumento do tamanho do *adressSize* devido a característica imprevisível do número de bits do *dataset*. Com tuplas de tamanho menor tamanho é possível endereçar mais memórias *RAMs* e assim aumentar a capacidade da rede neural. Por outro lado a utilização de muitas memórias torna mais lento o treinamento, esse fator será analisado posteriormente. Como a acurácia da rede mostrou-se similar para o *adressSize* de tamanho 2 e 3 a escolha do valor do parâmetro deverá ser determinada pela velocidade de treino.

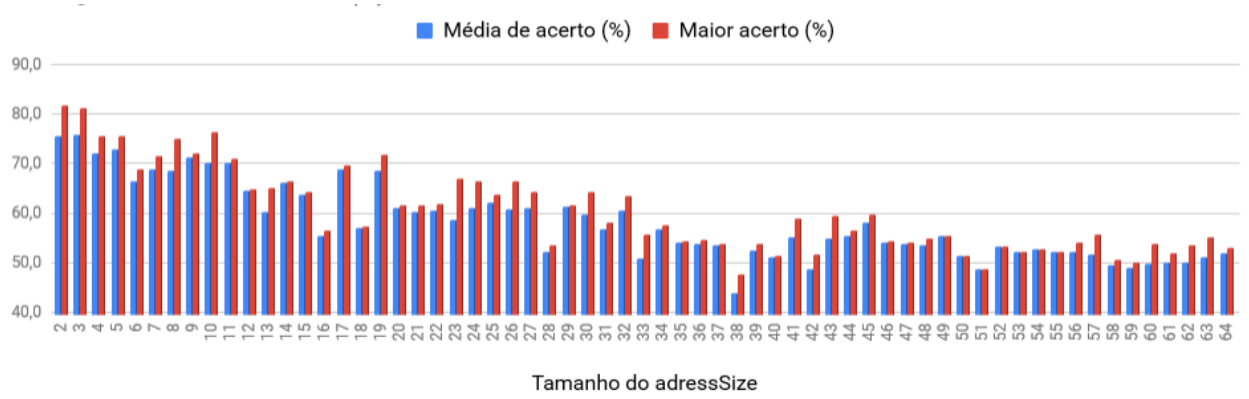


Figura 4.3: Taxa média e maior acerto para representação binária tradicional

A Figura 4.3 evidencia a melhora do desempenho da rede caso opte-se por utilizar o menor valor obtido em dez treinamentos ao invés do primeiro. Essa opção deverá ser realizada caso o tempo para treinar a rede dez vezes não comprometa o funcionamento do sistema. Efetuou-se essa análise após a decisão do sistema de representação binária. A Tabela 4.10 demonstra os resultados obtidos para o esquema de representação binária por faixas. Essa é composta pelos mesmos elementos da Tabela 4.9, e possui similaridades com a Tabela do esquema de binarização tradicional. Novamente quanto maior o valor do *adressSize*, menor a taxa eficácia de rede. Por outro lado, a diferença de desempenho para o valor do tamanho do endereçamento 2 e 3 aumentou significativamente, sendo preferível o menor deles.

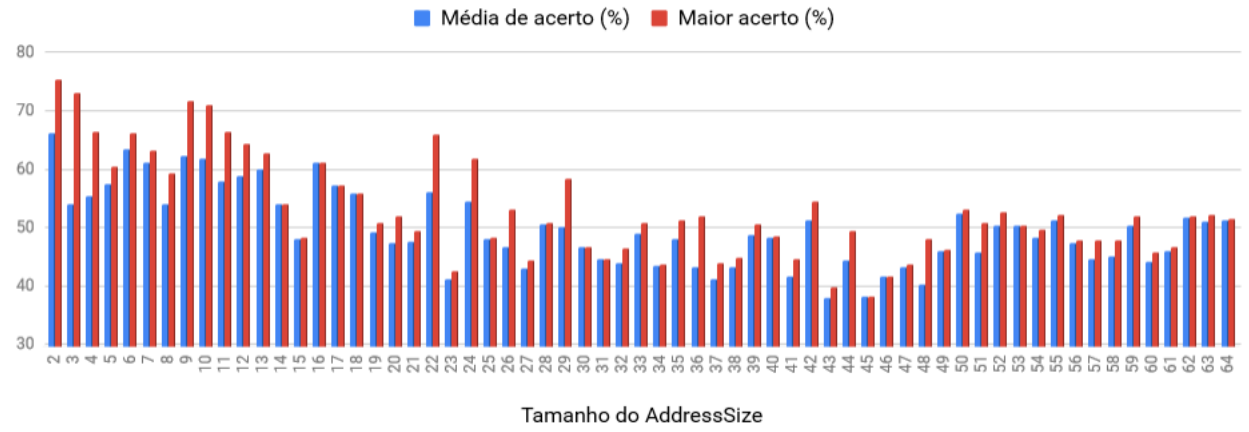


Figura 4.4: Taxa média e maior acerto para representação binária por intervalos

O gráfico representado na Figura 4.4 demonstra característica importante e negativa. Há grande variação entre o valor médio da taxa de acerto e o maior acerto, ou seja, há grande flutuação no valor que indica a assertividade do sistema. Ademais quando comparado com a representação binária tradicional esta apresenta pior desempenho para a maioria dos valores de *adressSize*. Por isso optou-se pela adoção da representação binária tradicional, cujos resultados são exibidos na Tabela 4.9.

Com o esquema de representação binário definido, basta decidir se o tamanho do parâmetro *adressSize* adotado deve ser “2” ou “3” e se é possível utilizar a maior taxa de acerto obtida após

Tabela 4.10: Resultado do teste da rede neural para binarização por intervalos

Tamanho do AddressSize	Média de acerto (%)	Maior acerto (%)
2	66,16	75,51
3	53,95	73,04
4	55,49	66,41
5	57,47	60,48
6	63,51	66,17
7	61,1	63,3
8	53,99	59,24
9	62,38	71,61
10	61,79	71,11
11	57,87	66,41
12	58,88	64,44
13	60,08	62,71
14	54,07	54,1
15	48,16	48,31
16	61,11	61,27
17	57,36	57,36
18	55,93	55,93
19	49,31	50,74
20	47,47	52,02
21	47,65	49,35
22	56,19	65,92
23	41,14	42,63
24	54,54	61,77
25	48,02	48,31
26	46,8	53,21
27	43,01	44,36
28	50,55	50,79
29	50,04	58,5
30	46,58	46,58
31	44,65	44,65
32	43,95	46,53
33	49,09	50,79

Tamanho do AddressSize	Média de acerto (%)	Maior acerto (%)
34	43,5	43,81
35	47,98	51,18
36	43,22	52,07
37	41,18	43,91
38	43,13	44,75
39	48,74	50,64
40	48,24	48,61
41	41,66	44,7
42	51,17	54,59
43	38,06	39,76
44	44,42	49,45
45	38,22	38,22
46	41,54	41,54
47	43,32	43,76
48	40,35	48,07
49	45,9	46,24
50	52,5	53,06
51	45,85	50,74
52	50,37	52,67
53	50,34	50,39
54	48,32	49,75
55	51,29	52,12
56	47,33	47,87
57	44,54	47,87
58	45,15	47,82
59	50,33	52,02
60	44,09	45,79
61	45,91	46,58
62	51,73	51,97
63	51,04	52,27
64	51,33	51,43
64	52,1	53,0

dez treinos e testes ao invés do primeiro resultado (representado pela média). Para isto mediu-se o tempo de execução gasto no treino e teste da rede neural para ambos os valores do parâmetro.

Tabela 4.11: Tempo de execução para dez treinos seguidos de teste para melhores valores do “addressSize”

Valor do “adressSize”	Tempo de execução (segundos)
2	17,67
3	8,45

Com os resultados da Tabela 4.11 define-se os últimos parâmetros do sistema inteligente. Decidiu-se por utilizar o valor do “addressSize” como “2”, pois a assertividade da rede manterá um patamar elevado conforme Tabela 4.9. Ademais verificou-se a possibilidade da utilização da maior taxa de acerto atingida em dez tentativas, visto que a execução de 17,67 segundos não acarreta prejuízo na usabilidade do sistema. A rede neural será, pois, treinada apenas uma vez durante a inicialização do sistema, portanto intervalo de tempo abaixo de 20 segundos é aceitável.

A Tabela 4.12 mostra os parâmetros escolhidos para a melhor acurácia e desempenho da rede neural e, consequentemente, do sistema inteligente. A taxa de acerto de aproximadamente 81,8% é inferior aos resultados obtidos pelo *weka* para ambos os algoritmos nos dois métodos de divisão

Tabela 4.12: Características da rede neural

Representação binária	Valor do “adressSize”	Resultado adotado	Taxa de acerto esperada
Tradicional	2	Melhor em 10 tentativas	Aproximadamente 81,8%

do dataset testados; a assertividade atingida rede neural, entretanto, foi expressiva. No *weka* o mesmo *dataset* utilizado para treino serviu como teste de acordo com o método de divisão. Já no teste da rede neural o conjunto de dados era inédito e gerado de forma a diferenciar-se o máximo do *dataset* de treino.

Tabela 4.13: Tabela de resultados para *adressSize* definido como “2”

Número do teste	Maior taxa de acerto	Tempo de execução
1	81,45%	17,09 segundos
2	81,00%	17,14 segundos
3	81,60%	17,32 segundos
4	82,14%	17,63 segundos
5	79,37%	17,07 segundos
6	79,92%	16,77 segundos
7	80,11%	17,18 segundos
8	78,9%	17,30 segundos
9	82,09%	17,32 segundos
10	80,31%	17,20 segundos

A Tabela 4.13 confirma a possibilidade da escolha da maior taxa de acerto para o *adressSize* definido como “2”. Realizou-se 10 testes, cada um deles contendo 10 conjuntos de treino e teste da rede neural e, ainda assim, a maior taxa de acerto manteve-se próxima de 80% e o tempo de execução próximo a 17 segundos. Dessa forma pode-se afirmar que a taxa de acerto de aproximadamente 81,4%, estimado para o sistema na Tabela 4.12, não foi um ponto fora da curva.

O desempenho da rede neural sem peso *WiSARD* para o treinamento realizado com o *dataset* gerado conjuntamente pelo *webcrawler* e *PhishTank* mostrou-se bastante eficaz. Conclui-se que é possível a implementação do modelo proposto para o sistema inteligente de identificação de sítios infectados por *phishing*.

5 CONCLUSÃO E TRABALHOS FUTUROS

Compreender a base do aprendizado de máquinas e redes neurais é o primeiro passo para propor novas soluções na área de segurança cibernética. Conforme visto no decorrer do trabalho, a facilidade para criação de páginas falsas vem aumentando e há demanda para proposição de novas formas de prevenção desse tipo de ataque. Neste trabalho foram apresentadas bases da teoria de segurança cibernética juntamente com uma introdução ao aprendizado de máquina. A partir desses conceitos, foi possível propor um sistema para classificação de páginas infectadas pelo *phishing* por meio de uma rede neural.

Conforme o Capítulo 4, o mecanismo de busca de endereços dentro de páginas *web*, o *web-crawler*, é capaz de obter páginas não maliciosas com rapidez. Sua execução se deu de forma rápida e a proporção de endereços encontrados para cada página visitada foi satisfatória. Além disso, a execução de diversos *crawlers* por meio de multiprocessamento permitiu a otimização do tempo e a realização da busca horizontal para aprimorar a diversidade do *dataset* de treino.

A extração de características, chamadas de *features*, dos link obtidos mostrou-se a parte problemática do sistema inteligente. A natureza diversificada da rede mundial de computadores afetou esse processo devido aos diversos e imprevisíveis formatos das variáveis adquiridas nas consultas ao *WhoIs*, além do impacto negativo dos *timeouts* no tempo de execução das classes responsáveis pela análise *WhoIs* e *HTTP*. Resolveu-se, paliativamente, os problemas supracitados com intervenções manuais no *dataset* e com a implementação do multiprocessamento na análise das páginas.

A análise da qualidade do *dataset* por meio do *Weka* validou e forneceu a segurança necessária para implementação do sistema inteligente via rede neural sem peso. Além disso, a análise visual da árvore de decisão gerada pelo algoritmo *RandomTree* presente na Figura 4.1 ajudou na compreensão deste conceito. Os gráficos exibidos na Seção 4.2.2 auxiliaram, ademais, o entendimento da importância e da relevância de cada *feature* do *dataset*.

A implementação da rede neural sem peso *WiSARD* treinada pelo *dataset* exigiu a definição de características para otimização dos resultados. Mesmo com o desempenho abaixo daquele obtido pelo *Weka*, a assertividade e a velocidade do treinamento da rede neural corroboram com a possibilidade da implementação do sistema inteligente proposto.

Por fim, a análise conjunta de todos os elementos da ferramenta determinou ser possível a implementação do sistema inteligente proposto para a identificação de páginas infectadas por *phishing*.

5.1 SUGESTÕES DE TRABALHOS FUTUROS

Conforme descrição do sistema inteligente, os seus componentes não são integrados. Sendo algumas etapas, como o cálculo da idade de endereços por meio do *excel* e a limpeza do *dataset*, realizadas de forma manual. A primeira proposta de continuação é a integração de ambos os componentes do sistema de forma automatizada.

Além disso, propõe-se a implementação da interface responsável por analisar, em tempo real, as URLs acessadas pelo usuário no navegador. Sugere-se a criação de um *plugin* capaz de consultar a rede neural treinada para classificação.

Por último, sugere-se realizar testes para possível melhoria do *dataset* e, consequentemente, da assertividade da rede neural e do sistema inteligente como um todo. Para isso, propõe-se estudar com maior profundidade diferentes métodos de binarização, pois sabe-se que a representação binária tradicional não produz os melhores resultados. Além disso, é importante aumentar o número de URLs de partida utilizadas no *webcrawler*, para obtenção de páginas não maliciosas, e buscar outras fontes de páginas maliciosas além do *PhishTank*, a fim de aumentar *dataset* e melhorar a assertividade da rede.

REFERÊNCIAS BIBLIOGRÁFICAS

1 DFNDR LAB. *4º Relatório da Segurança Digital no Brasil.* Disponível em: <https://www.psafe.com/dfndr-lab/wp-content/uploads/2018/08/Relat\%C3\%B3rio-da-Seguran\%C3\%A7a-Digital-no-Brasil-2-trimestre-2018.pdf>. Acesso em: 20 Nov. 2018.

2 MACHADO, T. M. Projeto dedicado de redes neurais sem peso baseadas em neurônios de lógica probabilística multi-valorada. p. 1–155, 2017.

3 TECHTECHNIK. *How to make Phishing Page for Gmail*. Disponível em: <http://www.techtechnik.com/how-to-make-phishing-page-for-gmail/>. Acesso em: 26 Nov. 2018.

4 TECHTUDO. *Golpe de phishing com lojas virtuais cresce quase 300% em 2018.* Disponível em: <https://www.techtudo.com.br/noticias/2018/10/golpe-de-phishing-com-lojas-virtuais-cresce-quase-300-em-2018.ghtml/>. Acesso em: 29 Out. 2018.

5 GARERA S.; PROVOS, N. C. M. R. A. D. A framework for detection and measurement of phishing attacks. p. 1–8, 2006.

6 JEEVA S. C. E RAJSINGH, E. B. Intelligent phishing url detection using association rule mining. p. 1–19, 2016.

7 SAHOO CHENGHAO LIU, e. S. C. H. D. Malicious url detection using machine learning: A survey. p. 1–21, 2017.

8 GAIKWAD, S. Review of phishing detection techniques. p. 15–19, 2014.

9 ALKHOZAE M. G. E BATARFI, O. A. Phishing websites detection based on phishing characteristics in the webpage source code. p. 283–291, 2010.

10 GOOGLE. *Goole Safe Browsing API.* Disponível em: <http://code.google.com/apis/safebrowsing/>. Acesso em: 10 Jul. 2018.

11 MICROSOFT. *Microsoft Smart Screen.* Disponível em: <http://windows.microsoft.com/en-US/>. Acesso em: 10 Jul. 2018.

12 JOIN the fight against phishing.

13 WINTERFIELD STEVE; ANDRESS, J. *The Basics of Cyber Warfare.* [S.l.]: Syngress Publishing, 2012.

14 WITTEN I. H.; FRANK, E. *Data mining: practical machine learning tools and techniques.* [S.l.]: San Francisco: Morgan Kaufmann Publishers., 2005.

15 QUINLAN, J. R. *C4.5: Programs for machine learning.* [S.l.]: Morgan Kaufmann PublishersInc.,San Francisco, CA, USA., 1993.

16 MCCULLOCH W., P. W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, v. 5, p. 115–133, 1943.

17 BLEDSOE W. W., B. I. Pattern recognition and reading by machine. *Managing Requirements Knowledge, International Workshop on*, v. 0, p. 225, 1959.

18 AUSTIN, J. *RAM-Based Neural Networks*. [S.l.]: River Edge, NJ, USA, World Scientific Publishing Co., Inc., 1998.

19 BRAGA A. DE P.; CARVALHO, A. P. d. L. F. d. L. T. B. *Redes Neurais Artificias: Teoria e Aplicações.* [S.l.]: Travessa do Ouvidor, 11, Rio de Janeiro: LTC editora, 2000.

20 ALEKSANDER I.; THOMAS, W. V. B. P. A. Wisard a radical step forward in image recognition. *Emerald Insight*, 1984.

21 ALEKSANDER I.; MORTON, H. An introduction to neural computing. *London: Chapman and Hall*, 1989.

22 ALEKSANDER, I. Emergent intelligent properties of progressively structured pattern recognition nets. *Pattern Recognition Lett.*, v. 1, p. 375–384, 1983.

23 LE, A. M. e. M. F. A. Phishdef: Url names say it all. p. 1–9, 2010.

24 PARMAR P.; PATEL, K. Comparison of phishing detection techniques. p. 749–751, 2014.

25 GITHUB. *IAZero/wisardpkg*. Disponível em: <https://github.com/IAZero/wisardpkg>. Acesso em: 19 Nov. 2018.

26 ITA. *Lei Amdhal*. Disponível em: <http://www.comp.ita.br/~pauloac/ce703/ce703_cap2_p2.pdf>. Acesso em: 22 Nov. 2018.

6 APÊNDICE

6.1 CÓDIGOS FONTE

6.1.1 Webcrawler

```
# -*- coding: utf-8 -*-

from html.parser import HTMLParser
from urllib.request import urlopen
from urllib import parse
import linecache, shutil
from multiprocessing import Process, Manager
import time

class LinkParser(HTMLParser):

    def handle_starttag(self, tag, attrs):
        if tag ==  a :
            for (key, value) in attrs:
                if key ==  href :
                    self.links = self.links + [newUrl]
    def getLinks(self, url):
        self.baseUrl = url
        response = urlopen(url)
        if response.getheader( Content-Type )== text/html  or response.getheader(
            Content-Type )== text/html; charset=utf-8  or response.getheader(
            Content-Type )==  text/html; charset=UTF-8 :
            htmlBytes = response.read()
            htmlString = htmlBytes.decode("utf-8")
            self.feed(htmlString)

            return htmlString, self.links
        else:
            return "",[]

def spider(url, maxPages, pid, visitados,listaboa,listaruim):
    numberVisited=0
    visitados.append(url)

    while numberVisited < maxPages:
        numberVisited = numberVisited +1
        try:
            print(numberVisited, "Visiting:", url)
            parser = LinkParser()
            data, links = parser.getLinks(url)

```

```
40              for link in links:
41                  if link not in visitados and ( http://  in link or
42                           https://  in link):
43                      visitados.append(link)
44              if numberVisited < len(visitados):
45                  url = visitados[numberVisited]
46              listaboa.append(url)
47          except Exception as e:
48              print(" **Failed!**")
49              if numberVisited < len(visitados):
50                  url = visitados[numberVisited]
51              listaruim.append(url)
52      with open("Output" + str(pid) + ".txt","w") as text_file:
53          for link in visitados:
54              try:
55                  print(link, file=text_file)
56              except:
57                  print("Falha ao escrever")
58
59      print("Finalizado processo", url)
60
61  def main():
62      start_time = time.time()
63      visitados = []
64      listaboa = Manager().list()
65      listaruim = Manager().list()
66
67      with open("links.txt","r") as f:
68          StartUrls = f.readlines()
69      NumeroSaltos = 5
70      processes = []
71      pid=1
72
73      for StartUrl in StartUrls:
74          p = Process(target=spider, args=(StartUrl,NumeroSaltos,pid,visitados,
                listaboa,listaruim))
75          pid += 1
76          p.start()
77          processes.append(p)
78
79
80      for p in processes:
81          p.join()
82
83
84      with open("Listaboa.txt","w") as text_file:
85          for link in listaboa:
86              print(link, file=text_file)
87
88      with open("Listaruim.txt","w") as text_file:
89          for link in listaruim:
90              print(link, file=text_file)
```

```
91
92      arquivos =[]
93      for  x in range(1,len(StartUrls)+1):
94          arquivos.append( Output{}.txt .format(x))
95      print(arquivos)
96
97      with open( output_file.txt , wb ) as wfd:
98          for f in arquivos:
99              with open(f, rb ) as fd:
100                 shutil.copyfileobj(fd, wfd, 1024*1024*10)
101
102
103     print("--- %.4s segundos de execu  o ---" % (time.time() - start_time))
104     print(arquivos)
105
106     input("Press enter to exit ;)")
107
108 if __name__ == "__main__":
109     main()
```

6.1.2 Extrator de informações de páginas

```
1  # -*- coding: utf-8 -*-
2
3  import whois
4  import types
5  import re
6  import csv
7  import time
8  import os
9  import numpy as np
10 from datetime import datetime
11 from urllib.request import urlopen
12 from multiprocessing import Process, Manager
13
14 def main():
15
16     with open("output_file.txt","r") as arquivo:
17         links = arquivo.readlines()
18     split_file("output_file.txt",len(links)//20)
19
20     start_time = time.time()
21     processes = []
22
23     for pid in range(0, 21):
24
25         p = Process(target=Executa, args=( output_file_{}.txt .format(pid),pid))
26         p.start()
27         processes.append(p)
```

```
28
29      for p in processes:
30          p.join()
31
32
33      print("--- %.6s segundos de execu  o ---" % (time.time() - start_time))
34      input("Press enter to exit ;)")
35
36
37  def Executa(path,pid):
38
39      WHOISLIST, SYNTAXLIST, HTMLLIST, LINKSLIST, ZEROLIST = ([] for i in range(5))
40      with open(path,"r") as arquivo:
41          links = arquivo.readlines()
42
43      num_urls = 0
44      time_who = 0
45      time_sintatica = 0
46      time_HTML = 0
47      for URL in links:
48          num_urls += 1
49          print("\nAnalisando a URL",num_urls,"de", len(links),"do processo",pid ,
                URL)
50          Country, State, RegDate, LastUpdate = AnaliseWhois(URL)
51          if Country != "None" or State != "None" or RegDate != " None" or
                LastUpdate != "None":
52              WHOISLIST.append([Country,State,RegDate,LastUpdate])
53              SYNTAXLIST.append(AnaliseSintatica(URL))
54              HTMLLIST.append(AnaliseHTML(URL))
55          else:
56              print("Tudo None")
57
58      for who in WHOISLIST:
59          ZEROLIST.append("0")
60      fieldnames = "URL_LGTH,NUM_SLASHES,NUM_DOTS,NUM_ESP_CAR,NUM_KEYWORDS,DOM_LGTH,
            PRE_DOM_LGTH,NUM_DIR,LGTH_DIR,BIG_SUBDIR,NUM_NUM_PRE_DOM,NUM_NUM_POS_DOM,
            NUM_NUM_DOM,WHOIS_REGDATE,WHOIS_UPDATE,CONTENT_LENGTH,IS_MALICIOUS"
61      np.savetxt( file_name{}.csv .format(pid), np.column_stack((SYNTAXLIST,
            WHOISLIST, HTMLLIST,ZEROLIST)), delimiter=",", fmt= %s , header=fieldnames)
62
63
64
65  def AnaliseHTML(URL):
66      TimeOut = 0
67      try:
68          Content_length = response.getheader( Content-Length )
69      except Exception as e:
70          print("Exce  o AnaliseHTML",e)
71          Content_Type =  None
72      print("Finalizou Analise HTML")
73      return Content_Type
74
```

```
75
76 def AnaliseSintatica(URL):
77
78     URL = URL.replace("http://","")
79     URL = URL.replace("https://","")
80
81     #Tamanho da URL
82     TAMANHO_URL = len(URL.split("/")[0])
83
84     #N mero de pontos e barras
85     NUMERO_PONTOS = URL.count( . )
86     NUMERO_BARRAS = URL.count( / )
87
88     #Numero caracters especiais excetuando pontos e barras
89     NUM_CAR_ESP = len(re.sub( [\w]+  ,  , URL)) - NUMERO_PONTOS - NUMERO_BARRAS
90
91     #Palavras chaves como "confirm, account, banking, secure, ebayisapi, webscr,
           login, e signin"
92     KeyWords = "confirm", "account", "banking", "secure", "ebayisapi", "webscr", "
           login", "signin", "pay", "free", "senha", "security"
93     NUMERO_KEYWORDS = 0
94     for Word in KeyWords:
95         if Word in URL:
96             NUMERO_KEYWORDS += 1
97
98     #Tamanho do domnnio, diretorio, html, do maior subdiretorio e numero de
           diretorios
99     MAIOR_SUB = 0
100    TAMANHO_DIR = 0
101    TAMANHO_ANTES_DOMINIO = 0
102    QTD_NUMEROS_PRE_DOM = 0
103    QTD_NUMEROS_DOM = 0
104    QTD_NUMEROS_POS_DOM = 0
105    for string in URL.split("/"):
106        #ANALISE APOS PRIMEIRA BARRA
107        if string != URL.split("/")[0]:
108            TAMANHO_DIR += len(string)
109            QTD_NUMEROS_POS_DOM += len(re.sub( [\D]+  ,  , string))
110            if len(string) > MAIOR_SUB:
111                MAIOR_SUB = len(string)
112        #ANALISE ANTES DA PRIMEIRA BARRA
113        else:
114            #SE MAIS DE 3 PONTOS ANTES DA BARRA (EX: portal.tcu.gov.br)
115            if len(string.split(".")) > 3:
116                    QTD_NUMEROS_PRE_DOM = len(re.sub( [\D]+  ,  , string.split("."
                           )[0]))
117                    TAMANHO_ANTES_DOMINIO = len(string.split(".")[0])
118                    QTD_NUMEROS_DOM = len(re.sub( [\D]+  ,  , string.split(".")
                           [1])) + len(re.sub( [\D]+  ,  , string.split(".")[2])) +
                           len(re.sub( [\D]+  ,  , string.split(".")[3]))
119                    TAMANHO_DOMINIO = len(string.split(".")[1]) + len(string.split
                           (".")[2]) + len(string.split(".")[3]) + 2
```

```
            else:
                QTD_NUMEROS_DOM = len(re.sub( [\D]+  ,  , string.split("/")[0]))
                TAMANHO_DOMINIO = len(URL.split("/")[0])
    NUMERO_DIR = NUMERO_BARRAS - 1
    print("Finalizou Analise Sintatica")

    return TAMANHO_URL, NUMERO_BARRAS, NUMERO_PONTOS, NUM_CAR_ESP, NUMERO_KEYWORDS
        , TAMANHO_DOMINIO, TAMANHO_ANTES_DOMINIO, NUMERO_DIR, TAMANHO_DIR,
        MAIOR_SUB, QTD_NUMEROS_PRE_DOM, QTD_NUMEROS_DOM, QTD_NUMEROS_POS_DOM


#Funcao responsavel pela extracao das informacoes do WhoIs
def AnaliseWhois(URL):
    try:
        url_data = whois.whois(URL)
    except Exception as e:
        print("Exception AnaliseWhoIs",e)
        RegDate = "None"
        LastUpdate = "None"
        print("Finalizou Analise WhoIs1")
        return RegDate, LastUpdate

    try:
        match = re.search(r \d{8} , url_data.created[0])
        RegDate = datetime.strptime(match.group(),  %Y%m%d ).date()
        match = re.search(r \d{8} , url_data.changed[0])
        LastUpdate = datetime.strptime(match.group(),  %Y%m%d ).date()

    except Exception as e:
        print("Exception data")
        if(type(url_data.creation_date) is list):
            RegDate = url_data.creation_date[0].strftime("%Y-%m-%d")
        else:
            if url_data.creation_date is None:
                RegDate = "None"
            else:
                if (type(url_data.creation_date) is str):
                    RegDate = "None"
                else:
                    RegDate = url_data.creation_date.strftime("%Y-%m-%d")


        if(type(url_data.updated_date) is list):
            LastUpdate = url_data.updated_date[0].strftime("%Y-%m-%d")
        else:
            if url_data.updated_date is None:
                LastUpdate = "None"
            else:
                if (type(url_data.updated_date) is str):
                    LastUpdate = "None"
                else:
                    LastUpdate = url_data.updated_date.strftime("%Y-%m-%d")
```

```
    print("Finalizou Analise WhoIs2")
    return RegDate, LastUpdate



Format_Date = lambda d: "{}-{}-{}".format(d[0:4], d[4:6], d[6:8])

#
def split_file(filepath, lines_per_file=100):
        lpf = lines_per_file
    path, filename = os.path.split(filepath)
    with open(filepath,  r ) as r:
        name, ext = os.path.splitext(filename)
        try:
            w = open(os.path.join(path,  {}_{}{}.txt .format( lixo , 0, ext)),  w
                )
            for i, line in enumerate(r):
                if not i % lpf:
                    w.close()
                    teste = i//lpf
                    filename = os.path.join(path,
                                            {}_{}{} .format(name, i//lpf,ext))
                    w = open(filename,  w )
                w.write(line)
        finally:
            w.close()

if __name__ == "__main__":
    main()
```

6.1.3 Rede Neural

6.1.3.1 Rede neural com binarização tradicional

```
  import  wisardpkg as wp
import time
from wisardpkg import ClusWisard

def main():

        with open("DataSetTreino.csv","r") as arquivo:
                ArquivoTreino = arquivo.readlines()
        with open("DataSetTeste.csv","r") as arquivo:
                ArquivoTeste = arquivo.readlines()
        Entrada = [0 for i in range(len(ArquivoTreino))]
        Teste = [0 for i in range(len(ArquivoTeste))]

        Label = []
```

```
15          nMalicious = 0
16          nNMalicious = 0
17
18
19          print("######BINARIZANDO TREINO######")
20          for i,linha in enumerate(ArquivoTreino):
21                  print("Link:", i)
22                  Dados = linha.split( , )
23                  Entrada[i] = Binarizer(Dados[0],8,1) + Binarizer(Dados[1],5,2) +
                        Binarizer(Dados[2],5,3)
24                  Entrada[i] += Binarizer(Dados[3],7,4) + Binarizer(Dados[4],3,5) +
                        Binarizer(Dados[5],7,6)
25                  Entrada[i] += Binarizer(Dados[6],6,7) + Binarizer(Dados[7],5,8) +
                        Binarizer(Dados[8],11,9)
26                  Entrada[i] += Binarizer(Dados[9],11,10) + Binarizer(Dados
                        [10],4,11) + Binarizer(Dados[11],5,12)
27                  Entrada[i] += Binarizer(Dados[12],9,13) + Binarizer(Dados
                        [13],11,14) + Binarizer(Dados[14],9,15)
28                  Entrada[i] += Binarizer(Dados[15],25,16)
29                  if "1" in Dados[16]:
30                          Label.append("Malicious")
31                  if "0" in Dados[16]:
32                          Label.append("Not Malicious")
33
34
35          print("######BINARIZANDO TESTE######")
36          for i,linha in enumerate(ArquivoTeste):
37                  print("Link:", i)
38                  Dados = linha.split( , )
39                  Teste[i] = Binarizer(Dados[0],8,1) + Binarizer(Dados[1],5,2) +
                        Binarizer(Dados[2],5,3)
40                  Teste[i] += Binarizer(Dados[3],7,4) + Binarizer(Dados[4],3,5) +
                        Binarizer(Dados[5],7,6)
41                  Teste[i] += Binarizer(Dados[6],6,7) + Binarizer(Dados[7],5,8) +
                        Binarizer(Dados[8],11,9)
42                  Teste[i] += Binarizer(Dados[9],11,10) + Binarizer(Dados[10],4,11)
                        + Binarizer(Dados[11],5,12)
43                  Teste[i] += Binarizer(Dados[12],9,13) + Binarizer(Dados[13],11,14)
                         + Binarizer(Dados[14],9,15)
44                  Teste[i] += Binarizer(Dados[15],25,16)
45                  if "1" in Dados[16]:
46                          nMalicious = nMalicious + 1
47                  if "0" in Dados[16]:
48                          nNMalicious = nNMalicious + 1
49
50          WisardRedeNeural(Entrada,Teste,Label,nMalicious,nNMalicious)
51          print("Maliciosos", nMalicious)
52          print("N o maliciosos", nNMalicious)
53          input("Press enter to exit ;)")
54
55  def WisardRedeNeural(Entrada,Teste,Label,nMalicious,nNMalicious):
56
```

```
57          ignoreZero = False
58          MediaErro = [0 for i in range(0,65)]
59          MenorValor = [999 for i in range(0,65)]
60          nTentativas = 10
61          MenorAddressSize = 2
62          MaiorAddressSize = 3
63          verbose = False
64          Tempo = [0 for i in range(0,65)]
65
66          for addressSize in range(MenorAddressSize,MaiorAddressSize+1): ##Testa
                para todos os AddressSize definidos nas vari veis "MenorAdressSize" e
                "MaiorAdressSize"
67              print("##############################ADDRESS SIZE = ",addressSize,
                    "##############################")
68              Tempo[addressSize] = time.time()
69              for Tentativa in range(1,nTentativas+1): ####Testa o n mero de
                    vezes definido na Vari vel nTentativas
70                  ClassErrado = 0
71                  ClassCerto = 0
72                  print("#####Tentativa =",Tentativa,"#####")
73                  wsd = wp.Wisard(addressSize, ignoreZero=ignoreZero,
                        verbose=verbose)
74
75                  wsd.train(Entrada,Label)
76
77                  out = wsd.classify(Teste)
78
79                  for i,d in enumerate(Teste):
80                          if i < nMalicious: ###Olhando maliciosos
81                                  if "Not" in out[i]:
82                                          ClassErrado += 1
83                                  else:
84                                          ClassCerto += 1
85                          if i > nMalicious: ##Olhando n o maliciosas
86                                  if "Not" in out[i]:
87                                          ClassCerto += 1
88                                  else:
89                                          ClassErrado += 1
90                  MediaErro[addressSize] +=100*ClassErrado/(nMalicious+
                        nNMalicious)
91                  if (100*ClassErrado/(nMalicious+nNMalicious)) < MenorValor
                        [addressSize]:
92                      MenorValor[addressSize] = 100*ClassErrado/(nMalicious+
                            nNMalicious)
93                  print("Porcentagem de erro: ", 100*ClassErrado/(nMalicious
                        +nNMalicious))
94
95              Tempo[addressSize] = time.time() - Tempo[addressSize]
96
97              MediaErro[addressSize] = MediaErro[addressSize]/nTentativas
98
99          for i in range(MenorAddressSize,MaiorAddressSize+1):
```

```
100                 print("Media de erro para address size =",i,":", 100 - MediaErro[i
                        ])
101
102         for i in range(MenorAddressSize,MaiorAddressSize+1):
103                 print("Maior acerto para address size =",i,":", 100 - MenorValor[i
                        ])
104                 print(Tempo[i])
105
106
107 def Binarizer(NumDec, NumBits,nParam):
108         binario = str(bin(int(NumDec)))
109         ListBin = []
110         if(len(binario) < NumBits + 3):
111                 for x in range(0, NumBits + 2 - len(binario)):
112                         ListBin.append(0)
113                 for x in range(2,len(binario)):
114                         ListBin.append(int(binario[x]))
115                 return ListBin
116         else:
117                 for x in range(0,NumBits):
118                         ListBin.append(0)
119                 print("N mero maior que previsto no par metro",nParam)
120                 return ListBin
121
122 if __name__ == "__main__":
123     main()
```

6.1.3.2 Rede neural com binarização por intervalos

```
1   import   wisardpkg as wp
2 from wisardpkg import ClusWisard
3 import math
4
5 def main():
6
7         with open("DataSetTreino.csv","r") as arquivo:
8                 ArquivoTreino = arquivo.readlines()
9         with open("DataSetTeste.csv","r") as arquivo:
10                ArquivoTeste = arquivo.readlines()
11        Entrada = [0 for i in range(len(ArquivoTreino))]
12        Teste = [0 for i in range(len(ArquivoTeste))]
13
14        RecebeLabel = [0 for i in range(len(ArquivoTreino))]
15        Label = []
16        nMalicious = 0
17        nNMalicious = 0
18        print("######BINARIZANDO TREINO######")
19        for i,linha in enumerate(ArquivoTreino):
20                print("Link:", i)
```

```
21                  Entrada[i],RecebeLabel[i] = Binarizer(linha, MaxTreino,nMalicious,
                        nNMalicious, True)
22                  Label += RecebeLabel[i]
23
24          print("######BINARIZANDO TESTE######")
25          for i,linha in enumerate(ArquivoTeste):
26                  print("Link:", i)
27                  Teste[i],nMalicious, nNMalicious = Binarizer(linha, MaxTeste,
                        nMalicious,nNMalicious, False)
28
29          WisardRedeNeural(Entrada,Teste,Label,nMalicious,nNMalicious)
30          print("Maliciosos", nMalicious)
31          print("N o maliciosos", nNMalicious)
32          input("Press enter to exit ;)")
33
34  def Binarizer(linha, Maximo,nMalicious,nNMalicious,isTeste):
35          Dados = linha.split( , )
36          ListBin = []
37          ListLabel = []
38
39          for i,Dado in enumerate(Dados):
40                  Max = Maximo[i]
41                  if i == 16 :
42                          if int(Dado) == 1:
43                                  nMalicious = nMalicious + 1
44                                  ListLabel.append("Malicious")
45                          if int(Dado) == 0:
46                                  ListLabel.append("Not Malicious")
47                                  nNMalicious = nNMalicious + 1
48
49                  if i < 16:
50                          #Aplica o LN nas features com grande v a r i a   o
51                          if i == 15 or i == 8 or i == 9 or i == 12 or i == 13:
52                                  if int(Dado) != 0:
53                                          Dado = math.log(int(Dado))
54                                  if int(Maximo[i]) != 0:
55                                          Max = math.log(int(Maximo[i]))
56
57
58                          if int(Dado) >= 0 and int(Dado)< int(Max)/8:
59                                  ListBin.append(1)
60                                  ListBin.append(1)
61                                  ListBin.append(0)
62                                  ListBin.append(0)
63                                  ListBin.append(1)
64                                  ListBin.append(1)
65                                  ListBin.append(0)
66                                  ListBin.append(0)
67                          if int(Dado) >= int(Max)/8 and int(Dado)< int(Max)*2/8:
68                                  ListBin.append(1)
69                                  ListBin.append(0)
70                                  ListBin.append(1)
```

```
71                              ListBin.append(0)
72                              ListBin.append(1)
73                              ListBin.append(0)
74                              ListBin.append(1)
75                              ListBin.append(0)
76                      if int(Dado) >= int(Max)*2/8 and int(Dado)< int(Max)*3/8:
77                              ListBin.append(1)
78                              ListBin.append(1)
79                              ListBin.append(1)
80                              ListBin.append(1)
81                              ListBin.append(0)
82                              ListBin.append(0)
83                              ListBin.append(0)
84                              ListBin.append(0)
85                      if int(Dado) >= int(Max)*3/8 and int(Dado)< int(Max)*4/8:
86                              ListBin.append(1)
87                              ListBin.append(1)
88                              ListBin.append(0)
89                              ListBin.append(0)
90                              ListBin.append(0)
91                              ListBin.append(0)
92                              ListBin.append(1)
93                              ListBin.append(1)
94                      if int(Dado) >= int(Max)*4/8 and int(Dado)< int(Max)*5/8:
95                              ListBin.append(0)
96                              ListBin.append(0)
97                              ListBin.append(1)
98                              ListBin.append(1)
99                              ListBin.append(0)
100                             ListBin.append(0)
101                             ListBin.append(1)
102                             ListBin.append(1)
103                     if int(Dado) >= int(Max)*5/8 and int(Dado)< int(Max)*6/8:
104                             ListBin.append(0)
105                             ListBin.append(1)
106                             ListBin.append(0)
107                             ListBin.append(1)
108                             ListBin.append(0)
109                             ListBin.append(1)
110                             ListBin.append(0)
111                             ListBin.append(1)
112                     if int(Dado) >= int(Max)*6/8 and int(Dado)< int(Max)*7/8:
113                             ListBin.append(0)
114                             ListBin.append(0)
115                             ListBin.append(0)
116                             ListBin.append(0)
117                             ListBin.append(1)
118                             ListBin.append(1)
119                             ListBin.append(1)
120                             ListBin.append(1)
121                     if int(Dado) >= int(Max)*7/8 and int(Dado)<= int(Max):
122                             ListBin.append(0)
```

```
123                                 ListBin.append(0)
124                                 ListBin.append(1)
125                                 ListBin.append(1)
126                                 ListBin.append(1)
127                                 ListBin.append(1)
128                                 ListBin.append(0)
129                                 ListBin.append(0)
130
131         if isTeste:
132                 return ListBin, ListLabel
133         else:
134                 return ListBin, nMalicious, nNMalicious
135
136 def WisardRedeNeural(Entrada,Teste,Label,nMalicious,nNMalicious):
137
138         ignoreZero = False
139         MediaErro = [0 for i in range(0,65)]
140         MenorValor = [999 for i in range(0,65)]
141         nTentativas = 10
142         MenorAddressSize = 2
143         MaiorAddressSize = 64
144         verbose = False
145
146         for addressSize in range(MenorAddressSize,MaiorAddressSize+1): ##Testa
              para todos os AddressSize definidos nas vari veis "MenorAdressSize" e
              "MaiorAdressSize"
147                 print("##############################ADDRESS SIZE = ",addressSize,
                        "##############################")
148                 for Tentativa in range(1,nTentativas+1): ####Testa o n mero de
                        vezes definido na Vari vel nTentativas
149                         ClassErrado = 0
150                         ClassCerto = 0
151                         print("#####Tentativa =",Tentativa,"#####")
152                         wsd = wp.Wisard(addressSize, ignoreZero=ignoreZero,
                                verbose=verbose)
153
154                         wsd.train(Entrada,Label)
155
156                         out = wsd.classify(Teste)
157
158                         for i,d in enumerate(Teste):
159                                 if i < nMalicious: ###Olhando maliciosos
160                                         if "Not" in out[i]:
161                                                 ClassErrado += 1
162                                         else:
163                                                 ClassCerto += 1
164                                 if i > nMalicious: ##Olhando n o maliciosas
165                                         if "Not" in out[i]:
166                                                 ClassCerto += 1
167                                         else:
168                                                 ClassErrado += 1
```

```
169                             MediaErro[addressSize] +=100*ClassErrado/(nMalicious+
                                    nNMalicious)
170
171                             if (100*ClassErrado/(nMalicious+nNMalicious)) < MenorValor
                                    [addressSize]:
172                                 MenorValor[addressSize] = 100*ClassErrado/(nMalicious+
                                        nNMalicious)
173
174                             print("Porcentagem de erro: ", 100*ClassErrado/(nMalicious
                                    +nNMalicious))
175
176                 MediaErro[addressSize] = MediaErro[addressSize]/nTentativas
177
178         for i in range(MenorAddressSize,MaiorAddressSize+1):
179                 print("Media de erro para address size =",i,":", 100 - MediaErro[i
                        ])
180
181         for i in range(MenorAddressSize,MaiorAddressSize+1):
182                 print("Maior acerto para address size =",i,":", 100 - MenorValor[i
                        ])
183
184
185 def Ranges(Treino,Teste):
186         MaxTreino = [0 for i in range(17)]
187
188         MaxTeste = [0 for i in range(17)]
189
190         for linha in Treino:
191                 Dados = linha.split( , )
192                 for i,Dado in enumerate(Dados):
193                         if int(Dado) > int(MaxTreino[i]):
194                                 MaxTreino[i] = int(Dado)
195         for linha in Teste:
196                 Dados = linha.split( , )
197                 for i,Dado in enumerate(Dados):
198                         if int(Dado) > int(MaxTeste[i]):
199                                 MaxTeste[i] = int(Dado)
200
201         return MaxTreino,MaxTeste
202
203
204
205 if __name__ == "__main__":
206     main()
```