RAFAEL ANTONIO GONÇALVES LIMA

UTILIZAÇÃO DE SISTEMAS INTELIGENTES PARA CLASSIFICAÇÃO DE TRÁFEGO MALICIOSO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS – BRASIL
2014

RAFAEL ANTONIO GONÇALVES LIMA

UTILIZAÇÃO DE SISTEMAS INTELIGENTES PARA CLASSIFICAÇÃO DE TRÁFEGO MALICIOSO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae.*

APROVADA: 14 de fevereiro de 2014.

Alcione de Paiva Oliveira
(Coorientador)

Luiz Henrique Andrade Correia

Carlos de Castro Goulart
(Orientador)

Ficha catalográfica preparada pela Seção de Catalogação e Classificação da Biblioteca Central da UFV

T
L732u
2014

Lima, Rafael Antonio Gonçalves, 1986-
Utilização de sistemas inteligentes para classificação de tráfego malicioso / Rafael Antonio Gonçalves Lima. – Viçosa, MG, 2014.
xi, 80f. : il. (algumas color.) ; 29 cm.

Inclui anexos.
Orientador: Carlos de Castro Goulart.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f.60-62.

1. Redes neurais (Computação). 2. Algoritmos genéticos. 3. Sistemas de detecção de intrusão. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22. ed. 006.32

Dedico este trabalho a minha esposa Lívea e meus pais, Carlos e Celina por toda confiança e incentivos.

"Mesmo que já tenha feito uma longa caminhada,
sempre haverá mais um caminho a percorrer."
(Santo Agostinho)

Agradecimentos

Ao professor e orientador, Carlos de Castro Goulart, pelas orientações valiosas e, sobretudo pelos ensinamentos transmitidos e pela paciência e confiança depositada.

Ao ex-aluno e professor Jacson Rodrigues, que muito me incentivou e transmitiu sua experiência adquirida a respeito do tema deste trabalho.

Aos professores por todo apoio e ensinamentos repassados durante toda graduação e mestrado.

Aos meus colegas por todos os conhecimentos compartilhados, discutidos e dificuldades superadas.

Aos meus pais, Carlos e Celina, por todo apoio e ensinamentos durante toda minha vida.

As minhas irmãs, Illian e Laura, meus amigos e demais familiares por todos os apoios e incentivos.

A minha esposa, Lívea, que mesmo nas horas mais difíceis, sempre me apoiou e me ajudou a encontrar o tempo que sempre necessitava.

A Deus por iluminar minha vida e permitir que eu chegasse até essa etapa com firmeza, crescimento pessoal e profissional.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

LIMA, Rafael Antonio Gonçalves, M.Sc., Universidade Federal de Viçosa, Fevereiro de 2014. **Utilização de Sistemas Inteligentes para Classificação de Tráfego Malicioso**. Orientador: Carlos de Castro Goulart. Coorientadores: Alcione de Paiva Oliveira e Ricardo dos Santos Ferreira.

Nos últimos anos, várias pesquisas em detecção de intrusão têm sido realizadas, buscando melhorias nos índices de detecções e diminuição dos alarmes falsos. No entanto, várias delas são criticadas pela utilização de bases de dados que não representam a realidade atual do tráfego das redes de computadores, devido aos seus novos paradigmas e onde as ameaças e a sofisticação dos ataques têm aumentado. Assim, este trabalho traz um estudo sobre uma base de dados mais recente, a ISCX 2012, com características chaves para novas bases de dados e pesquisas. O presente trabalho utiliza etapas do processo de Descoberta de Conhecimento em Banco de Dados (KDD) e técnicas de classificação como árvores de decisão e redes neurais artificiais através do software *Weka* e de um sistema inteligente híbrido, além de realizar análises e comparações dos resultados encontrados.

Abstract

LIMA, Rafael Antonio Gonçalves, M.Sc., Universidade Federal de Viçosa, February of 2014. **Utilization for Intelligent Systems to Classification of Malicious Traffic**. Adviser: Carlos de Castro Goulart. Co-advisers: Alcione de Paiva Oliveira and Ricardo dos Santos Ferreira.

In the last years several researches on detection of intrusion have been done, seeking for improvements in detection and decreasing rates of false alarms. However, several of them are criticized by using databases which do not represent the current reality from computers networks traffic coming off their new paradigms and where the threats and the sophistication of the attacks have increased. This way, that work comes up a study based on a more recent database, ISCX 2012, with features key to new databases and researches. The present work uses Knowledge Discovery in Database (KDD) and classification techniques like decision trees and artificial neural networks through software Weka and soft computing, besides do analysis and comparisons from the results found.

Capítulo 1

Introdução

Nas últimas décadas, a necessidade da segurança da informação nas organizações sofreu mudanças importantes. Antes, a segurança era realizada basicamente através de meios físicos e administrativos como, por exemplo, a utilização de trancas com segredos nos armários para proteger documentos importantes. Com o surgimento do computador, novas ferramentas automatizadas se tornaram necessárias para proteger arquivos e outras informações armazenadas no computador [Stallings, 2010].

Uma importante mudança ocorreu ao surgir os sistemas distribuídos, o uso das redes e dos recursos de comunicação. Neste cenário, o compartilhamento de recursos tornou-se cada vez mais usual e novas ferramentas surgiram para manter a "segurança de rede", protegendo os dados durante sua transmissão.

Atualmente, em qualquer discussão séria que envolva redes de computadores, é obrigatório ter a Internet em mente. O maior crescimento nessa área ocorreu no âmbito dos serviços e das aplicações, como o desenvolvimento da Web, pela utilização dos serviços de correio eletrônico por milhares de usuários simultâneos, pela recepção de áudio e vídeo, pela telefonia através da Internet, pelos serviços de mensagens instantâneas, pelas aplicações P2P (*Peer-to-Peer*) e pelo comércio eletrônico [Kurose & Ross, 2010].

Embora antigamente o uso das redes e de computadores fosse privilégio de poucas organizações e usuários, com o advento da Internet e seu aumento de utilização, a necessidade de segurança ficou mais evidente. Neste cenário, a quantidade de dados e informações transmitidas e armazenadas nos sistemas computacionais aumentou e a Internet se tornou uma ferramenta padrão de comunicação no mundo, desempenhando um papel essencial na gestão das empresas mais bem sucedidas [Mzila & Dube, 2013].

Contudo, juntamente com as vantangens que a Internet traz, há também uma desvantagem substancial que é a exposição de informações valiosas e confidenciais aos riscos de invasões e ameaças cibernéticas [Mzila & Dube, 2013].

1.1 O problema e sua importância

Vários mecanismos de segurança têm sido utilizados pelas empresas para a proteção de seus dados contra acessos indevidos, principalmente provenientes de um ambiente externo como a Internet. Esses mecanismos ao longo do tempo têm se adaptado e incorporado diversas funções com o intuito de abranger a real necessidade das empresas e usuários.

Contudo, a segurança, que mais recentemente se tornou um destaque nos meios de comunicação, por muito tempo fora um quesito deixado em segundo plano, pois envolvia investimentos adicionais que poderiam inviabilizar diversos projetos.

Várias ameaças sempre existiram, mas ao longo do tempo não eram preocupações para a maioria da população, pois não havia cenário propício aos danos. Hoje, entretanto, esta situação mudou. A Internet é uma rede de nível mundial em que as pessoas fazem inúmeras operações financeiras, como compras, pagamentos, etc.

Assim, as redes de computadores atuais são cada vez mais alvos de ameaças. O próprio CERT.br (Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil) ao longo dos últimos anos tem divulgado o total de incidentes que usuários e organizações tem reportado, mostrado na Figura 1.1.

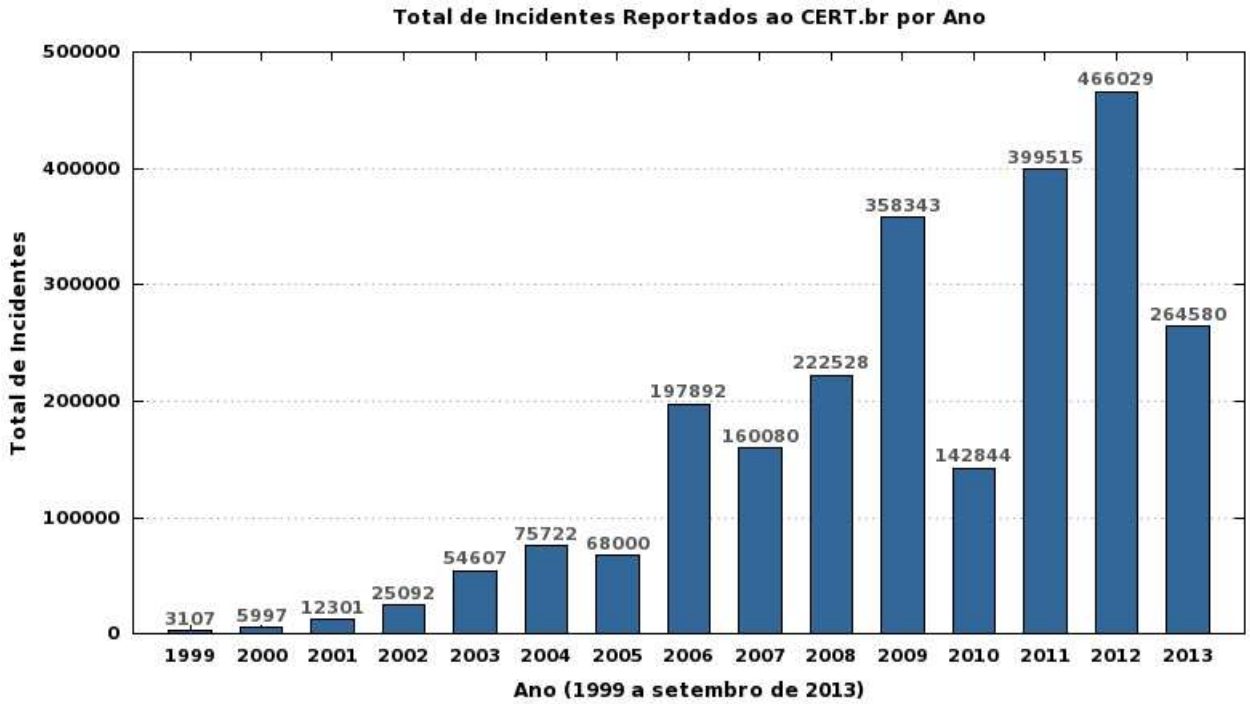

Figura 1.1. Total de incidentes reportados ao CERT.br por ano [CERT.br, 2014]

Os mecanimos de segurança não têm atendido às necessidades cada vez mais claras

dos usuários e organizações. Muitos ataques são variações de técnicas já conhecidas e, portanto, muitos devem ser capazes de serem identificados.

Como exemplo, nas Figuras 1.2 e 1.3 são mostrados respectivamente, os tipos de ataques reportados ao CERT.br de junho a setembro de 2013, e de abril a junho de 2013. Nesse caso é claro os vários tipos de ataques já conhecidos, que representam acima de 80% dos incidentes.

Figura 1.2. Incidentes reportados ao CERT.br (junho a setembro de 2013) [CERT.br, 2014]

Figura 1.3. Incidentes reportados ao CERT.br (abril a junho de 2013) [CERT.br, 2014]

Os Sistemas de Detecção de Intrusão (SDIs) são os mecanismos que permitem tais detecções, no entanto, complicadores têm impedido o real sucesso de tais sistemas. Entre esses complicadores, podemos citar as redes de alta velocidade, as quais geram

um grande volume de dados, necessidade de proteção em tempo real, redes móveis, grande número de alarmes falsos gerados por estes sistemas e a falta de bases de dados que representem a realidade atual das redes.

Esses complicadores têm fomentado a utilização de diversas técnicas pelos pesquisadores, como Mineração de Dados e Inteligência Artificial (IA), com o intuito de melhorar as taxas de detecção, diminuir os alarmes falsos e aumentar o desempenho destes sistemas. Características como detecção em tempo real e ações para prevenir a intrusão é algo cada vez mais buscado pelas pesquisas. O ideal seria um SDI com detecção em tempo real, que pudesse prevenir um ataque de forma automática, registrar um evento alertando o administrador e sem gerar alarmes falsos.

Nessa busca pela solução ideal, a importância se justifica ainda pela capacidade desses sistemas detectarem tanto ataques externos, como aqueles gerados internamente em uma organização, além de identificar ataques não conhecidos. Assim, surge um novo complicador, pois o sistema necessita criar um perfil do ambiente que seria normal, para que, a partir deste perfil consiga identificar o que seria um ataque. Outro problema é que, existem poucas bases de dados recentes. A maioria dos trabalhos utiliza bases de dados antigas, que têm recebido críticas de vários pesquisadores, como em McHugh [2000] e Shiravi et al. [2012]. De maneira geral, muitas bases de dados não são disponibilizadas por questões de segurança da organização e outras não possuem todo o tráfego capturado e identificado.

1.2 Objetivos

O objetivo geral deste trabalho é avaliar vários métodos de classificação, a partir de uma base de dados mais recente, ao identificar o tráfego malicioso de uma rede de computadores.

Especificamente, pretende-se:

- avaliar os níveis das taxas de detecções e esforço computacional de diversas técnicas de classificação, de forma que se compare as técnicas de classificação utilizadas entre si e com trabalhos relacionados;
- identificar as melhores técnicas de detecção e, em quais situações cada uma se adapta melhor;
- avaliar as taxas de detecções de intrusão utilizando um Sistema Inteligente Híbrido (SIH), de forma que seja possível melhorar o nível de detecção ao combinar mais de uma técnica de classificação;

- detectar a intrusão de um tráfego de rede antes que chegue ao destino, eliminando características desnecessárias no processo de classificação, sendo um enfoque importante para a detecção em tempo real.

1.3 Organização deste trabalho

O restante do trabalho é organizado com a estrutura a seguir. No Capítulo 2 é apresentado o referencial teórico, o qual detalha os conceitos mais pertinentes relacionados a este trabalho sobre redes de computadores, segurança em redes de computadores, sistemas de detecção de intrusão, detecção de anomalias, técnicas de classificação e trabalhos relacionados.

No Capítulo 3 é apresentada a metodologia do trabalho, descrevendo algumas bases de dados com tráfego de rede, incluindo a base de dados principal considerada neste estudo. São descritos também, os passos para a preparação dos dados através do processo de Descoberta de Conhecimento em Banco de Dados (KDD) e como foram realizados os testes de classificação e as técnicas usadas.

Em seguida, no Capítulo 4 são mostrados e discutidos os resultados obtidos. E por fim, o Capítulo 5 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Referencial Teórico

Neste capítulo, são apresentados primeiramente os conceitos mais pertinentes que permitiram o desenvolvimento deste trabalho como, redes de computadores, segurança em redes de computadores, sistemas de detecção de intrusão, detecção de anomalias e técnicas de classificação. A seguir são apresentados alguns trabalhos correlatos, que foram importantes para aprofundar sobre o tema desenvolvido.

2.1 Redes de Computadores

No início de sua existência, as redes de computadores foram usadas principalmente por pesquisadores universitários para enviar mensagens de correio eletrônico e, também, por funcionários de empresas para compartilhar recursos como impressoras. As redes de computadores acabaram com o antigo modelo de que apenas um computador atendia às necessidades do usuário ou da organização. Contudo, existem muitas redes no mundo que apresentam diferentes tipos de *hardware* e *software*, mas que precisam se comunicar entre si. Para que esse desejo se torne realidade é preciso que estabeleçam conexões entre redes, inclusive incompatíveis. Este conjunto de redes interconectadas formam uma rede mais ampla, interligada como, por exemplo, a Internet que tem alcance em nível mundial [Tanenbaum, 2011].

Duas importantes arquiteturas são o modelo de referência OSI composto por 7 camadas, desenvolvido pela ISO (*International Standards Organization*) com o objetivo de padronização internacional dos protocolos usados nas várias camadas e o modelo TCP/IP que surgiu desde a ARPANET que era uma rede de pesquisadores patrocinada pelo Departamento de Defesa dos Estados Unidos (DoD) e, que posteriormente foi melhorado e definido como um padrão

Dentre as camadas e funções dos dois modelos, a camada Internet (camada de Rede) e a camada de Transporte possuem funções similares nos dois modelos. A camada de Rede tem por função entregar pacotes (datagramas) onde eles são necessários, podendo chegar até mesmo em uma ordem diferente, importando-se com o roteamento, mesmo em caso de falhas em alguns nós (ex.: roteadores). A camada de Transporte permite a conversação dos pares dos *hosts* de origem e destino.

Na Internet, os protocolos mais comuns são o IP (*Internet Protocol*) para a camada de Rede; e o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*) para a camada de Transporte. O modelo TCP/IP não tem as camadas de sessão ou de apresentação, ao invés disso, as aplicações simplesmente incluem quaisquer funções de sessão e apresentação que forem necessárias ([Tanenbaum, 2011]).

2.1.1 Protocolos IP, UDP e TCP

O protocolo IP é o elemento que mantém a Internet unida, permitindo o roteamento de pacotes da origem para o destino sem garantias, independentemente de essas máquinas estarem na mesma rede ou de haver outras redes entre elas. Os roteadores IP encaminham cada pacote pela Internet, por um caminho de um roteador para o seguinte, até que o destino seja alcançado. No destino, a camada de Rede entrega os dados à camada de Transporte, que os oferece ao processo receptor. O protocolo IP mais comum atualmente, é o IP versão 4 (IPv4), em que um datagrama IPv4 consiste em parte de cabeçalho e uma parte de dados, conforme a Figura 2.1. O cabeçalho tem uma parte fixa de 20 bytes e uma parte opcional de tamanho variável de até 40 bytes. O tamanho total do datagrama é de 65.535 bytes, considerando o cabeçalho e o campo de dados.

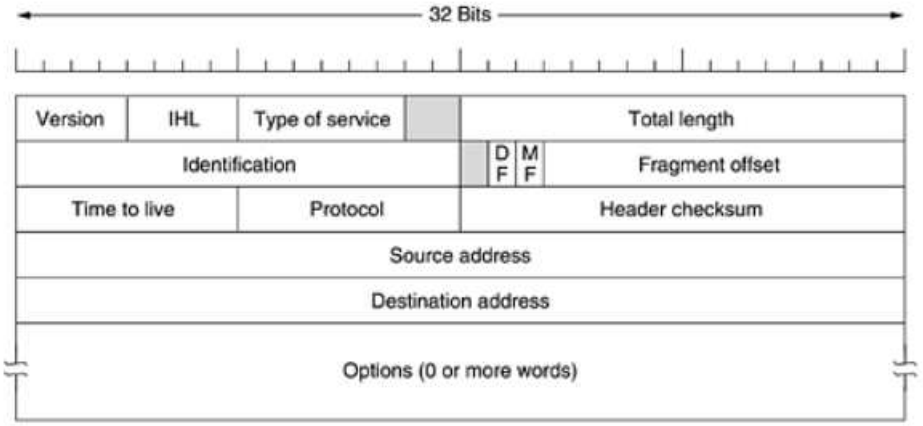

Figura 2.1. O cabeçalho IPv4 [Tanenbaum, 2011]

A Internet possui dois protocolos principais na camada de Transporte, o UDP e o TCP. O protocolo UDP é o mais simples, não orientado à conexões, capaz de enviar datagramas IP encapsulados sem que seja necessário estabelecer uma conexão. O cabeçalho de um pacote UDP é composto por 8 bytes, seguido pela carga útil, mostrado na Figura 2.2. Em seus campos temos a "Porta de origem" e "Porta de destino" que servem para identificar os pontos extremos nas máquinas de origem e destino. O "Comprimento do UDP" inclui o cabeçalho de 8 bytes e os dados. O comprimento mínimo é de 8 bytes e o máximo é de 65.515 bytes, devido ao limite de tamanho nos pacotes IP. Um campo opcional de "*Checksum* do UDP" é fornecido para gerar confiabilidade extra. Ele faz o *checksum* do cabeçalho, dos dados e de um pseudocabeçalho conceitual do IP.

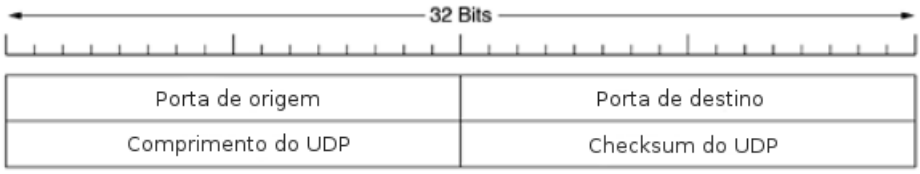

Figura 2.2. O cabeçalho UDP [Tanenbaum, 2011]

O protocolo TCP é um protocolo orientado à conexões, confiável, que permite a entrega sem erros de um fluxo de bytes, no qual é fragmentado na entrada em mensagens discretas e passa cada uma delas para a camada de Rede e, no destino, o processo TCP receptor volta a montar as mensagens recebidas no fluxo de saída. O TCP também cuida do controle de fluxo, impedindo que um transmissor sobrecarregue um receptor lento com um volume de segmentos maior do que ele pode manipular [Tanenbaum, 2011].

As entidades transmissoras e receptoras do TCP trocam dados na forma de segmentos. Um segmento TCP consiste em um cabeçalho fixo de 20 bytes e uma parte opcional, seguido por uma parte de dados. O software TCP decide qual deve ser o seu tamanho, no entanto, dois fatores restringem o tamanho deles. Primeiro, cada segmento TCP deve caber na carga útil do IP, que é de 65.515 bytes e, segundo, cada enlace geralmente tem uma unidade máxima de transferência (MTU - *Maximum Transfer Unit*) e, o segmento deve caber na MTU do transmissor e do receptor, de modo que possa ser enviado e recebido em um único pacote não fragmentado. Na prática, a MTU geralmente tem 1.500 bytes, tamanho da carga útil dos quadros Ethernet e, portanto, define o limite superior de tamanho dos segmentos [Tanenbaum, 2011].

Para um melhor entendimento, a Figura 2.3 mostra os campos do segmento TCP e, alguns deles serão analisados, pois são importantes para entender o trabalho

desenvolvido.

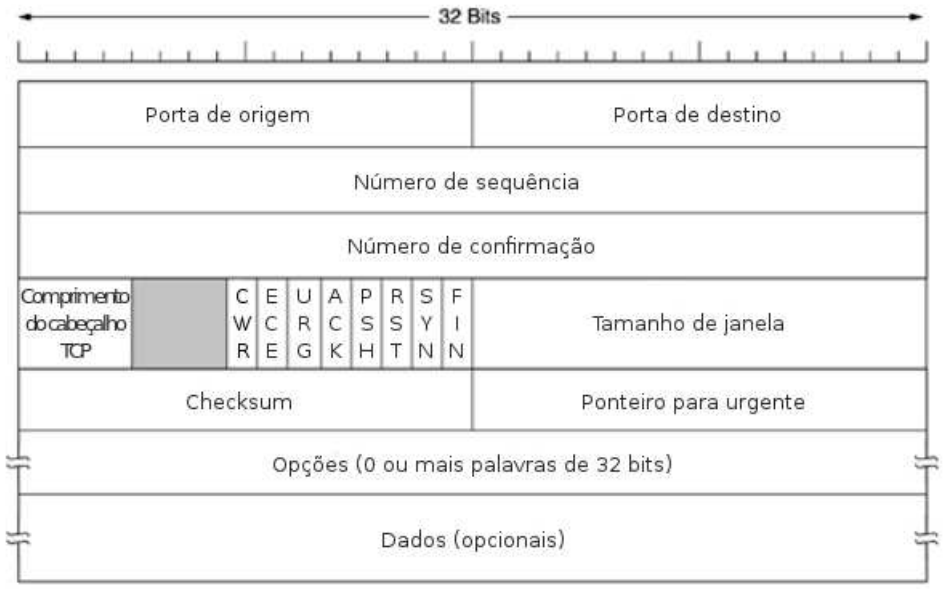

Figura 2.3. O cabeçalho TCP [Tanenbaum, 2011]

Os campos "Porta de origem" e "Porta de destino" identificam os pontos terminais da conexão, da mesma forma que do UDP. A porta TCP mais o endereço IP de seu *host* forma uma extremidade exclusiva de 48 bits. As extremidades de origem e destino juntas identificam a conexão. Os campos "Número de sequência" e "Número de confirmação" desempenham suas funções habituais, sendo o segundo a especificação do próximo byte esperado. O campo "Comprimento do cabeçalho TCP" informa quantas palavras de 32 bits existem no cabeçalho TCP. Essa informação é necessária, pelo fato do cabeçalho ter tamanho variável, devido ao campo "Opções". Em seguida, existe um campo de 4 bits que não é utilizado e 8 *flags* de 1 bit [Tanenbaum, 2011].

As *flags* CWR (*Congestion Window Reduced*) e ECE (ECN-Echo) são usadas para sinalizar congestionamento ao utilizar ECN (*Explicit Congestion Notification*). A ECE é definida para sinalizar uma mensagem ECN-Echo a um transmissor TCP para solicitar a redução de velocidade quando o receptor TCP receber uma indicação de congestionamento da rede. A *flag* CWR é usada para sinalizar "Janela de congestionamento reduzida" do transmissor TCP para o receptor TCP, de modo que ele saiba que o transmissor diminuiu a velocidade e pode parar de enviar uma mensagem ECN-Echo [Tanenbaum, 2011].

A *flag* URG é preenchida com o valor 1 se o campo "Ponteiro para urgente" estiver sendo usado. O "Ponteiro para urgente" é usado para indicar um deslocamento de bytes a partir do número de sequência atual em que os dados urgentes devem ser encontrados.

À *flag* ACK é atribuído o bit 1 para indicar que o "Número de confirmação" é válido. Isso acontece para a maioria dos segmentos. Se ACK for igual a zero, significa que o segmento não contém uma confirmação e assim o campo "Número de confirmação" é ignorado [Tanenbaum, 2011].

A *flag* PSH indica dados com PUSH, no qual o receptor é solicitado a entregar os dados à aplicação mediante sua chegada, em vez de armazená-los até que um buffer completo tenha sido recebido. A *flag* RST é utilizada para reiniciar uma conexão que tenha ficado confusa devido a uma falha no host ou por qualquer outra razão. A RST também é utilizada para rejeitar um segmento inválido ou para recusar uma tentativa de conexão. Em geral, se receber um segmento com o bit RST ativado, pode indicar algum problema [Tanenbaum, 2011].

As demais *flags*, SYN e FIN, são usadas respectivamente para o estabelecimento e encerramento de conexões. A solicitação de conexão tem SYN=1 e ACK=0 (*CONNECTION REQUEST*) e a resposta contém uma confirmação e, portanto, SYN=1 e ACK=1 (*CONNECTION ACCEPTED*). A *flag* FIN indica que o transmissor não tem mais dados para transmitir [Tanenbaum, 2011].

Os demais campos, "Tamanho de janela" indica quantos bytes podem ser enviados a partir do byte confirmado, o "Checksum" que é fornecido para aumentar a confiabilidade, conferindo o *checksum* do cabeçalho, dos dados e do pseudocabeçalho, e o campo "Opções" que foi projetado para fornecer recursos extras, que não foram previstos no cabeçalho comum, podendo ter até 40 bytes [Tanenbaum, 2011].

Esta seção teve por objetivo apresentar alguns conceitos que, apesar de serem básicos da área de Redes de Computadores, foram essenciais para o desenvolvimento deste trabalho. Isso permitiu um maior esclarecimento das informações presentes na base de dados usada como, por exemplo, campos dos protocolos, *flags* e *status* das conexões.

2.2 Segurança em Redes de Computadores

A segurança em redes de computadores pode ser definida como uma situação sem risco ou sem senso de ameaça e é caracterizada através de algumas propriedades básicas, como confidencialidade, integridade, disponibilidade, autenticação e o não repúdio.

Enquanto a confidencialidade é a propriedade de proteção do conteúdo da informação para apenas os usuários autorizados, a integridade é a propriedade de proteger a informação contra alterações por usuários não autorizados. Todavia, a disponibilidade é a propriedade que garante que a informação sempre esteja disponível para uso legí-

timo e a autenticação tem por objetivo assegurar que a informação provém das fontes enunciadas. Por fim, o não repúdio visa garantir que o autor da informação não negue ter realizado determinada ação [Douligeris & Serpanos, 2007].

Atualmente milhares de pessoas usam as redes para executar operações bancárias, realizar compras e outras atividades, surgindo pontos falhos e a segurança tornou-se um problema de grandes proporções.

A maior parte dos problemas de segurança é causada propositalmente por pessoas mal-intencionadas, que tentam obter algum benefício. Alguns dos invasores mais comuns são pessoas ressentidas com a organização a que pertencem. Os sistemas de segurança devem ser projetados tendo em vista esse fato e não apenas problemas como erros de programação [Tanenbaum, 2011].

Contudo, a ampla variedade de sistemas e tecnologias que precisam estar protegidos tende a ser um dificultador para os analistas de segurança. As empresas gastam milhões de doláres em equipamento e pessoal para proteger "coisas" que não precisam de proteção. Por outro lado, concordam que seus usuários, acidentalmente ou por intenção, são seu maior problema de segurança, pois utilizam de forma inapropriada os recursos da empresa, como má utilização de senhas de acesso e realização de *downloads* de vírus e softwares indevidos [Stallings, 2010].

Ainda assim, os dispositivos de usuário final, geralmente são os mais comuns de serem atacados, e a maioria das ameaças poderiam ser resolvidas ou bloqueadas com simples atualizações de sistemas, antivírus e *firewalls* pessoais. Desse modo, com a segurança de redes de computadores, tentamos evitar o tráfego sem autorização e entender a natureza da ameaça, a fim de assegurar as suas propriedades básicas.

2.2.1 Classificação de Ataques

Um modo de classificar os ataques de segurança é definido na RFC 2828, na qual considera os ataques como passivos e ativos. À medida que um ataque passivo tenta aprender ou utilizar informações do sistema sem afetar os recursos do mesmo, um ataque ativo tenta alterar os recursos do sistema ou afetar sua operação [Stallings, 2010].

Dois tipos de ataques passivos muito comuns são o vazamento de conteúdo de mensagens e a análise de tráfego, que podem ser exemplificados facilmente quando um e-mail ou um arquivo da organização é transferido para um terceiro não autorizado, e quando o tráfego da rede é capturado com o intuito de extrair informações relevantes do tráfego.

Nesse tipo de ataque, a detecção é muito difícil pois não envolve qualquer alteração nos dados. O tráfego de mensagem é enviado e recebido de uma maneira aparentemente normal e, o receptor e o emissor não percebem que terceiros interceptaram a mensagem ou que observaram o padrão do tráfego transmitido [Stallings, 2010]. Um exemplo muito comum, exemplificado nos incidentes do CERT.br [2014], são os ataques de "scan", que realiza varreduras em redes de computadores, com o intuito de identificar quais computadores estão ativos e quais serviços estão sendo disponibilizados por eles. É amplamente utilizado por atacantes para identificar potenciais alvos, pois permite associar possíveis vulnerabilidades aos serviços habilitados em um computador.

Uma técnica para mascarar o conteúdo, protegendo-o, é a criptografia (que vem das palavras gregas que significam "escrita secreta"), embora ainda permita a análise de algum padrão do tráfego, como origem e destino da mensagem, tamanho da mensagem, duração, frequência, etc.

Os ataques ativos, no entanto, são caracterizados por alguma modificação do fluxo de dados transmitido, ou até pela criação de um fluxo falso e podem ser subdivididos em quatro categorias: de falsidade, de repetição, de modificação de mensagens e de negação de serviço.

Um ataque de falsidade, por exemplo, sequências de autenticação são capturadas e repetidas após uma sequência de autenticação válida, ocorre quando uma entidade finge ser uma entidade diferente e, normalmente inclui uma das outras formas de ataque ativo, como o ataque de repetição. Nesse caso, a captura passiva de uma unidade de dados e sua retransmissão podem produzir um efeito não autorizado [Stallings, 2010].

Outro ataque, a modificação de mensagens, significa que alguma parte de uma mensagem legítima é alterada, ou que essas mensagens são atrasadas ou reordenadas, a fim de produzir um efeito não autorizado. Um exemplo seria a alteração de um e-mail em que concede uma permissão de acesso para determinado colaborador. Neste caso, a alteração pode simplesmente negar essa autorização ou substituir o nome autorizado por um terceiro [Stallings, 2010].

Ataques que visam especificamente o comprometimento de servidores Web ou desfigurações de páginas na Internet também são ataques ativos comuns, identificados pelo CERT.br [2014] como ataques "web". Outras categorias identificadas pelo CERT.br [2014] são as "invasões", as quais resultam no acesso não autorizado a um computador ou rede, podendo ser classificado como ativo ou passivo dependendo da ação do invasor, os "worms", que são atividades maliciosas relacionadas com o processo automatizado de propagação de códigos maliciosos na rede e as "fraudes", que englobam as notificações de tentativas de fraudes, ou seja, de incidentes em que ocorre uma

tentativa de obter vantagem.

Os ataques de negação de serviço (DoS - *Denial of Service*) impedem ou inibem o uso ou o gerenciamento normal dos recursos de comunicação, podendo ter um alvo específico ou um distúrbio de toda uma rede ou servidor, sobrecarregando os seus recursos, de modo que comprometa o desempenho ou cause travamento de determinado serviço.

Os dois tipos de ataques apresentam características opostas. Embora os ataques passivos são difíceis de detectar, medidas estão disponíveis para impedir seu sucesso. Por outro lado, os ataques ativos são difíceis de prevenir [Stallings, 2010]. Assim, um objetivo comum é impedir o sucesso dos ataques antes que aconteçam, bem como detectar e recuperar qualquer prejuízo causado, caso ocorram.

Neste trabalho, o tráfego de um ambiente de rede controlado foi considerado, de modo a detectar os ataques passivos ou ativos. Apesar da detecção não especificar cada tipo de ataque, procurou-se classificar cada tráfego transmitido na rede em uma atividade "normal" ou em um "ataque".

2.3 Sistemas de Detecção de Intrusão

A segurança em redes de computadores visa assegurar as propriedades básicas descritas anteriormente, contudo, existem muitos tipos de ataques que visam burlar essas propriedades, como abuso de *logon*, interceptação não autorizada de informações (*eavesdropping*), mascaramento de endereços de rede (*spoofing*), roubo de sessão (*hijacking*), intrusões que visam acesso não autorizado, ataques de negação de serviço (*Denial of Services* - DoS) e demais ataques no nível de aplicação como vírus, *trojans*, entre outros [Douligeris & Serpanos, 2007].

Entre esses tipos de ataques, as intrusões são algumas das formas de ameaças mais divulgadas e, que têm tido uma grande atenção da indústria e de pesquisas nos últimos anos [Stallings, 2010]. Portanto, os sistemas de detecção de intrusão têm surgido com o intuito de combater tais ameaças.

Para melhor entendermos sobre os Sistemas de Detecção de Intrusão (SDIs), devemos ainda conhecer as formas de ameaças que atingem os sistemas computacionais. Em relação à origem de ameaças, os intrusos potenciais podem ser classificados em duas categorias [Zhang et al., 2001]:

- Intrusos Externos: O ataque é lançado por um usuário de computador não autorizado, utilizando técnicas como roubo ou violações de senha, vulnerabilidades

do sistema, más configurações ou técnicas de engenharia social, com o objetivo de obter o acesso aos computadores.

- Intrusos Internos: O intruso possui um acesso legítimo ao sistema, mas utiliza algumas das técnicas mencionadas anteriormente para obter privilégios adicionais, a fim de usar de forma inapropriada ou causar danos aos dados. Esse tipo de ataque é o mais comum dentro das organizações.

Dessa forma, é necessário detectar ambos os tipos de intrusão, sendo os SDIs considerados uma segunda linha de defesa que busca pelo menos minimizar os danos ocasionados pelas intrusões. Esse tipo de sistema possui a capacidade de detectar e prevenir diversas formas de ameaças, sendo nesse caso uma de suas principais vantagens. No entanto, as técnicas e os padrões de comportamento dos intrusos estão em constante alteração e os ataques estão cada vez mais sofisticados [Stallings, 2010].

Nesse tipo de sistema, as informações coletadas de vários locais são analisadas, como um computador (SDI baseado em *host*) ou rede (SDI baseado em rede) para identificar as possíveis brechas de segurança. Em uma rede, por exemplo, o SDI pode estar localizado em um switch ou roteador e monitora o tráfego em modo promíscuo, como um *snnifer*. Os pacotes da rede são então coletados e analisados para violações de regras a partir de um algoritmo de reconhecimento de padrões. Quando essas violações são detectadas, o SDI realiza determinada ação, como uma emissão de alerta [Patcha & Park, 2007]. Caso este sistema filtre o tráfego suspeito, é considerado também um sistema de prevenção de intrusões [Douligeris & Serpanos, 2007].

De modo geral o termo IDS/IPS (*Intrusion Detection System/Intrusion Prevention System*) é bastante utilizado comercialmente para indicar que o sistema, além de realizar a detecção, realiza os filtros necessários.

Como exemplo, a Figura 2.4 mostra um cenário de rede com vários sensores de detecção de intrusão, ou seja, vários pontos de coleta de informações, dispostos em cada segmento de rede, monitorando o tráfego passante.

Um SDI ainda pode ser caracterizado pela forma como analisa as informações, seja de forma centralizada ou descentralizada. Em um SDI centralizado, o monitoramento, detecção e as informações reportadas são controladas diretamente de um local central, enquanto que em um SDI descentralizado, o monitoramento e a detecção são controlados a partir de um nó, o qual reporta hierarquicamente para um ou mais locais centrais [Al-Janabi & Saeed, 2011].

Conforme o tipo de detecção realizada, os SDIs podem ser classificados em três tipos conforme descrito em Zhang et al. [2001] e Marhusin et al. [2008].

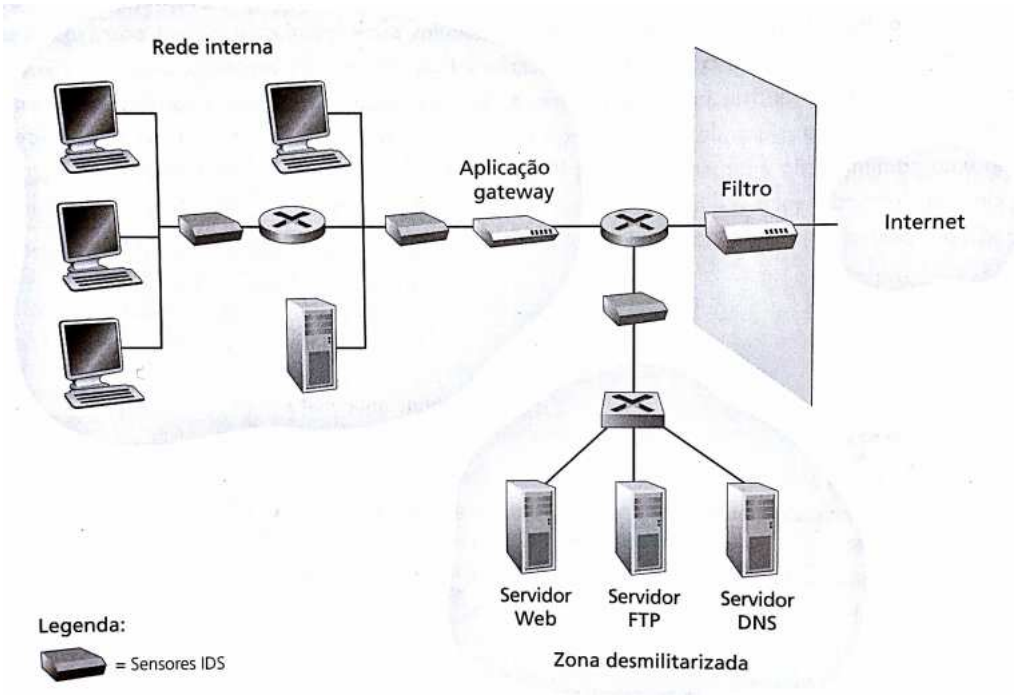

Figura 2.4. Um cenário de rede com sensores para detecção de intrusão [Kurose & Ross, 2010]

- Detecção baseada em abuso ou regras (*rule-based / misuse detection*): procura por técnicas conhecidas (assinaturas) ou vulnerabilidades do sistema.
- Detecção baseada em anomalias (*anomaly-based detection*): cria um perfil de tráfego enquanto observa o tráfego em operação normal e detecta intrusões, procurando por cadeias de pacote que são estatisticamente incomuns.
- Detecção híbrida (*hybrid detection*): combina o uso dos modelos baseados em abuso e anomalias, podendo alcançar o melhor das duas abordagens.

Comercialmente os SDIs são predominantemente baseados em detecção por assinaturas. Em contraste com os sistemas predominantes, os SDIs baseados em anomalias são bastante atrativos conceitualmente pelo fato de também poderem detectar ataques desconhecidos e do "dia zero" (falhas conhecidas, mas não corrigidas). Entretanto, esse tipo de sistema possui problemas que o impede de ser largamente adotado, tais como alto percentual de alarmes falsos, escalabilidade para maiores velocidades, proteção em tempo real, etc [Patcha & Park, 2007].

Para exemplificar, alguns sistemas comerciais como o FortiGate Firewall da Fortinet e o Aker Firewall da Aker Security Solutions possuem diversos serviços integrados, além de uma interface de gerenciamento unificada de ameaças, sendo conhecidos como

UTM (*Unified Threat Management*). O módulo de IDS/IPS desses sistemas utiliza a abordagem de detecção híbrida e são centralizados, pois todo o processamento, monitoramento e as informações reportadas são realizadas localmente.

Entre alguns dos outros serviços oferecidos por estes sistemas estão: filtros de pacotes com controle de estado, os quais analisam o conteúdo dos cabeçalhos IP e TCP/UDP, e os gateways de aplicação, conhecidos também como proxies, os quais tomam decisões com base em dados de determinada aplicação. Neste caso, vários gateways de aplicação podem ser executados nestes sistemas, como por exemplo, um proxy HTTP, FTP, Telnet, SMTP, etc [Kurose & Ross, 2010].

Outros sistemas comerciais também são verificados, como por exemplo, Symantec End Point Protection e Kaspersky End Point for Business, os quais também possuem diversos serviços integrados, como antívirus, *antispyware* e *firewall* pessoal, além do módulo de IDS/IPS que pode reportar as informações hierarquicamente para servidores remotos, funcionando portanto de forma descentralizada.

Conforme Shiravi et al. [2012], a detecção de intrusão tem atraído a atenção de muitos pesquisadores em identificar o problema cada vez maior de atividades intrusivas, especialmente, a detecção de anomalias, devido a sua característica de detectar novos ataques. Os sistemas tradicionais não têm se adaptado adequadamente aos novos paradigmas de redes, como redes sem fio, redes móveis e redes de alta velocidade. Outros fatores como constantes mudanças de tráfego e aumento do tráfego de redes também têm dificultado a construção de perfis com tráfego normal, implicando assim em mudanças fundamentais nas abordagens de detecção de intrusão [Patcha & Park, 2007].

Muitas técnicas têm sido utilizadas para a detecção de intrusão nos SDIs baseados em anomalias ou híbridos. Entre essas técnicas, podemos citar as Árvores de Decisão (DT), Redes Neurais Artificiais (RNAs), Algoritmos Genéticos (AGs), Lógica Nebulosa, Máquina de Vetores de Suporte (SVM), Sistema Imune Artificial e os Sistemas Inteligentes Híbridos (SIH), sendo este último conhecido também como *Soft Computing*, no qual combina duas ou mais técnicas distintas para resolver um problema [Rezende, 2005].

O trabalho desenvolvido realizou a detecção de intrusão baseada em anomalias, devido as suas características e necessidades. Não obstante, suas funcionalidades podem ser estendidas para uma abordagem de detecção híbrida.

2.4 Detecção de Anomalias

A detecção de anomalias também é conhecida como detecção de desvios, tendo como objetivo encontrar objetos que sejam diferentes da maioria. Esses objetos têm características que se desviam significativamente dos valores de atributos esperados ou típicos [Tan et al., 2009].

Uma diversidade de abordagens de detecção de anomalias existe em diversas áreas, as quais incluem estatística, aprendizado de máquinas e mineração de dados. Todas elas tentam identificar os objetos anômalos como sendo diferentes ou de alguma forma inconsistentes em relação aos outros objetos. Apesar desses objetos serem relativamente raros, isto não significa que eles não ocorram com frequência em termos absolutos [Tan et al., 2009]. Como exemplo, um evento que ocorre "um em um milhão" pode ocorrer milhões de vezes quando bilhões de eventos são considerados.

As causas mais comuns de anomalias são: dados de classes diferentes, variação natural e erros de coleta. Na primeira causa, um objeto anômalo pode ser diferente de outros porque é de um tipo ou classe diferente. Vários exemplos se enquadram neste caso, como alguém cometendo fraude de cartão de crédito, intrusos em redes de computadores e demais sistemas computacionais, surtos de doenças, etc. Essas anomalias são muitas vezes de considerável interesse e são o foco de detecção na área de mineração de dados, sendo que muitas são detectadas e removidas frequentemente como uma parte do pré-processamento de dados [Tan et al., 2009].

Na detecção de anomalias, algumas abordagens são verificadas, como técnicas baseadas em modelos, baseadas em proximidade ou distância ou baseadas em densidade. Devido à utilização neste trabalho, as técnicas que primeiramente constroem um modelo dos dados merecem um destaque. Neste tipo, por exemplo, um modelo da distribuição dos dados pode ser criado para avaliar os parâmetros de uma distribuição de probabilidade. Um objeto que não se enquadra muito bem dentro do modelo, é portanto considerado como anomalia se não for muito provável na distribuição. Se este modelo for um conjunto de agrupamentos, a anomalia então é um objeto que não pertence convincentemente a algum grupo. E quando um modelo de regressão é usado, uma anomalia é um objeto que esteja distante do seu valor previsto [Tan et al., 2009].

O fato das anomalias e objetos normais poderem ser vistos como duas classes distintas, técnicas de classificação podem ser consideradas para construir modelos destas duas classes, desde que alguns rótulos de classes estejam disponíveis para alguns dos objetos, permitindo que um conjunto de treinamento possa ser criado. Em outros casos, se a distribuição estatística dos dados for desconhecida ou nenhum dado de treinamento esteja disponível, técnicas que não são baseadas em modelos podem ser

usadas [Tan et al., 2009].

Outro destaque na detecção de anomalias é como diferenciar o grau no qual os rótulos de classes (anômala ou normal) estão disponíveis para pelo menos alguns dos dados. Basicamente, há três tratamentos possíveis [Tan et al., 2009].

- Detecção supervisionada de anomalias: em que as técnicas de detecção requerem a existência de um conjunto de treinamento com objetos anômalos e normais, podendo haver mais de uma classe normal e anômala.
- Detecção não supervisionada de anomalias: em que os rótulos de classes não estão disponíveis. Neste caso, o objetivo é atribuir um valor (ou um rótulo) a cada instância que reflita o grau, no qual a mesma é anômala. Deve-se observar que caso as anomalias sejam semelhantes entre si pode fazer com que todas sejam rotuladas como normais. Dessa forma, para que a detecção não supervisionada seja efetuada com sucesso, as anomalias devem ser distintas entre si.
- Detecção semi-supervisionada de anomalias: em que ocorre quando contêm os dados normais rotulados, mas não possui informações sobre os objetos anômalos. O objetivo é encontrar um rótulo ou valor de anomalia para um conjunto de objetos, usando as informações de objetos normais rotulados. Contudo, na prática, pode ser difícil encontrar um conjunto pequeno de objetos normais representativos.

Neste trabalho, pelo fato da base de dados usada possuir seus registros rotulados, permitiu a utilização de técnicas baseadas em modelos considerando uma detecção supervisionada de anomalias.

2.5 Técnicas de Classificação

As técnicas de classificação possuem o objetivo de organizar objetos em uma categoria, entre diversas, pré-definidas, sendo um problema universal que engloba muitas aplicações diferentes. Como exemplos, podemos citar a detecção de mensagens de *spam* em e-mails baseada no cabeçalho e conteúdo da mensagem, a categorização de células como malignas ou benignas baseada nos resultados de varreduras MRI (*Magnetic Resonance Imaging*) e na detecção de tráfego malicioso em redes de computadores baseada nos cabeçalhos dos PDUs (*Protocol Data Units*), entre outros [Tan et al., 2009].

Em uma tarefa de classificação, temos os dados de entrada, que são um conjunto de registros, conhecidos também como instâncias ou exemplos. Cada registro é caracterizado por uma dupla (x, y), onde x é um conjunto de atributos e y o atributo especial,

designado como rótulo da classe ou atributo alvo. O conjunto de atributos em sua maioria são discretos, podendo ser também contínuos. No entanto, o rótulo da classe, deve ser um atributo discreto. Esta é a característica que distingue a classificação da regressão, sendo esta uma técnica de modelagem preditiva onde a variável alvo a ser avaliada é contínua.

Assim, a regressão possui como objetivo aprender uma função alvo f que mapeie cada conjunto de atributos x em uma saída de valores contínuos y, enquanto a classificação é a tarefa de aprender uma função alvo f que mapeie cada conjunto de atributos x para um dos rótulos de classes y pré-determinados. A função alvo é conhecida também como modelo de classificação. Este modelo pode ser tratado como uma caixa preta que atribui automaticamente um rótulo de classe ao receber um conjunto de atributos de um registro desconhecido, conforme verificado na Figura 2.5 [Tan et al., 2009].

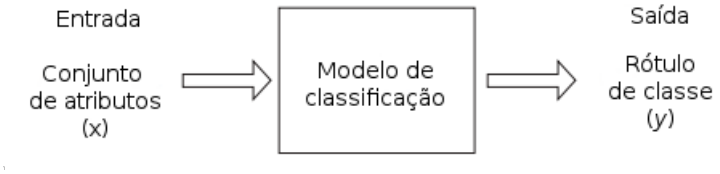

Figura 2.5. Classificação como a tarefa de mapear um conjunto de atributos x no seu rótulo de classe y [Tan et al., 2009]

A classificação de dados é um processo de duas etapas, que consiste na etapas de aprendizado ou etapa de treinamento, onde o modelo de classificação é construído e a etapa de classificação, onde o modelo é usado para predizer o rótulo da classe [Han et al., 2011].

Na primeira etapa, de aprendizado, o algoritmo de classificação constrói o classificador de acordo com a análise de um conjunto de treinamento formado pelas tuplas, conhecidas também como sendo os registros da base de dados e de seus rótulos de classes associados. A tupla x é representada por um vetor n-dimensional, $x = (x_1, x_2, ..., x_n)$.

Pelo fato de cada tupla ter sua classe conhecida, este passo também é conhecido como aprendizado supervisionado ou ainda pode ser visto como aprendizagem de um mapeamento ou função $y = f(x)$, que pode prever a classe associada ao rótulo y de uma determinada tupla x [Han et al., 2011].

De maneira geral, este mapeamento é representado em forma de regras de classificação, árvores de decisão ou fórmulas matemáticas, sendo possível ainda a utilização de outras técnicas de inteligência computacional.

Na segunda etapa, o modelo é usado para classificação. Primeiramente, a precisão da previsão do classificador é estimada e um conjunto de testes composto de suas tuplas e seus rótulos de classe é utilizado. Este conjunto não foi utilizado para a construção do classificador. A precisão de um classificador é o percentual das tuplas do conjunto de testes que foram corretamente classificadas. O rótulo de classe de cada tupla de teste é comparada com a previsão da classe apreendida para a tupla.

O presente trabalho utilizou diversas técnicas de classificação, as quais permitiram identificar para cada fluxo do tráfego de rede, da base de dados escolhida, se fez parte da classe de "ataque" ou de tráfego "normal" de rede.

2.5.1 Árvore de Decisão

Uma árvore de decisão é uma técnica de classificação simples, porém muito usada. Diante de uma série de questões, suas respostas podem ser organizadas em forma de árvore de decisão, com sua estrutura hierárquica consistindo de nodos e arestas direcionadas. A árvore tem alguns tipos de nodos: um nodo raiz que não possui arestas chegando, e zero ou mais arestas saindo; nodos internos, cada qual possuindo exatamente uma aresta chegando e duas ou mais saindo; e nodos folha ou terminais, em que cada um possui exatamente uma aresta chegando e nenhuma saindo, conforme mostrado na Figura 2.6 [Tan et al., 2009].

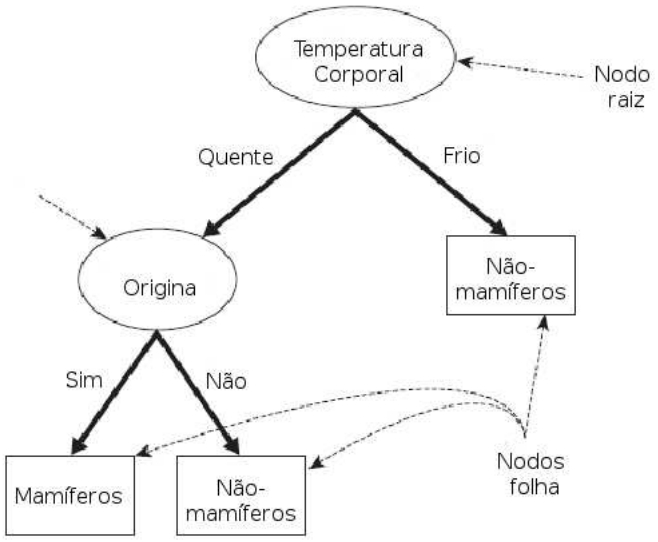

Figura 2.6. Uma árvore de decisão para o problema de classificação de mamíferos [Tan et al., 2009]

Na árvore de decisão, cada nodo folha recebe um rótulo de classe. Os nodos não terminais, que incluem o nodo raiz e outros nodos internos, contêm condições de

testes de atributos para separar registros que possuam características diferentes. Por exemplo, o nodo raiz mostrado na Figura 2.7 usa o atributo Temperatura Corporal para separar vertebrados de sangue quente dos de sangue frio.

Ao realizar uma classificação de um registro de testes, o resultado é direto, pois neste caso a árvore de decisão já foi construída. Começando do nodo raiz, aplicamos a condição de teste ao registro e seguimos a ramificação apropriada baseados no resultado do teste. Assim, o próximo passo será a análise de uma nova condição em outro nodo interno, ou levará a um nodo folha. Quando chegar a um nodo folha, o rótulo da classe é atribuído ao registro.

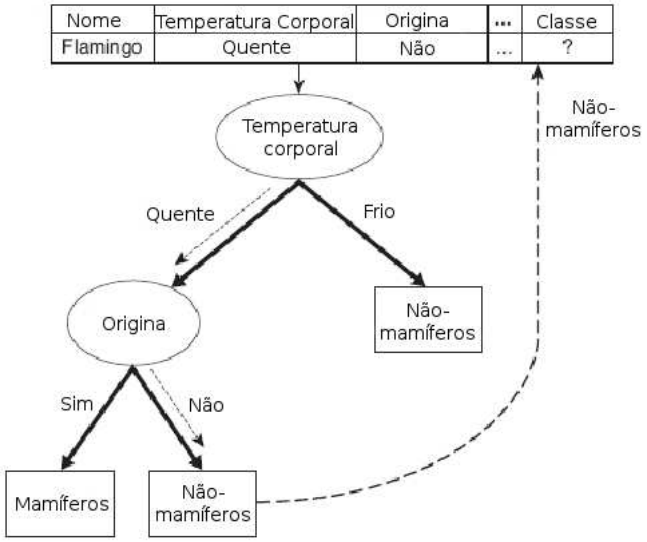

Figura 2.7. Classificação de um vertebrado sem rótulo [Tan et al., 2009]

A partir de um mesmo conjunto de atributos, é possível construir muitas árvores de decisão. Embora algumas árvores sejam mais precisas que outras, encontrar árvores ótimas é computacionalmente inviável por causa do tamanho exponencial do espaço de pesquisa. No entanto, vários algoritmos eficientes têm sido desenvolvidos para induzir a uma árvore de decisão razoalvemente precisa, não ótima, mas em uma razoável quantidade de tempo. De modo geral, utilizam um estratégia que cresce uma árvore de decisão tomando uma série de decisões localmente ótimas sobre qual atributo usar para particionar os dados [Tan et al., 2009].

Dessa maneira, cada passo recursivo do processo de crescimento da árvore deve selecionar uma condição de teste de atributo para dividir os registros em subconjuntos menores e, este processo será limitado através de uma condição de parada.

Para selecionar a melhor forma de divisão da árvore, existem várias métricas, definidas em termos da distribuição da classe dos registros antes e depois da divi-

são. Algumas das possibilidades para escolher o atributo que particiona o conjunto de exemplos em cada iteração são [Rezende, 2005]:

- aleatória: seleciona qualquer atributo aleatoriamente;
- menos valores: seleciona o atributo com a menor quantidade de valores possíveis;
- mais valores: seleciona o atributo com a maior quantidade de valores possíveis;
- ganho máximo: seleciona o atributo que possui o maior ganho de informação esperado, ou seja, o atributo que resultará no menor tamanho esperado das sub-árvores;
- e o índice Gini: mostrado na equação 2.2.

Essas métricas são geralmente baseadas no grau de impureza dos nodos filhos, sendo que, quanto menor o grau de impureza, mais distorcida é a distribuição das classes. Por exemplo, um nodo com a distribuição de classe (0,1) possui impureza zero, enquanto que um nodo com distribuição de classes uniforme (0.5, 0.5) possui a maior impureza, gerando portanto uma divisão mais balanceada [Tan et al., 2009].

As equações 2.1 e 2.2 são exemplos de métricas de impureza, onde $c-1$ indica a quantidade de classes e $p(i|t)$ a fração de registros que pertencem à classe i em um determinado nodo t.

$$Entropia(t) = -\sum_{i=0}^{c-1} p(i|t) log_2 p(i|t) \tag{2.1}$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2 \tag{2.2}$$

Como exemplo de cálculo, considere o nodo N_1 com duas classes, o qual possui 1 registro na classe A e 5 registros na classe B. Os cálculos de Entropia e Gini são respectivamente, as equações 2.3 e 2.4.

$$Entropia(N_1) = -(1/6) log_2(1/6) - (5/6) log_2(5/6) = 0.650 \tag{2.3}$$

$$Gini(N_1) = 1 - (1/6)^2 - (5/6)^2 = 0.278 \tag{2.4}$$

Para determinar o quão bem uma condição de teste é executada, precisamos comparar o grau de impureza do nodo pai (antes da divisão) com o grau de impureza dos nodos filhos (após a divisão). Quanto maior a diferença, melhor a condição de teste. O ganho, Δ, é um critério que pode ser usado para determinar a qualidade de uma divisão, conforme a equação 2.5 [Tan et al., 2009].

$$\Delta = I(pai) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j) \tag{2.5}$$

Onde $I(t)$ é a medida de impureza de um determinado nodo t, N é o número total de registros no nodo pai, k é o número de valores dos atributos e $N(v_j)$ é o número de registros associados ao nodo filho v_j. De modo geral, muitos algoritmos de árvore de decisão escolhem uma condição de testes que maximia o ganho Δ, e pelo fato de $I(pai)$ ser o mesmo para todas as condições de teste, maximizar o ganho é equivalente a minimizar as medidas de média ponderada de impureza dos nodos filhos.

Quando a entropia é usada como métrica de impureza na equação 2.5, a diferença na entropia é conhecida como Ganho de Informação, Δ_{info}, a qual mede a eficácia de um atributo em classificar os dados de treino. A escolha do atributo mais eficaz, o qual mais reduz a entropia, faz com que a tendência seja a de gerar árvores menos profundas, possuindo menos nodos e ramificações.

Um algoritmo para indução de árvores de decisão é um exemplo de algoritmo de estrutura conhecido como TDIDT (*Top-Down Induction of Decision Trees*). Este algoritmo utiliza a estratégia "dividir para conquistar", em que um problema complexo é divido em subproblemas mais simples.

No algoritmo que constrói uma árvore de decisão, o passo principal é a escolha de um atributo para rotular o nó atual da árvore. Deve-se escolher o atributo que tenha o maior poder de discriminação entre as classes para os exemplos no nó atual. No processo de indução da árvore, o algoritmo escolhe um atributo, estende a árvore adicionando um ramo para cada valor do atributo, passa os exemplos para as folhas e associa para cada nó folha uma determinada classe quando seus exemplos são todos da mesma classe, senão repete o processo.

No entanto, a árvore construída podem estar ajustada demais aos dados de treinamento necessitando, portanto, de um do procedimento conhecido como "poda de árvore", o qual pode ser executado para reduzir o seu tamanho, melhorando assim a sua capacidade de generalização e não ocorrendo o fenômeno *overfitting*, no qual o erro de treinamento diminui, mas o erro de teste aumenta [Tan et al., 2009].

Na literatura existem vários algoritmos para indução de árvores de decisão como, NB*Tree*, REP *Tree*, *Random Tree*, J48, entre outros, que foram utilizados neste trabalho através do software *Weka*, com o intuito de realizar a classificação do tráfego malicioso de rede. Os diversos algoritmos escolhidos foram baseados em resultados de trabalhos relacionados, os quais permitiram realizar análises e comparações no Capítulo 4.

2.5.2 Redes Neurais Artificiais

A área das redes neurais artificiais, também conhecida como conexionismo ou sistemas de processamento paralelo e distribuído, é caracterizada por sistemas que, em algum nível, relembram a estrutura do cérebro humano e, constitui uma alternativa à computação algorítmica convencional por não ser baseada em regras [Braga et al., 2007].

O cérebro humano consiste principalmente de células nervosas chamadas neurônios, conforme mostrado na Figura 2.8, que são ligados com outros através de fios de fibra chamados axônios. Estes são usados para transmitir impulsos nervosos de um neurônio para outro, sempre que os neurônios forem estimulados. Um neurônio é conectado aos axônios de outros neurônios através de dentritos, que são extensões do corpo da célula do neurônio e, o ponto de contato entre um dendrito e um axônio é chamado de sinapse. O cérebro humano aprende quando se altera a força da conexão da sinapse entre neurônios em repetidos estímulos pelo mesmo impulso [Tan et al., 2009].

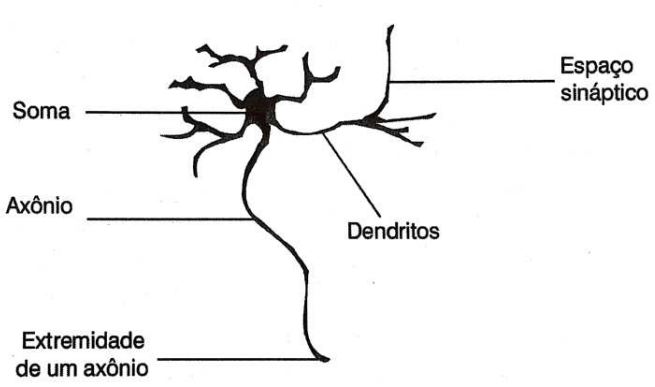

Figura 2.8. Componentes do neurônio biológico [Braga et al., 2007]

As Redes Neurais Artificiais (RNAs) são sistemas compostos por unidades de processamento simples (neurônios artificiais), que calculam determinadas funções matemáticas (normalmente não-lineares), sendo dispostas em uma ou mais camadas e

interligadas por um grande número de conexões. Na maioria dos modelos essas conexões estão associadas a pesos, os quais armazenam o conhecimento adquirido pelo modelo e servem para ponderar a entrada recebida por cada neurônio da rede [Braga et al., 2007].

Pelo fato de possuirem a capacidade de aprender por meio de exemplos e generalizar a informação aprendida torna-se o atrativo principal da solução de problemas por meio das RNAs. A generalização está associada à capacidade de a rede aprender por meio de um conjunto reduzido de exemplos e então apresentar respostas coerentes para dados não conhecidos. Elas, portanto, são capazes de atuar como mapeadores universais de funções multivariáveis, além de terem a capacidade de auto-organização e processamento temporal, permitindo assim a solução de problemas complexos [Braga et al., 2007].

O primeiro modelo artificial de um neurônio biológico foi fruto do trabalho de Warren McCulloch e Walter Pitts em 1943. Sua descrição matemática resultou em um modelo com n terminais de entrada (dentritos) que recebem os valores $x_1, x_2, ..., x_n$ (que representam as ativações dos neurônios) e apenas um terminal de saída y (representando o axônio). Para representar o comportamento das sinapses, os terminais de entrada do neurônio tem pesos acoplados $w_1, w_2, ..., w_n$ conforme mostra a Figura 2.9 [Braga et al., 2007].

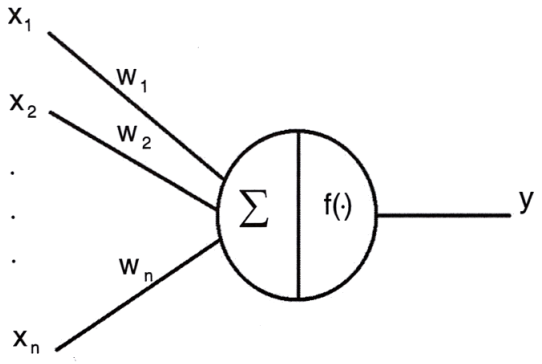

Figura 2.9. Neurônio de McCulloch e Pitts, no qual $\sum$ representa a soma ponderada das entradas e $f(\cdot)$ a função de ativação [Braga et al., 2007]

O primeiro módulo do neurônio artificial, denominado regra de propagação, executa uma soma ponderada das entradas x_i multiplicadas pelos seus pesos sinápticos w_i, associados a cada entrada. O segundo módulo, denominado função de ativação, aplica uma função ao resultado da regra de propagação. Essa função é responsável por gerar a saída y do neurônio a partir dos valores dos vetores de peso e de entrada.

Apesar das diversas funções de ativações possíveis, como as mostradas nas figuras 2.10 e 2.11, os neurônios individuais possuem capacidade computacional limitada. Entretanto, um conjunto de neurônios artificiais conectados na forma de uma rede é capaz de resolver problemas complexos. Assim, diversas formas de organização dos neurônios são possíveis, denominadas arquiteturas.

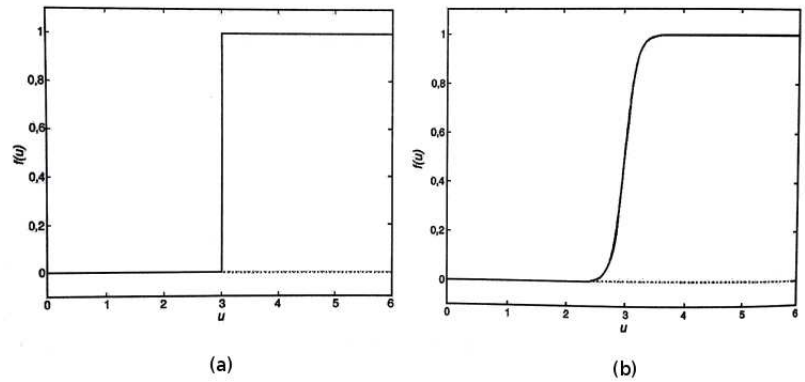

Figura 2.10. Função de ativação degrau (a) e sigmoidal (b) [Braga et al., 2007]

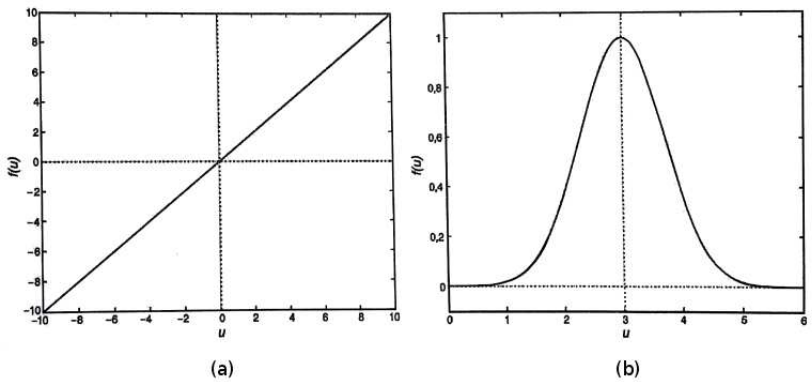

Figura 2.11. Função de ativação linear (a) e gaussiana (b) [Braga et al., 2007]

Por exemplo, uma arquitetura simples mostrada na Figura 2.12(a), corresponde a uma rede neural única alimentada para a frente (*feedforward*), a qual é capaz de resolver problemas multivariáveis de múltiplas funções acopladas, mas ainda com algumas restrições de complexidade, devido ao fato de possuir apenas uma camada. Outra estrutura, mostrada na Figura 2.12(b), possui uma camada adicional, conferindo à RNA uma maior capacidade computacional e universalidade na aproximação de funções contínuas. Essas estruturas por não possuirem recorrência (realimentação), são consideradas estáticas, ao contrário das estruturas mostradas na Figura 2.13, as quais

são geralmente utilizadas em problemas que envolvam processamento temporal, como previsão de eventos futuros.

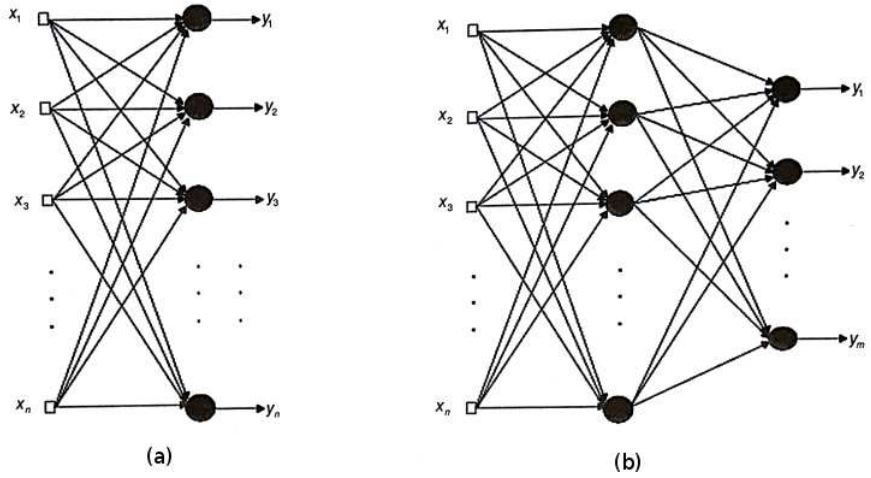

Figura 2.12. Rede *feedforward* de uma camada (a) e de duas camadas (b) [Braga et al., 2007]

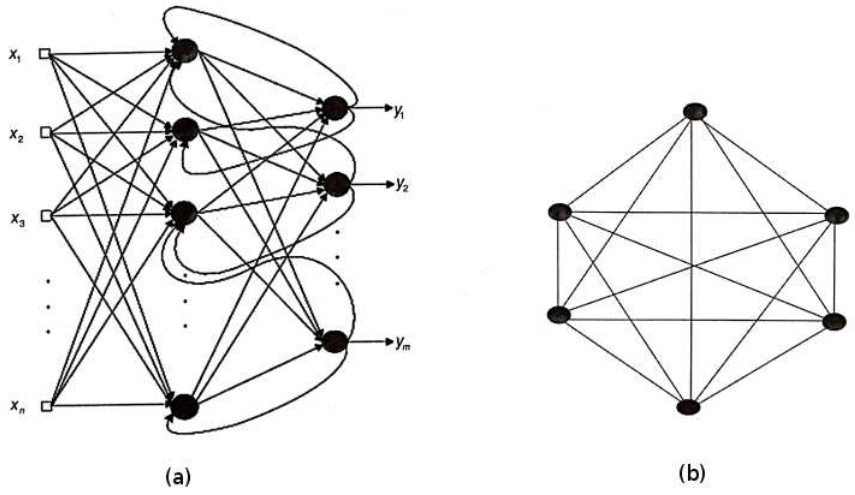

Figura 2.13. Rede com recorrência entre saídas e camada intermediária (a) e rede com recorrência auto-associativa (b) [Braga et al., 2007]

Entre os tipos de RNAs mais utilizadas podemos citar as redes de múltiplas camadas MLP (*Multilayer Perceptrons*), as quais são formadas por uma camada de entrada, uma ou mais camadas intermediárias (ocultas) e uma camada de saída. Outro tipo, constituída geralmente de três camadas são as redes RBF (*Radial Base Functions*). Nessa arquitetura os neurônios da camada intermédiaria utilizam funções de bases radiais como, por exemplo, a função gaussiana mostrada na Figura 2.11(b), em que a

ativação de um neurônio pode ser função da distância entre seus vetores de entrada e de peso, ao contrário das redes MLP que utilizam geralmente funções de base sigmodais [Braga et al., 2007].

Tanto as redes RBF como as redes MLP são aproximadoras universais de funções, sendo portanto equivalentes. Entretanto, existem diferenças como a partição do espaço de padrões de entrada realizada pela camada intermediária do modelo. Enquanto uma rede RBF constrói aproximadores locais através de hiperlipsóide, isto é, apenas as regiões do espaço de entrada que apresentam dados de treinamento terão resposta da rede. As redes MLP, por outro lado, particionam o espaço de entradas através de hiperplanos [Braga et al., 2007].

Assim, as redes MLP possuem maior capacidade de generalização e necessitam, em geral, de menos dados para treinamento que as redes RBF. Estas no entanto, necessitam de mais dados para generalizar a informação e são utilizadas geralmente em problemas bem definidos, que possuem dados dinâmicos e que necessitam de aprendizado continuado como, por exemplo, previsões de séries temporaris e operações financeiras [Braga et al., 2007].

Para a resolução de problemas, a escolha de uma arquitetura correta é fundamental, a qual depende de vários fatores como complexidade do problema, dimensionalidade do espaço de entrada, características dinâmicas ou estáticas, conhecimento a *priori* do problema e representatividade dos dados. Todavia, o aprendizado das RNAs, conforme a abordagem conexionista, não é adquirido através de regras explícitas, mas através do ajuste de intensidades das conexões entre os neurônios [Braga et al., 2007].

A etapa de aprendizado de uma RNA consiste em um processo iterativo, o qual ajusta os pesos das conexões, que guardam ao final do processo, o conhecimento que a rede adquiriu do ambiente externo. Este conhecimento está relacionado com a melhoria de desempenho da rede segundo algum critério preestabelecido. Por exemplo, o erro quadrático médio da resposta da rede é utilizado como critério de desempenho pelos algoritmos de correção de erros, que são utilizados no treinamento das RNAs. Dessa forma, espera-se que o erro diminua à medida que o aprendizado prossiga. Genericamente, o valor do vetor de pesos $w(t+1)$ no instante $t+1$ pode ser escrito conforme a Equação 2.6, onde $w(t)$ e $w(t+1)$ representam os valores dos pesos nos instantes t e $t+1$, respectivamente, e $\Delta w(t)$ é o ajuste aplicado aos pesos. É justamente a forma como calcula este ajuste $\Delta w(t)$ que diferencia os diversos algoritmos para treinamento das RNAs. Esses algoritmos podem ser agrupados em dois paradigmas principais: aprendizado supervisionado e aprendizado não-supervisionado [Braga et al., 2007].

$$w(t+1) = w(t) + \Delta w(t) \tag{2.6}$$

Para o trabalho desenvolvido, o qual consiste em realizar tarefas de classificação, o aprendizado supervisionado é o aplicado. Este paradigma implica, necessariamente, a existência de um supervisor, o qual é responsável por estimular as entradas da rede por meio de padrões de entrada e observar a saída calculada, comparando-a com a saída desejada.

Assim, os pesos são ajustados aproximando a saída da rede da saída desejada. Esse processo é realizado para cada padrão de entrada, em que a rede tem sua saída corrente comparada com a saída desejada pelo supervisor, que fornece informações sobre a direção de ajuste dos pesos. O aprendizado supervisionado pode ser implementado tanto *offline* como *online* sendo que, no primeiro, os dados do conjunto de treinamento não mudam e, ao encontrar uma solução para a rede, esta deve permanecer fixa. Se novos dados forem adicionados, um novo treinamento, inclusive com os dados anteriores, deve ser realizado novamente. No segundo, por sua vez, o conjunto de dados muda continuamente e a rede precisa estar em um contínuo processo de adaptação [Braga et al., 2007].

De maneira geral, o aprendizado supervisionado é aplicado em problemas em que se deseja obter um mapeamento entre padrões de entrada e saída como, por exemplo, problemas de classificação, em que padrões são apresentados às entradas e as classes correspondentes são apresentadas às saídas da rede durante o processo de aprendizado. Neste exemplo, a rede deverá adaptar os seus pesos de forma a mapear as relações entre padrões de entrada e classes correspondentes de saída, tendo por base os dados do conjunto de treinamento. Outros exemplos que utilizam o aprendizado supervisionado são os problemas de aproximação e previsão [Braga et al., 2007].

Os mais conhecidos algoritmos para aprendizado supervisionado são a regra delta, conhecido também como LMS (*Least Mean Square*), e sua generalização para as redes de múltiplas camadas, o algoritmo *back-propagation*, que baseia-se na heurística por correção de erro, o qual é retropropagado da camada de saída para as camadas intermediárias da RNA [Braga et al., 2007].

Neste trabalho, o aprendizado supervisionado foi considerado, pois consistiu em realizar a classificação do tráfego de rede. Assim, a partir de uma amostra dos dados utilizando o software *Weka*, as RNAs RBF e MLP foram usadas para classificar os registros. Além disso, as RNAs MLP também foram empregadas em conjunto com a técnica de Algoritmo Genético, a qual permitiu melhorar os resultados.

2.5.3 Algoritmos Genéticos

Outra técnica de classificação bastante utilizada são os Algoritmos Genéticos, que fazem parte da Computação Evolutiva (CE), a qual baseia-se em mecanismos evolutivos encontrados na natureza. Os AGs são inspirados nos modelos biológicos, os quais fundamentam-se na Genética de Mendel e na Teoria da Evolução de Darwin [Artero, 2009]

Os AGs são métodos estocásticos de busca que procuram soluções próximas das melhores. Em seu processo, inicialmente é formada uma população com um conjunto aleatório de indivíduos gerando possíveis soluções de um problema. Logo após, é iniciado o processo evolutivo, onde cada indivíduo é avaliado sobre sua adaptação ao ambiente. Assim, uma porcentagem dos membros mais adaptados é mantida e a restante é descartada. Dentre os indivíduos mantidos, ainda pode ocorrer a mutação e o cruzamento, conhecido como recombinação genética, gerando mais membros [Britt et al., 2007].

Todo esse processo, conhecido como reprodução, é repetido até encontrar uma solução satisfatória. De maneira geral, busca-se resolver problemas para os quais ainda não existe um algoritmo conhecido. Contudo, esta técnica também é bastante utilizada para aprimorar outras técnicas, como por exemplo as Redes Neurais Artificiais, contribuindo para encontrar os seus parâmetros iniciais.

Como os Algortimos Genéticos utilizam conceitos inspirados na Genética, vários de seus termos são utilizados, tais como [Artero, 2009]:

- Genes: correspondem a uma representação de algum parâmetro de interesse, de acordo com algum alfabeto, podendo usar valores inteiros, reais e cadeias de caracteres, contudo, o mais comum é utilizar apenas os valores 0 e 1 de um alfabeto binário.

- Cromossomos: correspondem a uma cadeia de genes que representam cada indivíduo da população. Como exemplos de representação, temos respectivamente: $cromossomo1 = \{4.8; 3.2; 5.3\}$ utilizando números reais, $cromossomo2 = \{3; 2; 6; 7\}$ utilizando inteiros e $cromossomo3 = \{0001101010101\}$ utilizando números binários.

- Indivíduos: um indivíduo corresponde a um cromossomo, sendo usados para representar as soluções a serem encontradas em um problema.

- População: formada por um conjunto de indivíduos que irão competir pela sobrevivência e pela reprodução, com o intuito de perpetuar suas características.

- Geração: uma geração corresponde a uma população em determinado período, sendo portanto, os valores dos indivíduos obtidos em uma dada iteração.
- Função de aptidão: a função de aptidão é usada para medir a habilidade do indivíduo para sobreviver e se reproduzir. Nos AGs são representadas por funções matemáticas, que conforme o problema a ser resolvido, devem ser maximizadas ou minimizadas.

Os algoritmos genéticos possuem algumas características que os diferem de outras técnicas convencionais, trabalhando com uma codificação do conjunto de parâmetros e com uma população, e não apenas com um valor e com os próprios parâmetros. O processamento realizado por um algoritmo genético é realizado em algumas etapas, conforme podemos verificar na Figura 2.14. O algoritmo mostrado na Figura 2.15 apresenta o diagrama geral das etapas de um algoritmo genético.

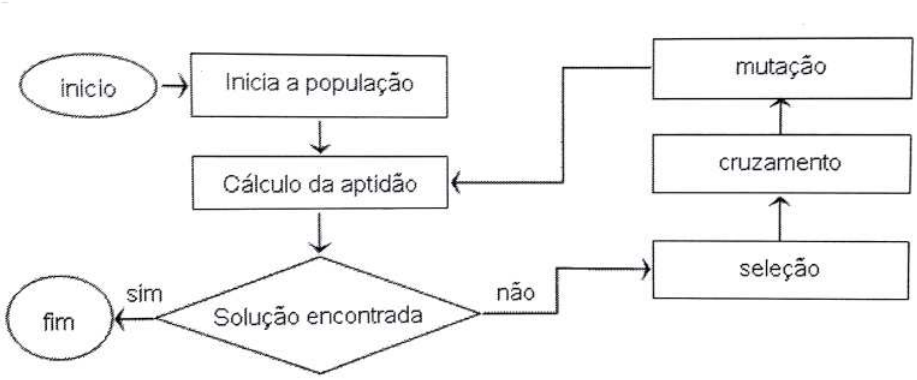

Figura 2.14. Etapas do Algoritmo Genético [Artero, 2009]

A etapa de seleção é usada para escolher os indivíduos para a reprodução, podendo ser realizada de diferentes maneiras, porém, as propostas mais comuns são a seleção aleatória, a seleção por torneio e a seleção usando a roleta. Na seleção aleatória, dois indivíduos da população são escolhidos ao acaso. Na seleção por torneio, o melhor deles, conforme a função de aptidão, será selecionado para a reprodução. Este processo será repetido para encontrar também o segundo indivíduo a participar do cruzamento. Na modalidade de seleção por roleta, as habilidades de todos os indivíduos da população são calculadas e os valores obtidos corresponderão, proporcionalmente, aos setores de uma roleta, conforme o exemplo da Figura 2.16. Dessa maneira, a probabilidade dos setores maiores serem selecionados será maior ao girar a roleta, contudo os setores menores ainda podem ser selecionados, apesar da menor probabilidade [Artero, 2009].

Algoritmo 1 Algoritmo Genético

```
1: T = 0;
2: Gerar População Inicial P(0);
3: para todo cada indivíduo i da população atual P(t) faça
4:    Avaliar aptidão do indivíduo i;
5: fim para
6: enquanto Critério de parada não for satisfeito faça
7:    t = t + 1;
8:    Selecionar população P(t) a partir de P(t − 1);
9:    Aplicar operadores de cruzamento sobre P(t);
10:   Aplicar operadores de mutação sobre P(t);
11:   Avaliar P(t);
12: fim enquanto
```

Figura 2.15. Algoritmo de alto nível de um AG [Rezende, 2005]

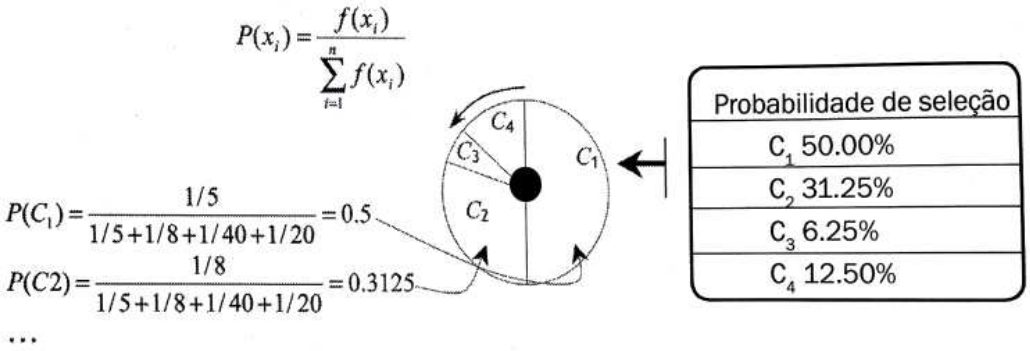

Figura 2.16. Roleta com tamanhos de setores proporcionais à chance do cromossomo ser selecionado [Artero, 2009]

Logo, a etapa de cruzamento (*crossover*) consiste no cruzamento entre o material genético de dois indivíduos. No caso de utilizar valores binários, são usados pontos de corte aleatoriamente posicionados entre os bits. Como exemplo temos a Figura 2.17, onde as linhas verticais indicam os pontos de corte [Artero, 2009].

Os *bits* do C_1 antes do primeiro corte são copiados para o $filho_1$, e após o corte, os *bits* são copiados para o $filho_2$. Similarmente, os *bits* do C_2 antes do primeiro corte são copiados para o $filho_2$, e após o corte, os *bits* são copiados para o $filho_1$. Com dois cortes, os *bits* antes do primeiro corte e após o segundo corte são enviados para o $filho_1$, enquanto que os *bits* entre o primeiro e segundo cortes são enviados para o $filho_2$ [Artero, 2009].

É possível também ocorrer múltiplos cortes quando os *bits* antes do primeiro não são trocados. Neste caso, a partir do primeiro corte, os *bits* são trocados até que seja

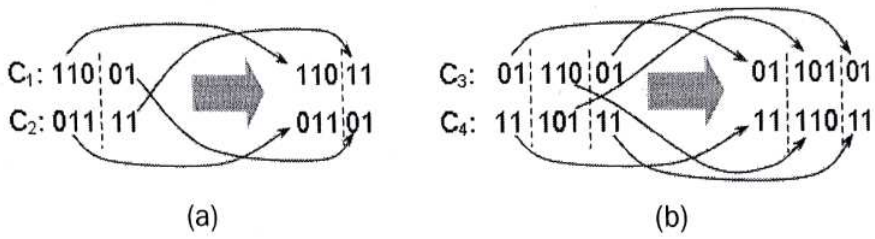

Figura 2.17. Cruzamentos com um corte (a) e com dois cortes (b) [Artero, 2009]

encontrado o próximo corte. A seguir, os *bits* não serão trocados até o próximo corte, sendo este processo repetido para os demais cortes [Artero, 2009].

Apesar dos AGs terem sido propostos originalmente para operar com valores binários, as perdas de precisão decorrentes da representação dos números reais em binário, levaram ao surgimento de propostas para se operar diretamente com valores reais. Neste cenário, o cruzamento pode ser efetuado usando uma média aritmética ou geométrica entre os pais $\{x_1, x_2, ..., x_n\}$ e $\{y_1, y_2, ..., y_n\}$. Dois métodos conhecidos que funcionam desta maneira são: Cruzamento Aritmético Único (*Single Arithmetic Crossover*) e Cruzamento Aritmético Simples (*Simple Arithmetic Crossover*) [Artero, 2009].

No primeiro, toma-se um valor k, aleatório e inteiro, responsável por determinar a posição dos genes a serem cruzados e também um valor real aleatório, $\alpha \in [0, 1]$, gerando os filhos: $\{x_1, x_2, ..., \alpha y_k + (1-\alpha)x_k, ..., x_n\}$ e $\{1_1, y_2, ..., \alpha x_k + (1-\alpha)y_k, ..., y_n\}$ [Artero, 2009]. Por exemplo, considerando $\alpha = 0.4$, $k = 3$, $Pai_1 = \{0.5, 1.0, 1.5, , 2.0\}$ e $Pai_2 = \{0.2, 0.7, 0.2, 0.7\}$, resultará nos filhos: $filho_1 = \{0.5, 1.0, (0.4)(0.2) + (0.6)(1.5), 2.0\} = \{0.5, 1.0, 0.98, 2.0\}\}$ e $filho_2 = \{0.2, 0.7, (0.4)(1.5)+(0.6)(0.2), 0.7\} = \{0.2, 0.7, 0.72, 0.7\}\}$.

Para o Cruzamento Aritmético Simples, de forma similar, também toma-se um valor aleatório k, que vai determinar a posição dos genes a serem cruzados e $\alpha \in [0, 1]$, com os filhos obtidos: $\{x_1, x_2, ..., \alpha y_k + (1-\alpha)x_k, ..., (1-\alpha)x_n, \}$ e $\{1_1, y_2, ..., \alpha x_k + (1-\alpha)y_k, ..., (1-\alpha)y_n\}$ [Artero, 2009]. Por exemplo, considerando $\alpha = 0.4$, $k = 3$, $Pai_1 = \{0.5, 1.0, 1.5, 2.0\}$ e $Pai_2 = \{0.2, 0.7, 0.2, 0.7\}$, resultará nos filhos: $filho_1 = \{0.5, 1.0, (0.4)(0.2) + (0.6)(1.5), (0.4)(0.7) + (0.6)(2.0)\} = \{0.5, 1.0, 0.98, 1.48\}\}$ e $filho_2 = \{0.2, 0.7, (0.4)(1.5) + (0.6)(0.2), (0.4)(2.0) + (0.6)(0.7)\} = \{0.2, 0.7, 0.72, 1.22\}\}$.

Consequentemente, em algumas situações, o cromossomo pode ser modificado em

algumas de suas partes, assumindo novas características, que não foram herdadas de seus pais. Este cenário, o qual consiste a etapa de mutação, possui uma probabilidade baixa de ocorrer, sendo que ao utilizar números binários, consiste na inversão de alguns bits, enquanto que no caso de números reais, somam-se pequenos valores aleatórios (positivos ou negativos). Se a representação for feita usando cadeias de caracteres, alguns caracteres podem ser alterados aleatoriamente [Artero, 2009].

Outro aspecto importante de um AG é a definição de seus parâmetros que influenciam fortemente seu desempenho. Entre esses parâmetros, o tamanho da população afeta o desempenho global e a eficiência dos AGs. O desempenho neste caso pode cair se a população for pequena, pois ela fornece uma pequena cobertura do espaço de busca do problema, enquanto que uma população grande fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais em vez de globais. No entanto, grandes populações necessitam de mais recursos computacionais ou que o algoritmo trabalhe por um período de tempo muito maior [Rezende, 2005].

A taxa de cruzamento quanto maior, mais rapidamente novas estruturas são introduzidas na população. Contudo, se for muito alta, estruturas com boas aptidões poderão ser retiradas mais rapidamente que a capacidade da seleção em criar melhores estruturas. Se a taxa for muito baixa, a busca pode estagnar [Rezende, 2005].

A taxa de mutação baixa previne que a busca fique estagnada em sub-regiões do espaço de busca. Possibilita ainda que qualquer ponto do espaço de busca seja antigido, entretanto, uma taxa muito alta faz com que a busca seja essencialmente aleatória [Rezende, 2005].

Outro parâmetro importante é o intervalo de geração, o qual controla a porcentagem da população que será substituída para a próxima geração. Com um intervalo grande, a maior parte da população será substituída podendo ocasionar perdas de estruturas de alta aptidão. Um intervalo pequeno, no entanto, pode fazer com que o algoritmo se torne muito lento [Rezende, 2005].

Por fim, o parâmetro critério de parada é importante para terminar a execução de um AG. Existem diversos critérios como, por exemplo, dado um número de gerações, quando a aptidão média ou do melhor indivíduo não melhorar mais ou quando as aptidões dos indivíduos de uma população se tornarem muito parecidas. Ao conhecer a resposta máxima da função objetivo, é possível utilizar este valor como critério de parada.

Neste trabalho, a técnica de Algoritmos Genéticos foi usada para encontrar os parâmetros iniciais das RNAs MLP. Desse modo, mesmo que não tenha encontrado uma solução ótima, os valores encontrados permitiram melhores resultados, conforme

apresentados no Capítulo 4, do que simplesmente utilizar as RNAs de forma isolada.

2.5.4 Máquina de Vetores de Suporte

Outra técnica de classificação que tem recebido atenção é a Máquina de Vetores de Suporte (*Support Vector Machine - SVM*), que possui seus fundamentos na teoria de aprendizagem estatística e tem mostrado resultados empíricos promissores em várias aplicações práticas, como reconhecimento de dígitos escritos à mão e categorização de textos [Tan et al., 2009].

Esta técnica funciona bem com dados de alta dimensionalidade, além de representar o limite de decisão usando um subconjunto dos exemplos de treinamento, conhecidos como os "vetores de suporte". A SVM pode ser treinada para resolver problemas tanto linearmente separáveis como os não linearmente separáveis [Tan et al., 2009].

Para um melhor entendimento, será apresentado o conceito de "hiperplano de margem máxima". Conforme a Figura 2.18, um conjunto de dados contendo exemplos que pertencem a duas classes, são mostrados. Neste exemplo, o conjunto de dados é linearmente separável, pois é possível encontrar um hiperplano em que todos os quadrados fiquem de um lado do hiperplano e os círculos do outro lado. Contudo, muitos hiperplanos são possíveis e o classificador deve escolher apenas um para representar seu limite de decisão [Tan et al., 2009].

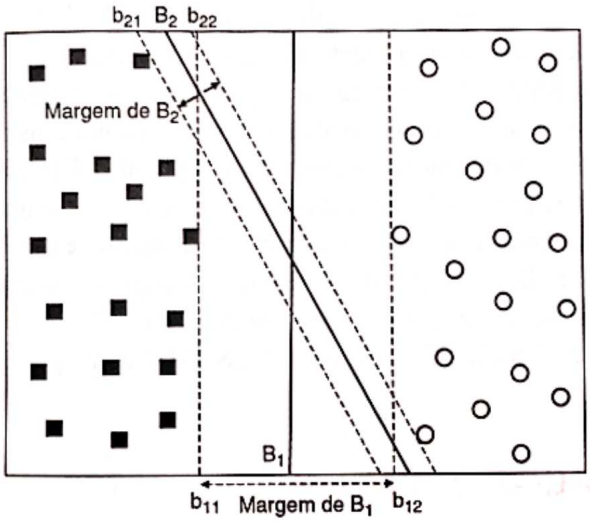

Figura 2.18. Margem de um limite de decisão [Tan et al., 2009]

Consequentemente, limites de decisão com margens grandes tendem a ter erros de generalização melhores do que aqueles com margens pequenas. Desse modo, caso a margem seja pequena, qualquer perturbação mínima no limite de decisão pode ter um

impacto bastante significativo sobre sua classificação. Assim, classificadores que produzem limites de decisão com margens pequenas são mais susceptíveis a *overfitting* de modelo e tendem a generalizar de forma fraca sobre exemplos não vistos anteriormente [Tan et al., 2009].

Conforme mostrado em Tan et al. [2009], a capacidade de um modelo linear está inversamente relacionada a sua margem. Modelos com margens pequenas possuem capacidades maiores, sendo mais flexíveis e podendo conter mais conjuntos de treinamento, ao contrário dos modelos com margens grandes. Entretanto, de acordo com o princípio de aprendizagem estatístico SRM (minimização do risco estrutural), à medida que a capacidade aumenta, o limite do erro de generalização também aumentará. Dessa forma, projetar classificadores lineares que maximizem as margens dos seus limites de decisão é desejável para assegurar que seus erros de generalização do pior caso sejam minimizados. Como exemplo, temos o classificador SVM linear que funciona dessa maneira [Tan et al., 2009].

Todavia, o SVM linear pode ser extendido para a classificação de dados não linearmente separáveis, utilizando para isso um método conhecido como a abordagem da "margem flexível". Esse método é capaz de descobrir um limite de decisão que seja tolerável a pequenos erros de treinamento. Para fazer isso, o algoritmo de descoberta em SVM deve considerar o balanceamento entre a largura da margem e o número de erros de treinamento cometidos pelo limite de decisão linear [Tan et al., 2009].

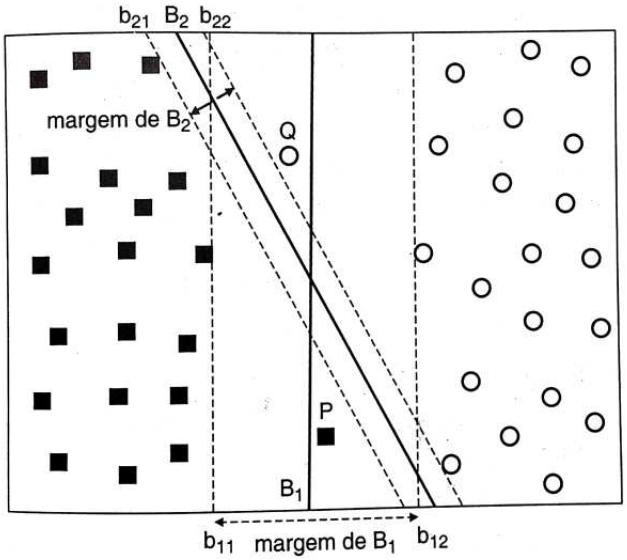

Figura 2.19. Limite de decisão de SVM para caso não separável [Tan et al., 2009]

Como exemplo, a Figura 2.19 mostra um conjunto de dados que é semelhante à

Figura 2.18, exceto que possui dois novos exemplos, P e Q. Neste caso, embora o limite de decisão B_1 classifique erroneamente os novos exemplos e B_2 classifica corretamente, isto não significa que B_2 seja um limite de decisão melhor do que B_1 porque os novos exemplos podem corresponder a ruídos nos dados de treinamento. Assim, B_1 ainda deve ser preferido com relação a B_2 por que possui uma margem maior, sendo portanto, menos susceptível a *overfitting* [Tan et al., 2009].

Neste trabalho, a técnica SVM foi utilizada para a classificação do tráfego de rede através do software *Weka*. Embora não tenha alcançado os melhores resultados em relação as demais técnicas de classificação, serviu para análise e comparação com outros trabalhos.

2.5.5 Sistemas Inteligentes Híbridos

Os Sistemas Inteligentes Híbridos (SIHs) são resultados da combinação de duas ou mais técnicas distintas, sendo pelo menos uma delas de Inteligência Artificial (IA). A vantagem desses sistemas está no fato de que em alguns casos, apenas uma técnica apresenta deficiências na resolução de problemas e, então, a combinação de mais de uma técnica pode combinar as vantagens, superando as desvantagens que cada técnica apresenta individualmente [Rezende, 2005].

É comum a utilização de técnicas como Redes Neurais artificiais em conjunto com técnicas de Computação Evolutiva (CE) como, por exemplo, Algoritmos Genéticos e técnicas que utilizam conhecimento como Lógica Nebulosa. Dessa forma, a combinação de várias técnicas pode levar a uma solução mais robusta e eficiente [Rezende, 2005].

As RNAs oferecem a possibilidade de aprendizado a partir de dados, modelagem empírica do comportamento humano, técnicas de aproximação universal, métodos de extração de conhecimento a partir dos dados, memória associativa, entre outras. Contudo, as RNAs exigem um tempo de treinamento elevado, além de não possuirem um mecanismo explicativo, nem um mecanismo automático e eficiente para auxiliar o desenvolvedor no projeto da rede. Enquanto isso, a CE apresenta outras vantagens como, não necessidade de um conhecimento matemático aprofundado do problema considerado, tolerância a ruídos e dados incompletos, além da otimização de parâmetros de funções objetivos complexas [Rezende, 2005].

Assim sendo, os paradigmas da Figura 2.20 podem ser vistos como complementares, podendo também propiciar soluções similares e equivalentes. Consequentemente, um método pode ser aplicado para melhorar o desempenho do outro, como por exemplo [Rezende, 2005]:

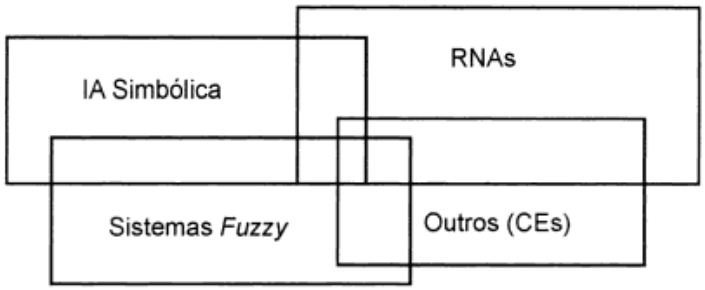

Figura 2.20. Paradigmas e métodos se sobrepõem como técnicas de resolução de problema [Rezende, 2005]

- CE pode ser usada para inicializar a estrutura de uma RNA a fim de acelerar o treinamento e melhorar a generalização;
- RNAs e CE podem ser empregadas para aprender regras (de lógica nebulosa ou não);
- RNAs podem ser usadas para refinar regras (de lógica nebulosa ou não);
- CE pode ser usada para ajustar a topologia e os pesos de uma RNA;
- RNA e CE podem ser usadas como parte de uma máquina de raciocínio simbólico.

Diante disso, os SIHs têm sido classificados em um esquema composto por 3 classes, conforme a Figura 2.21. Essa classificação considera fatores como: funcionalidade, arquitetura de processamento e requisitos de comunicação. O primeiro tipo é "substituição de função" (*funcion replacing*), a qual utiliza uma técnica para implementar a função de outra técnica. Nesse caso, não há nenhuma funcionalidade extra ao sistema, pois apenas tenta superar alguma limitação da técnica principal ou otimizar sua execução.

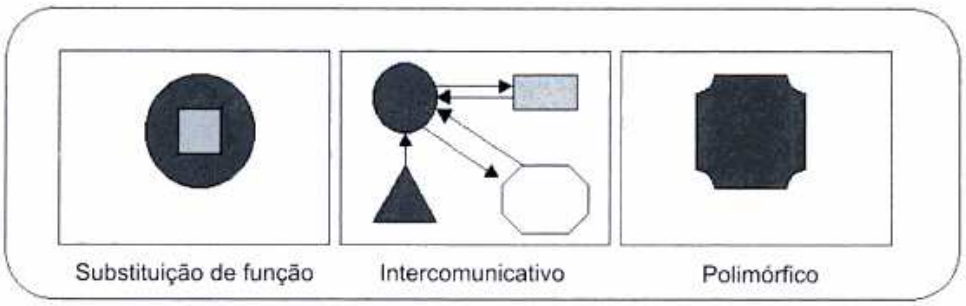

Figura 2.21. Classificação de sistemas híbridos inteligentes [Rezende, 2005]

A segunda classificação são dos "híbridos intercomunicativos" (*intercommunicating hybrids*), que procuram resolver problemas complexos que possam ser divididos em várias subtarefas independentes. O sistema híbrido é formado por vários módulos independentes, nos quais cada um usa uma técnica inteligente para resolver uma das subtarefas do problema principal. Por fim, o "híbridos polimórficos" é quando uma única técnica é adaptada para realizar uma tarefa inerente a uma outra técnica [Rezende, 2005].

Neste trabalho, um SIH intercomunicativo foi utilizado de modo que o AG inicializasse a estrutura das RNAs MLP, definindo melhores parâmetros, os quais permitiram uma melhor generalização dos resultados.

2.6 Trabalhos Relacionados

Nos últimos anos muitas pesquisas têm sido desenvolvidas sobre os SDIs e apesar dos avanços, ainda existe a necessidade de melhorias devido às mudanças de comportamento dos usuários e organizações e da evolução das redes e sistemas computacionais. Dessa forma são apresentados alguns trabalhos relacionados.

Em Tavallaee et al. [2009] é proposta a base de dados NSL-KDD que traz avanços em relação à base KDDCUP'99. Além de realizar uma análise detalhada da base KDDCUP'99, procurou eliminar alguns de seus problemas, que têm sido alvos de críticas a mais de uma década [McHugh, 2000], utilizando as técnicas do software *Weka* como Árvores de Decisão e Redes Neurais Artificiais (RNAs) para gerar os resultados.

O trabalho apresentado em Li [2010] utiliza 10% do conjunto de dados KDDCUP'99 através de RNAs com uma estrutura flexível e modular, utilizando AG para ajustar os seus parâmetros e, consequentemente, melhores taxas de detecção e menores taxas de falso positivos foram atingidas.

Em Pachghare & Kulkarni [2011] é realizado um estudo comparativo de vários tipos de algoritmos de Árvore de Decisão, considerando métricas como tempo de treinamento, precisão e tamanho da árvore. Neste estudo foi utilizado uma pequena parte do conjunto de dados da KDDCUP'99, sendo que a Árvore de Decisão do tipo J48*graft* obteve os melhores resultados.

No trabalho apresentado em Devaraju & Ramakrishnan [2011], o desempenho de detecção de intrusão utilizando a KDDCUP'99 é comparado através de vários classificadores de RNAs, além de comparar o conjunto de dados completo com uma versão reduzida da KDDCUP'99, mostrando que a utilização do conjunto reduzido se justifica devido ao melhor desempenho em relação à utilização do conjunto de dados completo.

O trabalho de Silva [2011] desenvolve métodos de detecção de intrusão com técnicas de IA como AG, RNAs e Lógica Nebulosa, criando assim um SIH que realizou uma análise detalhada e melhorias das taxas de detecção com a base de dados NSL-KDD, além de fazer comparações com os resultados obtidos em Tavallaee et al. [2009].

Em Shiravi et al. [2012], foi proposta uma nova base de dados, a ISCX 2012, que foi gerada a partir de tráfego real em um ambiente monitorado, fornecendo um guia com características importantes para criação de novas bases de dados, além de realizar comparações com outras como a KDDCUP'99, DEFCOM e CAIDA.

A partir da base ISCX 2012, alguns trabalhos recentes têm surgido, como em Mzila & Dube [2013] onde aplica o classificador k-NN (k-*Nearest Neighbor*) e elimina alguns atributos com características ligadas ao destino, permitindo que o modelo seja utilizado para detecção em tempo real.

Por fim, em Sallay et al. [2013] utilizou a ISCX 2012 e propôs um classificador para redes de alta velocidade baseado em treinamento *online* com o algoritmo de aprendizado de máquina (SVM), utilizando várias estratégias de aprendizado e modos de execução. Como conclusão, o trabalho apresenta um bom resultado entre a precisão e o tempo de processamento.

Capítulo 3

Métodos

Neste capítulo, são apresentadas as etapas realizadas para o desenvolvimento do trabalho. Desde o início, procurou-se aprofundar sobre o tema proposto através de informações comprovadas e publicadas na literatura, com o intuito de indentificar as principais necessidades e ser um guia para o progresso do trabalho. Foi identificada a necessidade de buscar uma base de dados com tráfego de rede mais atual e com características importantes para a detecção de intrusões.

Consequentemente, foi preciso preparar a base de dados escolhida. Para tal, utilizou-se o processo de Descoberta do Conhecimento em Base de Dados (KDD), o qual eliminou inconsistências, definiu o formato dos dados, selecionou uma amostra, para que então fosse possível aplicar a etapa de mineração dos dados.

Assim, escolheu-se as principais técnicas e definiu-se como aplicá-las na classificação do tráfego de rede. Nessa etapa, utilizou-se como base diversas técnicas empregadas em trabalhos correlatos, as quais obtiveram resultados promissores.

Com o intuito de melhorar o desempenho e resultados das classificações, buscou-se também realizar testes com menos atributos de entrada, considerados desnecessários para uma detecção em tempo real, além de utilizar um Sistema Inteligente Híbrido (SIH), o qual combinou mais de uma técnica para a classificação do tráfego.

3.1 Bases de Dados com Tráfego de Rede

Nesta seção são apresentadas duas das bases de dados que têm sido mais utilizadas nos trabalhos relacionados e uma base mais recente que é a chave deste trabalho desenvolvido.

Desde 1999, a base de dados KDDCUP'99 tem sido amplamente utilizada em pesquisas sobre métodos de detecção de intrusão. Este conjunto de dados foi construído

a partir de dados capturados pelo programa de avaliação de SDIs da DARPA'98 e preparada pelo MIT *Lincoln Labs*. Este conjunto de dados possui quarenta e dois atributos que caracterizam o comportamento de tráfego de rede separados em 4 classes de ataques e cerca de 4.900.000 de registros [KDD-Cup, 1999].

No entanto, uma das principais deficiências dessa base é o grande número de registros redundantes. Assim, em Tavallaee et al. [2009] foi proposta a base de dados NSL-KDD, a qual é uma versão melhorada da KDDCUP'99 [NSL-KDD, 2009], mas que ainda possui problemas da KDDCUP'99 como dados antigos e que não representam a realidade atual das redes de computadores.

A base de dados ISCX 2012 foi apresentada em Shiravi et al. [2012], a qual consiste de porções de tráfego de rede que incluem várias informações além de seus *payloads* completos. A ISCX 2012 foi gerada a partir de um tráfego real em um ambiente controlado, pelo *Information Security Centre of Excellence* (ISCX) na *University of New Brunswick* do Canadá, conforme a Figura 3.1, possuindo 2.071.657 registros que foram rotulados, indicando para cada conexão o tipo de tráfego.

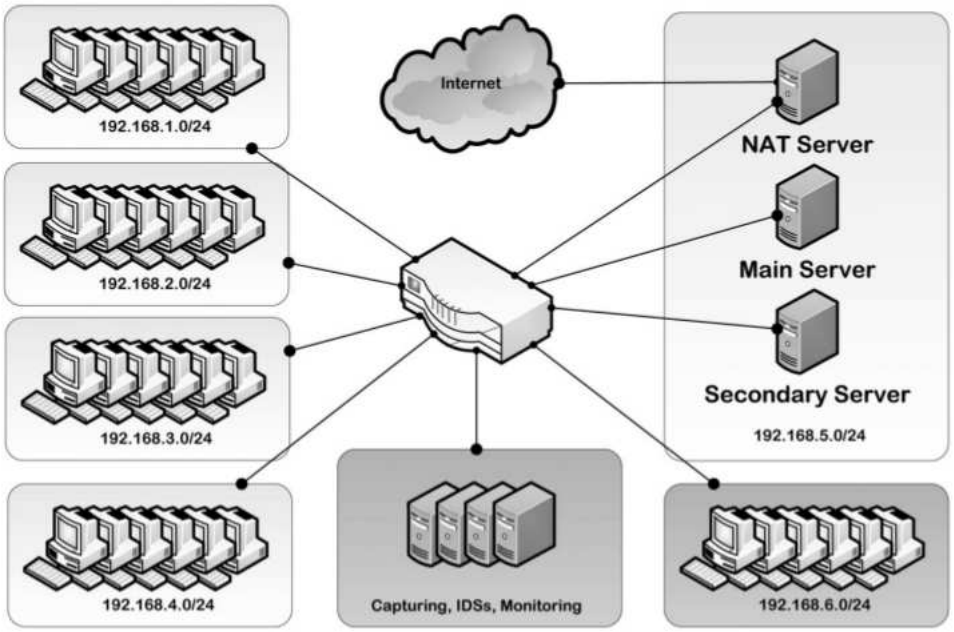

Figura 3.1. Arquitetura de rede utilizada na ISCX 2012 [Shiravi et al., 2012]

Assim, esta base foi elaborada com os principais objetivos e características para que seja aceitável, eliminando deficiências encontradas em outras bases. Portanto, os objetivos a seguir servem como um guia para a geração de futuras bases para análise de tráfego de rede [Shiravi et al., 2012]:

- rede realista e tráfego real: todos os dados foram capturados de uma rede real, e não apenas através de simuladores;
- conjunto de dados rotulados: todo o conjunto de dados é rotulado, identificando para cada conexão se é um tráfego normal de rede ou se é um ataque;
- captura total de interações de rede: todas as interações de rede são capturadas, como duração de cada conexão, endereços/portas de origem e destino, etc;
- captura completa de dados: o tráfego de rede foi capturado utilizando um *snnifer* e salvando os arquivos em formato .pcap para análises posteriores;
- diversos cenários de intrusões: são realizados vários testes de intrusões com diferentes técnicas, como ataque de negação de serviço (*DoS*), força bruta, etc.

Esta base de dados é composta por 17 atributos mostrados na Tabela 3.1 e o atributo "tag" indica se o tráfego é normal ou um ataque.

Tabela 3.1. Lista de atributos da base ISCX 2012

appName	protocolName
totalSourceBytes	totalDestinationBytes
totalSourcePackets	totalDestinationPackets
sourcePayloadAsBase64	destinationPayloadAsBase64
sourceTCPFlagsDescription	destinationTCPFlagsDescription
source	destination
sourcePort	destinationPort
startDateTime	stopDateTime
direction	tag

3.2 Preparação da Base de Dados

Nesta seção, são apresentados os passos de desenvolvimento do trabalho como o processo de Descoberta de Conhecimento em Banco de Dados (KDD - *Knowledge Discovery in Database*), ferramentas utilizadas e a preparação da base ISCX 2012 apresentada por Shiravi et al. [2012]. Foram aplicadas etapas do processo de KDD, mostradas na Figura 3.2, como limpeza, integração, transformação, seleção e mineração dos dados [Han et al., 2011].

Na etapa de limpeza de dados, através da linguagem *Python*, foram removidos os dados inconsistentes, como separações por vírgulas que estavam excedentes nos registros, que criavam campos adicionais na base original, além de reorganizar colunas

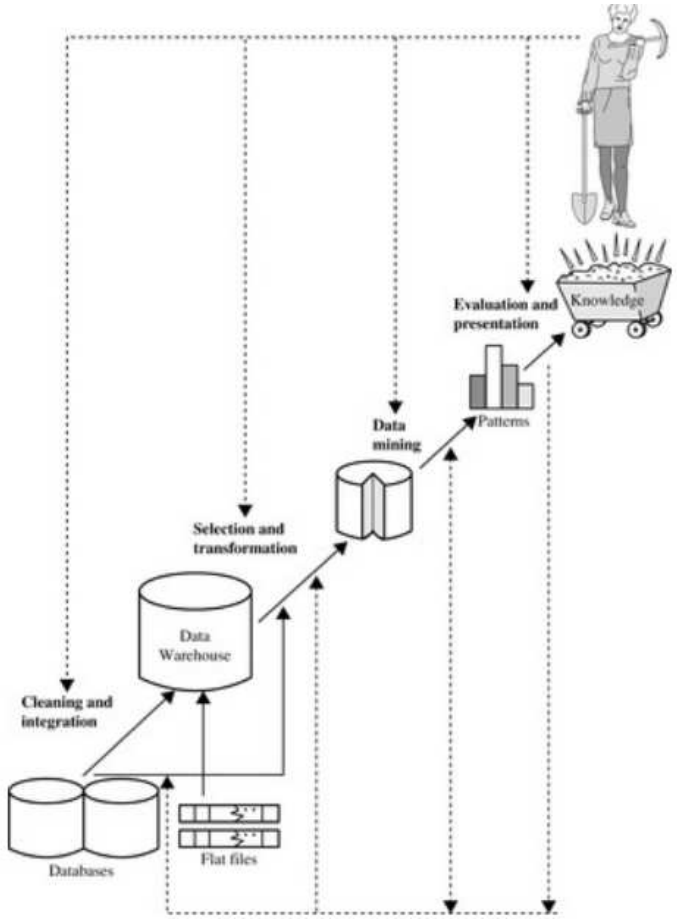

Figura 3.2. Processo de Descoberta de Conhecimento em Banco de Dados (KDD) [Han et al., 2011]

que estavam fora de ordem nos arquivos. O código utilizado para esta etapa é mostrado no Anexo A.

Em seguida, na integração dos dados, foram agrupados os dados dos diversos dias de coleta de informações em um único arquivo. O objetivo dessa integração deve-se ao fato de que, para cada dia de coleta de informações, uma determinada técnica de ataque foi utilizada conforme podemos verificar em Shiravi et al. [2012] e descrita na Tabela 3.2. Assim, ao agruparmos os dados de todos os dias, aumentamos o grau de dificuldade que nosso trabalho terá para detectar se é um "ataque" ou uma atividade "normal".

Na etapa de transformação dos dados, os arquivos rotulados da base ISCX 2012 foram formatados com a extensão .arff, formato padrão do software *Weka*, ferramenta esta bastante utilizada para mineração de dados, como classificação de padrões, técnicas de clusterização e associação [Weka, 2013]. O código utilizado para formatar os arquivos é mostrado no Anexo B.

Tabela 3.2. Descrição do tráfego de rede por dia da base ISCX 2012

Dia	Descrição
Sexta	Atividade normal. Sem tráfego malicioso.
Sábado	Atividade normal. Sem tráfego malicioso.
Domingo	Infiltração na rede com origem interna mais atividade normal.
Segunda	Ataque de negação de serviço (*DoS*) HTTP mais atividade normal.
Terça	Ataque de negação de serviço distribuido (*DDoS*) usando IRC Botnet.
Quarta	Atividade normal. Sem tráfego malicioso.
Quinta	Ataque de força bruta SSH mais tráfego normal.

Foram gerados os arquivos para análise de cada dia, excetuando da sexta-feira, dia em que não houve ataque registrado e todo o tráfego foi normal. O formato final dos arquivos é mostrado na Tabela 3.3.

Tabela 3.3. Formato dos arquivos .arff

```
@relation 'iscx2012'
@attribute 'duration' real
@attribute 'protocolName' {'tcp','udp','icmp'}
@attribute 'sourceTCPFlagsDescription' {'N/A','SPA', 'SRPA','FSPA', 'FRPA', 'FSRPA', 'FA',
'FRA', 'FPA',
'FSA', 'SRA', 'RPA', 'R', 'S', 'A','RA','PA', 'SA', 'SR', 'FSRA', 'FPU', 'FSPU', 'FSRPU'}
@attribute 'destinationTCPFlagsDescription' {'N/A','SPA', 'SRPA','FSPA', 'FRPA', 'FSRPA', 'FA',
'FRA',
'FPA', 'FSA', 'SRA', 'RPA', 'R', 'S', 'A','RA', 'PA','SA', 'FSRA'}
@attribute 'totalSourceBytes' real
@attribute 'totalDestinationBytes' real
@attribute 'source' {'0','1', '2', '3', '4', '5', '6'}
@attribute 'destination' {'0','1', '2', '3', '4', '5', '6'}
@attribute 'sourcePort' numeric
@attribute 'destinationPort' numeric
@attribute 'totalSourcePackets' real
@attribute 'totalDestinationPackets' real
@attribute 'direction' {'L2R','L2L','R2R','R2L'}
@attribute 'tag' {'Normal','Attack'}
@data
1811,tcp,SPA,SPA,148502,3111115,2,0,2328,51413,2219,2755,L2R,Normal
45,udp,N/A,N/A,6479,108260,2,0,58040,37642,80,340,L2R,Normal
3746,tcp,SPA,SPA,1119896,13181228,2,0,2115,51413,17066,24650,L2R,Normal
?,tcp,S,?,990,0,2,0,2130,5555,15,0,L2R,Attack
7,tcp,SPA,FSPA,659,2765,4,0,2377,80,7,7,L2R,Normal
0,icmp,N/A,N/A,74,0,0,2,0,0,1,0,R2L,Normal
```

Após a transformação dos dados relacionados a cada dia de coleta, os dados foram agrupados em um único arquivo, com 2.071.657 registros e, os *payloads* foram desconsiderados, concentrando-se então nos demais atributos, conforme a Tabela 3.4.

Portanto, no arquivo agrupado com todos os registros em formato .arff foram realizados outros passos através do software *Weka*, conforme descrito abaixo. Em seguida foi realizada a etapa final de mineração de dados através das técnicas de classificação.

- Normalização dos dados: os dados foram normalizados entre os valores [0:1] através do filtro *unsupervised-attribute-normalize*, a fim de considerar os atributos

Tabela 3.4. Atributos dos arquivos gerados em formato .arff

Nº	Atributo	Descrição
1	duration	Refere-se à subtração dos campos stopDateTime e startDateTime em segundos
2	protocolName	Identifica se o protocolo é 'tcp', 'udp' ou 'icmp'
3	sourceTCPFlagsDescription	Identifica as flags encontradas na comunicação do fluxo de dados de origem
4	destinationTCPFlagsDescription	Identifica as flags encontradas na comunicação do fluxo de dados de destino
5	totalSourceBytes	Número de bytes enviados da fonte para o destino
6	totalDestinationBytes	Número de bytes enviados do destino para a fonte
7	source	0 para endereços de origem da Internet, e 1 a 6 referentes às sub-redes internas 192.168.x.0/24 onde x varia entre 1 e 6
8	destination	0 para endereços com destino à Internet, e 1 a 6 referentes às sub-redes internas 192.168.x.0/24 onde x varia entre 1 e 6
9	sourcePort	porta de origem do protocolo de transporte
10	destinationPort	porta de destino do protocolo de transporte
11	totalSourcePackets	Número de pacotes enviados da fonte para o destino
12	totalDestinationPackets	Número de pacotes enviados do destino para a fonte
13	direction	Direção do fluxo de pacotes
14	tag	Rótulo informando se o tráfego é 'Normal' ou 'Attack'

com a mesma importância.

- Embaralhamento dos dados: foi utilizado o filtro *unsupervised-instance-randomize* para embaralhar os dados, uma vez que ao agrupar todos os dados (registros), determinadas técnicas de intrusão utilizadas ficaram localizadas em registros próximos devido ao dia de coleta.

- Amostragem aleatória simples: devido ao grande volume de dados da base ISCX 2012, foi retirada uma amostra aleatória simples de 212866 (cerca de 10,28%) registros do conjunto de dados através do filtro *unsupervised-instance-resample*. O objetivo dessa transformação é reduzir o conjunto de dados sem sacrificar sua integridade e permitir a utilização das técnicas de classificação. Para definir o valor da amostra n, utilizou-se a Equação 3.1, considerando um nível de confiança c de 99% (que define o valor de $Z_c = 2,575$), um erro de estimativa $E = 0,001$, e valores de $p = 0,033206$ e $(1 - p) = 0,966794$, a proporção de registros de ataques e normais, respectivamente, do conjunto de dados completo [Larson & Farber, 2004].

$$n = p(1-p)\left(\frac{Z_c}{E}\right)^2 \tag{3.1}$$

Na próxima seção será apresentado como foram realizados os testes de classificação e as técnicas empregadas.

3.3 Testes de Classificação

Para a realização dos testes de classificação, duas abordagens foram escolhidas. A primeira, conhecida como abordagem *Holdout*, na qual os dados são particionados em dois conjuntos distintos, um para a etapa de treinamento e criação do modelo e o outro para a etapa de testes. Esta técnica é recomendada quando se tem uma grande quantidade de dados [Han et al., 2011]. Assim, na abordagem *Holdout* foram considerados 20% dos dados da amostra para o conjunto de treinamento, similarmente como em Silva [2011] e Tavallaee et al. [2009] e, os 80% restantes para o conjunto de testes.

A segunda abordagem é conhecida como Validação Cruzada (*Cross-Validation*), sendo um método mais eficiente que o *Holdout*, em especial quando se tem uma quantidade menor de dados. Neste método, para o trabalho desenvolvido, os dados da amostra foram particionados em 10 partes e, a cada execução, uma das partições foi escolhida para teste, enquanto que as outras foram usadas para treinamento.

Ao utilizar o software *Weka*, as duas abordagens foram consideradas, escolhendo diversas técnicas, principalmente de árvores de decisão, redes neurais artificiais e aprendizado de máquina. Os critérios para a escolha destas técnicas foram baseados em alguns trabalhos relacionados como Tavallaee et al. [2009], Pachghare & Kulkarni [2011], Silva [2011] e Bukhtoyarov & Semenkin [2012].

Após a realização dos testes com o software *Weka*, foi utilizada a linguagem *Python* para unir as técnicas de Algoritmo Genético (AG) e Redes Neurais Artificiais (RNAs), gerando um Sistema Inteligente Híbrido (SIH). Os códigos utilizados pelo SIH são mostrados nos anexos C e D.

Para este novo teste, a mesma amostra dos dados foi considerada e então realizadas as etapas de treinamento e testes através da abordagem *Holdout*. Primeiramente foram considerados nos testes 13 entradas (atributos) para as RNAs MLP e uma saída (atributo alvo), conforme Tabela 3.4 já apresentada.

O objetivo foi utilizar o AG para buscar uma estrutura para a RNA que forneça uma taxa de acerto maior, como em Silva [2011] e Li [2010]. O teste consistiu na combinação da técnica de AG com as RNAs MLP, realizando permutações em vários parâmetros das RNAs, buscando melhores valores para a taxa de aprendizado, *mo-*

mentum, *decay*, quantidade de neurônios ocultos, diferentes funções de treinamento e quantidade de épocas de treinamento. Em todos os testes com o AG foram consideradas 100 gerações. Os melhores parâmetros encontrados para as 4 RNAs com melhores resultados, são os mostrados na Tabela 3.5.

Tabela 3.5. Melhores parâmetros do AG para as RNAs MLP com 13 entradas

Técnicas de Classificação	Função da Camada Oculta	Função da Camada de Saída	C1	C2	C3	C4	C5	C6
RNA 1	gaussiana simétrica	sigmoide passo a passo	12	rprop	0,3	0,6	0,3	300
RNA 2	sigmoide simétrica	sigmoide passo a passo	8	rprop	0,2	0,6	0,7	300
RNA 3	*elliot* simétrica	gaussiana	27	rprop	0,6	0,3	0,1	300
RNA 4	*elliot*	linear por partes	12	rprop	0,3	0,4	0,1	300
C1: Qtd. de Neurônios Ocultos		C3: Taxa de Aprendizado	C5: *Decay*					
C2: Algoritmo de Treinamento		C4: *Momentum*	C6: Qtd. de Épocas de Treinamento					

O AG executou vários testes nas RNAs MLP por cerca de 56 minutos, utilizando 100 gerações. Para realizar tais testes, foi usada a linguagem *Python* com a biblioteca FANN (*Fast Artificial Neural Network Library*) [FANN, 2011]. Esta biblioteca livre implementa Redes Neurais Artificiais em C e fornece um *framework* para criação, treinamento e aplicação.

A biblioteca FANN possui suporte às seguintes funções de ativação e que foram utilizadas como genes do AG, e então definidas para as camadas das RNAs:

- funções de ativação linear, linear por partes e linear por partes simétricas;
- funções de ativação limiar e limiar simétrica;
- funções de ativação sigmoide, sigmoide passo a passo, sigmoide com gradiente e sigmoide simétrica com gradiente;
- funções de ativação gaussiana e gaussiana simétrica;
- funções de ativação *elliot* e *elliot* simétrica;
- funções de ativação seno e seno simétrica;
- funções de ativação cosseno e cosseno simétrico.

Além da definição das funções de ativação e valores dos demais parâmetros, foi definido o algoritmo de treinamento, que é responsável pela atualização dos pesos das RNAs à procura de uma resposta melhor. Entre os algoritmos suportados pela biblioteca FANN e que foram utilizados pelo AG estão:

- *Back-propagation*;

- Retropropagação Resiliente (*Rprop*);
- *Quick Propagation* (*QuickProp*).

Os demais parâmetros, como taxa de *decay*, *momentum* e taxa de aprendizado, foram escolhidos dentre os valores: 0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8 e 0,9. Para o AG, foram considerados uma população de 20 indivíduos, satisfação de 1% de erro e o máximo de 100 gerações, similarmente como em Silva [2011].

Além dos testes que consideraram os 13 atributos, foram realizados novos testes com 7 atributos, utilizando tanto o *Weka*, como o SIH. Neste caso, o objetivo é detectar a ameaça antes que chegue ao destino, sendo um enfoque importante para a detecção em tempo real. As características desnecessárias referentes ao destino (duration, destinationTCPFlagsDescription, totalDestinationBytes, destination, destinationPort, totalDestinationPackets) foram, portanto, eliminadas, similarmente como em Mzila & Dube [2013]. Dessa maneira, os testes com 7 atributos são referentes a uma simulação para detecção em tempo real. No SIH os melhores parâmetros encontrados, para as 4 RNAs com melhores resultados, são os mostrados na Tabela 3.6. Os testes foram executados e seus resultados são apresentados no próximo capítulo.

Tabela 3.6. Melhores parâmetros do AG para as RNAs MLP com 7 entradas

Técnicas de Classificação	Função da Camada Oculta	Função da Camada de Saída	C1	C2	C3	C4	C5	C6
RNA 1	gaussiana simétrica	sigmoide passo a passo	15	rprop	0,1	0,2	0,3	300
RNA 2	gaussiana simétrica	sigmoide passo a passo	15	rprop	0,6	0,7	0,3	300
RNA 3	gaussiana simétrica	sigmoide passo a passo	15	rprop	0,4	0,2	0,3	300
RNA 4	gaussiana simétrica	sigmoide passo a passo	15	rprop	0,4	0,2	0,3	300
C1: Qtd. de Neurônios Ocultos		C3: Taxa de Aprendizado	C5: *Decay*					
C2: Algoritmo de Treinamento		C4: *Momentum*	C6: Qtd. de Épocas de Treinamento					

Capítulo 4

Resultados

Neste capítulo são apresentados os resultados dos testes de classificação das diversas técnicas usadas, utilizando tanto o software *Weka* como o Sistema Inteligente Híbrido (SIH). De modo geral, os resultados foram promissores, devido as altas taxas de detecção. Embora, algumas técnicas tenham demonstrado melhores resultados que outras, os resultados encontrados com essa nova base de dados nos mostraram que essas técnicas são fundamentais, podendo evoluir em trabalhos posteriores.

Os resultados apresentados neste capítulo são referentes à utilização da amostra de dados com 212866 registros (cerca de cerca de 10,28%), entretanto, foram realizados os testes considerando uma amostra de dados maior, de 30% e com o conjunto de dados completo. Neste caso verificou-se que os resultados de classificação não apresentaram diferenças significativas quando comparados ao resultados utilizando a amostra de 212866 registros, além de aumentarem significativamente o tempo gasto com o treinamento. Além disso, as técnicas de RNAs apresentaram problemas de generalização dos resultados quando utilizou-se um conjunto de dados maior. Assim, a utilização de uma amostra maior e do conjunto de dados completo foram desconsideradas neste capítulo.

No software *Weka*, os testes consideraram tanto a abordagem *Holdout*, como de Validação Cruzada, sendo que nesta última, foi usada apenas na detecção em modo *offline*, a qual utilizou-se 13 atributos de entrada, devido à necessidade de um custo computacional maior. Enquanto isso, a abordagem *Holdout*, usada no *Weka* e no SIH, foi considerada tanto na detecção em modo *offline*, como na simulação em tempo real (*online)*, a qual utilizou 7 atributos de entrada.

Para estas duas abordagens, foram realizadas as classificações utilizando várias técnicas, como em Tavallaee et al. [2009] e Pachghare & Kulkarni [2011]. Entre elas, estão as Árvores de Decisão (J48, J48*graft*, NB*Tree*, *Random Forest*, *Random Tree* e

Tabela 4.1. Modelo de Resultados

Tráfego	Detectado	Não Detectado
Normal	Verdadeiros Positivos $\frac{NormaisDetectados \times 100}{TotaldeNormais}$	Falso Positivos $\frac{NormaisNãoDetectados \times 100}{TotaldeNormais}$
Ataque	Verdadeiros Positivos $\frac{AtaquesDetectados \times 100}{TotaldeAtaques}$	Falso Negativos $\frac{AtaquesNãoDetectados \times 100}{TotaldeAtaques}$

REP *Tree*), as Redes Neurais Artificiais (MLP e RBF), *Naive Bayes* e Máquina de Vetores de Suporte (SVM) [Chang & Lin, 2011].

Nos testes, utilizaram-se em todas as técnicas os parâmetros padrões do *Weka*, similarmente como em Tavallaee et al. [2009] e Pachghare & Kulkarni [2011]. Foram executados em uma máquina virtual com o sistema operacional *Ubuntu* 12.04 (64 bits) com 16 GB de memória RAM e 2 processadores lógicos. Foram dedicados 12 GB de memória RAM para a inicialização do software *Weka* versão 3.6.6. O sistema *hypervisor* utilizado foi o *Microsoft Hyper-V Server 2008 R2* em um servidor físico *Dell Power Edge* R810 com processadores *Intel Xeon* E7540 de 2.00 Ghz.

Os resultados alcançados referentes às taxas de Verdadeiros Positivos (VP), Falso Positivos (FP) e Falso Negativos (FN) seguirão o modelo da Tabela 4.1 e são apresentados nas seções 4.1 e 4.2.

4.1 Resultados das Técnicas de Classificação

Nesta seção, os resultados das técnicas de classificação utilizadas através do *Weka* são apresentados. Primeiramente, a abordagem de Validação Cruzada foi considerada, devido ao fato de gerar resultados mais confiáveis, embora tenha um custo computacional maior. À vista disso, essa abordagem foi usada apenas para a detecção em modo *offline*.

Por outro lado, a abordagem *Holdout* também foi considerada e então comparada em relação à Validação Cruzada. Logo após, novos resultados foram coletados, utilizando apenas a abordagem *Holdout* numa detecção em tempo real, pois neste enfoque, o objetivo foi obter um melhor desempenho na detecção, com menor custo computacional.

Desse modo, os resultados gerais da abordagem de Validação Cruzada, para uma detecção em modo *offline*, informaram as taxas de classificação correta *(accuracy)* e incorreta *(error rate)*, que são apresentadas na Tabela 4.2.

Tabela 4.2. Resultados gerais das técnicas de classificação (*offline*)

Técnicas de Classificação	Taxa de Classificação		*Root Relative Squared Error (RRSE)*
	Correta	Incorreta	
J48	99,8018%	0,1982%	23,6680%
J48*graft*	99,7759%	0,2241%	25,3826%
NB*Tree*	99,9361%	0,0639%	13,7947%
Random Forest	**99,9460%**	**0,0540%**	**12,2028%**
Random Tree	99,9253%	0,0747%	15,2064%
REP *Tree*	99,9309%	0,0691%	14,4669%
Naive Bayes	95,1683%	4,8317%	120,0111%
SVM	98,3196%	1,6804%	72,9619%
RNA RBF	97,3152%	2,6848%	73,0845%
RNA MLP	98,6743%	1,3257%	48,8314%
Total de Registros: **212866**			
Total de Registros Classificados: **212866**			

A *Random Forest* possuiu os melhores resultados, com uma taxa de erro RRSE menor que 12,5%, mostrada na Figura 4.1. Essa técnica apresentou as melhores taxas de Verdadeiros Positivos para o tráfego "normal" e Falsos Positivos. No entanto, a NB*Tree* foi a que mais detectou o tráfego do tipo "ataque", com menos Falsos Negativos, conforme a Tabela 4.3.

A métrica de erro Raiz Quadrada do Erro Relativo (RRSE - *Root Relative Squared Error*) foi utilizada para comparar os desvios dos valores preditos com os desvios dos valores alvos (desejados). Quanto mais próximo de zero, melhor o resultado. O seu cálculo é aferido conforme a Equação 4.1, onde P_i e T_i são os valores preditos e atuais do exemplo de teste i, respectivamente, enquanto $\overline{T} = \frac{1}{n}\sum_{1}^{n}$ é a média dos valores atuais dos exemplos de teste.

$$E_i = \sqrt{\frac{\sum_{i=1}^{n}(P_i - T_i)^2}{\sum_{i=1}^{n}(T_i - \overline{T})^2}} \tag{4.1}$$

Verificou-se também que, ao considerar o tempo para criação do modelo de classificação, a *Random Tree* torna-se uma opção apropriada, seguida da REP *Tree*, pois sua taxa de erro *RRSE* foi abaixo de 16% e o tempo gasto foi cerca de 8 vezes menor que da *Random Forest*. As demais técnicas apresentaram tempos maiores para criação do modelo, conforme apresentado na Figura 4.4, excetuando a *Naive Bayes* que, apesar de possuir um tempo menor, apresentou as maiores taxas de erro. A NB*Tree*, apesar de ter detectado a maior parte dos "ataques", foi uma das que mais gastaram tempo para criação do modelo, sendo inferior apenas às técnicas SVM e RNA MLP.

Já em relação às RNAs, para todas as análises, a taxa de erro *RRSE* foi maior que

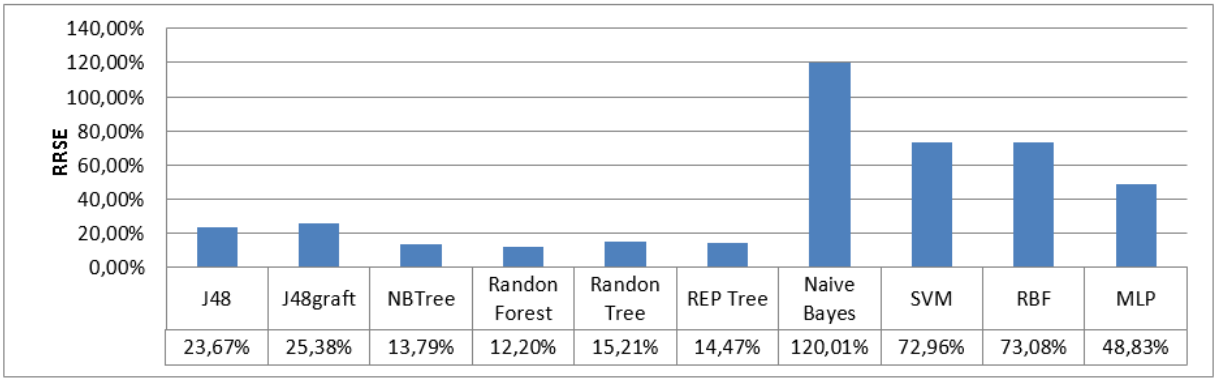

Figura 4.1. *Root Relative Squared Error (RRSE)* - Validação Cruzada

Tabela 4.3. Resultados das técnicas de classificação (*offline*)

Técnicas de Classificação	Normal Verd. Positivos	Ataque Verd. Positivos	Falso Positivos	Falso Negativos
J48	205579 (99,83%)	6865 (98,83%)	341 (0,17%)	81 (1,17%)
J48*graft*	205583 (99,84%)	6806 (97,98%)	337 (0,16%)	140 (2,02%)
NB*Tree*	205852 (99,97%)	**6878 (99,02%)**	68 (0,03%)	**68 (0,98%)**
Random Forest	**205875 (99,98%)**	6876 (98,99%)	**45 (0,02%)**	70 (1,01%)
Random Tree	205838 (99,96%)	6869 (98,89%)	82 (0,04%)	77 (1,11%)
REP *Tree*	205853 (99,97%)	6866 (98,85%)	67 (0,03%)	80 (1,15%)
Naive Bayes	195767 (95,07%)	6814 (98,10%)	10153 (4,93%)	132 (1,90%)
SVM	204527 (99,32%)	4762 (68,56%)	1393 (0,68%)	2184 (31,44%)
RNA RBF	204513 (99,32%)	2638 (37,98%)	1407 (0,68%)	4308 (62,02%)
RNA MLP	204845 (99,48%)	5199 (74,85%)	1075 (0,52%)	1747 (25,15%)

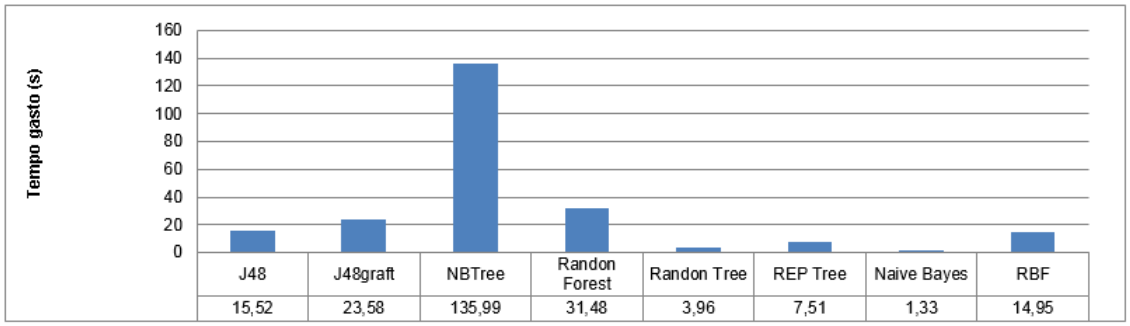

Figura 4.2. Tempo gasto com criação do modelo (s) - Validação Cruzada

das árvores de decisão. Verificou-se ainda que a utilização de todos os registros com as técnicas de RNA apresenta problemas de desempenho e generalização dos resultados, sendo o motivo pelo qual decidiu-se usar uma amostra do conjunto de dados. A RNA MLP torna-se mais apropriada quando são empregados menos registros, que conforme descrito em Braga et al. [2007] geralmente consegue generalizar a informação, mesmo a partir de poucos dados de treinamento. O tempo gasto com a criação do modelo pela RNA MLP foi muito alto (9916,93 s), motivo pelo qual não foi apresentado seu valor

na Figura 4.4. A RNA RBF não conseguiu generalizar adequadamente a informação aprendida, possuindo maiores taxas de erro que a RNA MLP.

Por fim, ao analisarmos as duas abordagens utilizadas, *Holdout* e Validação Cruzada, verificou-se que as técnicas J48*graft*, RNA MLP, *Naive Bayes*, SVM e RNA RBF, por exemplo, possuíram a menor diferença das taxas *RRSE*, conforme mostrado na Figura 4.3, e assim, a utilização da *Holdout* pode ser a mais apropriada, pois seu processo de classificação é mais rápido em comparação ao processo da Validação Cruzada.

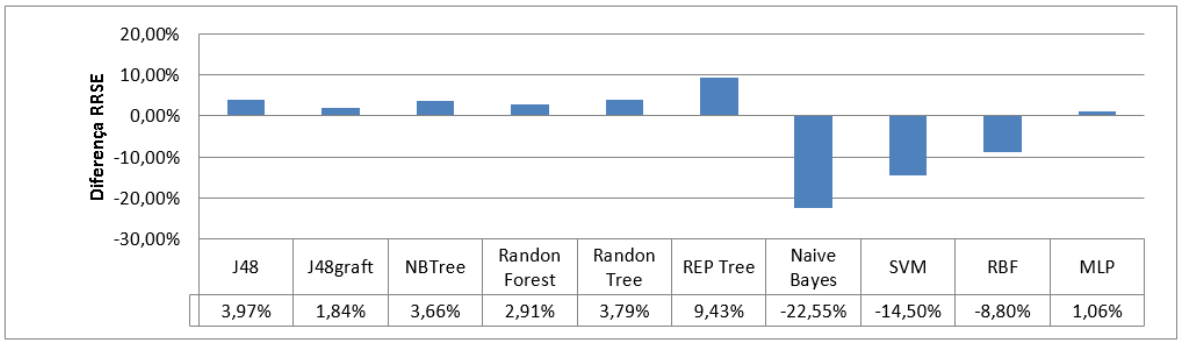

Figura 4.3. Diferença RRSE (*Holdout* - Validação Cruzada)

No entanto, a REP *Tree* apresentou a maior diferença da taxa *RRSE*, sendo acima de 9%. Neste caso, a utilização da Validação Cruzada pode ser mais adequada por apresentar os melhores resultados, em comparação à abordagem *Holdout*, mesmo possuindo um maior custo computacional para sua execução.

Ao comparar os resultados do presente trabalho com Pachghare & Kulkarni [2011], verificou-se que a REP *Tree* apresentou também um dos melhores tempos de criação do modelo (7,61 s) em relação as demais técnicas de árvores de decisão, ficando abaixo apenas da *Random Tree* que gastou 3,96 s. Entretanto, os resultados de classificações corretas foram ligeiramente piores que de algumas árvores de decisão. Neste trabalho, esta técnica além de ter apresentado o 2º melhor desempenho em relação ao tempo de criação do modelo das árvores de decisão, possuiu uma taxa de classificação melhor que as árvores de decisão J48 e J48*graft* e *Random Tree*.

Ao considerar o trabalho de Tavallaee et al. [2009], as árvores de decisão J48 e NB*Tree* foram as que apresentaram os melhores resultados de classificações corretas, enquanto a RNA MLP apresentou resultados piores do que todas as árvores de decisão. Esse resultado, referente às RNAs MLP e a NB*Tree* também se assemelha ao encontrado no presente trabalho.

Outros resultados importantes são referentes aos testes com apenas 7 atributos (modo de detecção em tempo real). Neste caso, verificou-se que ao eliminar as características relacionadas ao destino, o tempo gasto para criação do modelo foi maior em

todas as técnicas. Todavia, apenas uma pequena diferença na taxa de classificação correta foi constatada, similarmente como em Mzila & Dube [2013], justificando, portanto, a utilização de tal enfoque para uma detecção em tempo real, como pode ser observado nas tabelas 4.4 e 4.5.

Neste caso, apesar do tempo de criação do modelo ter sido maior em todas as técnicas, ainda assim, este enfoque pode ter um melhor desempenho por necessitar coletar menos informações dos PDUs transmitidos.

Tabela 4.4. Resultados gerais das técnicas de classificação (tempo real)

Técnicas de Classificação	Taxa de Classificação		*Root Relative Squared Error (RRSE)*
	Correta	Incorreta	
J48	99,7245%	0,2725%	27,4379%
J48*graft*	99,6717%	0,3283%	30,7240%
NB *Tree*	**99,8732%**	**0,1268%**	**17,4518%**
Random Forest	99,8673%	0,1327%	18,9205%
Random Tree	99,8185%	0,1815%	23,7178%
REP *Tree*	99,8456%	0,1544%	21,2610%
Naive Bayes	96,6986%	3,3014%	83,3296%
SVM	97,7668%	2,2332%	84,0441%
RNA RBF	98,3235%	1,6765%	66,4522%
RNA MLP	98,3928%	1,6072%	57,2780%
Total de Registros: **212866**			
Total de Registros Classificados: **170293**			

Tabela 4.5. Resultados das técnicas de classificação (tempo real)

Técnicas de Classificação	Normal Verd. Positivos	Ataque Verd. Positivos	Falso Positivos	Falso Negativos
J48	164366 (98,78%)	5463 (98,15%)	361 (0,22%)	103 (1,85%)
J48*graft*	164396 (99,80%)	5338 (95,90%)	331 (0,20%)	228 (4,10%)
NB*Tree*	164609 (99,93%)	**5468 (98,24%)**	118 (0,07%)	**98 (1,76%)**
Random Forest	**164627 (99,94%)**	5440 (97,74%)	**100 (0,06%)**	126 (2,26%)
Random Tree	164600 (99,92%)	5384 (96,73%)	127 (0,08%)	182 (3,27%)
REP *Tree*	164599 (99,92%)	5431 (97,57%)	128 (0,08%)	135 (2,43%)
Naive Bayes	159203 (96,65%)	5468 (98,24%)	5524 (3,35%)	98 (1,76%)
SVM	164082 (99,61%)	2408 (43,26%)	645 (0,39%)	3158 (56,74%)
RNA RBF	163441 (99,22%)	3997 (71,81%)	1286 (0,78%)	1569 (28,19%)
RNA MLP	163946 (99,53%)	3610 (64,86%)	781 (0,47%)	1956 (35,14%)

A árvore de decisão NB*Tree* teve a melhor precisão e, consequentemente o menor erro, contudo o seu tempo gasto para criação do modelo foi 33,96 s, sendo cerca de 10 vezes maior que de outras técnicas, como *Random Tree* e REP *Tree*, e quase 2 vezes maior que da *Random Forest*, conforme mostrado na Figura 4.3 Outro ponto que merece destaque é que, a *Random Forest* conseguiu detectar melhor o tráfego normal, enquanto a NB*Tree* detectou melhor o tráfego referente aos ataques.

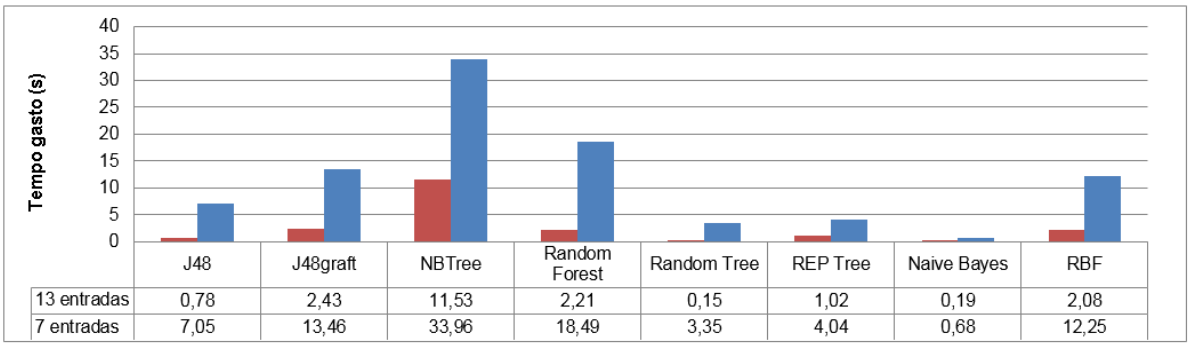

Figura 4.4. Tempo gasto com criação do modelo (s) - *Holdout*

4.2 Resultados do Sistema Inteligente Híbrido

Nesta seção, os resultados do Sistema Inteligente Híbrido (SIH) são apresentados. Neste caso, verificou-se que, ao combinar a técnica de AG com as RNAs MLP, melhores resultados foram alcançados.

A escolha dos parâmetros iniciais da rede neural pelo AG permitiu aperfeiçoar a estrutura inicial da RNA MLP, minimizando este problema inicial e melhorando seus resultados finais. Nestes resultados, a abordagem *Holdout* foi considerada, separando em dois grupos (treinamento e testes), de forma similar, quando utilizados no *Weka*.

Assim, os resultados para as 4 melhores RNAs encontradas pelo SIH, utilizando a abordagem *Holdout*, são apresentados nas tabelas 4.6, 4.7 e 4.8.

Tabela 4.6. Resultados gerais do sistema inteligente híbrido

Técnicas de Classificação	Taxa de Classificação (*offline*)		Taxa de Classificação (tempo real)	
	Correta	Incorreta	Correta	Incorreta
RNA 1	**99,7774%**	**0,2226%**	99,5578%	0,4422%
RNA 2	99,7639%	0,2361%	**99,5578%**	**0,4422%**
RNA 3	99,7628%	0,2372%	99,5549%	0,4451%
RNA 4	99,7533%	0,2467%	99,5543%	0,4457%
Total de Registros: **212866**				
Total de Registros Classificados: **170293**				

Tabela 4.7. Resultados do sistema inteligente híbrido (*offline*)

Técnicas de Classificação	Normal Verd. Positivos	Ataque Verd. Positivos	Falso Positivos	Falso Negativos
RNA 1	**164506 (99,85%)**	5408 (97,67%)	**250 (0,15%)**	129 (2,33%)
RNA 2	164495 (99,84%)	5396 (97,45%)	261 (0,16%)	141 (2,55%)
RNA 3	164471 (99,83%)	**5418 (97,85%)**	285 (0,17%)	**119 (2,15%)**
RNA 4	164462 (96,82%)	5411 (97,72%)	294 (0,18%)	126 (2,28%)

Conforme mostrado na Tabela 4.6, pode-se verificar que em ambos os modos de detecção, *offline* (13 entradas) e tempo real (7 entradas), o AG permitiu um acerto

maior das taxas de detecção, permitindo que as RNAs utilizassem novos parâmetros capazes de aproximar suas classificações de um melhor global. Seus valores, considerando o modo *offline*, aproximaram-se dos encontrados pela J48, superando os valores encontrados através das RNAs RBF e MLP pelas abordagens *Holdout* e de Validação Cruzada, quando utilizadas de forma isolada.

Ao compararmos com o trabalho de Silva [2011], é constatado também uma melhora nas taxas de acerto e uma diminuição dos alarmes falsos. Verificamos que, apesar de utilizar uma base de dados diferente, as técnicas combinadas contribuem para uma melhor precisão.

Tabela 4.8. Resultados do sistema inteligente híbrido (tempo real)

Técnicas de Classificação	Normal Verd. Positivos	Ataque Verd. Positivos	Falso Positivos	Falso Negativos
RNA 1	164276 (99,71%)	5264 (95,07%)	480 (0,29%)	273 (4,93%)
RNA 2	**164284 (99,71%)**	5256 (94,93%)	**472 (0,29%)**	281 (5,07%)
RNA 3	164264 (99,70%)	**5271 (95,20%)**	492 (0,30%)	**266 (4,80%)**
RNA 4	164268 (99,70%)	5266 (95,11%)	488 (0,30%)	271 (4,89%)

Para o modo de detecção em tempo real, o SIH obteve melhores resultados em relação às técnicas Naive Bayes, SVM, RNA RBF e RNA MLP, entretanto, as árvores de decisão ainda apresentaram os melhores resultados, conforme mostrado nas tabelas 4.4 e 4.5.

Isso mostra que a generalização das RNAs garante altas taxas de classificação na base de dados, porém o reconhecimento baseado na análise de semelhanças de dados, fornecido pelas árvores de decisão, ainda conseguiu apresentar taxas maiores de acerto, demonstrando que a análise generalizada dos dados possui limitações e que algoritmos que trabalham no agrupamento de similaridades possuem melhores resultados para esse tipo de dados.

Capítulo 5

Conclusões

Os resultados obtidos permitem concluir que as melhores técnicas podem ser combinadas em um sistema de detecção de intrusão real e, assim, dependendo do ambiente, pode-se optar por uma ou por outra, ou por um conjunto delas. Como exemplos, vários produtos atualmente do mercado utilizam a abordagem de detecção híbrida, apesar de não informarem certamente quais técnicas utilizam, alguns como o Aker Firewall, permite estender o produto com um módulo de IDS externo, que pode ser desenvolvido ou utilizado com outro produto, como por exemplo o software *opensource* Snort.

Ao classificar os registros da base ISCX 2012 é possivel concluir também que as técnicas de classificação utilizadas são promissoras mesmo nas bases mais recentes, em que o perfil de tráfego tende a ser diferente. E com a tendência de variabilidade no perfil de tráfego em redes reais, é importante investigar técnicas que tenham capacidade de adaptação às bases de dados de perfis de tráfego diferentes.

Na utilização das abordagens *Holdout* e Validação Cruzada, as técnicas *Random Forest* e NB*Tree*, por exemplo, apresentaram os melhores resultados quanto à precisão, mas em relação ao tempo gasto para criação do modelo, foram piores do que as demais. Neste caso, o ambiente que necessita de um melhor tempo de resposta, como uma detecção em tempo real e que possua um grande volume de dados, a aplicação da técnica que gaste menos tempo torna-se mais apropriada.

Entretando, em um ambiente, onde a detecção seja em modo *offline* ou que tenha um volume menor de dados, ou que o sistema computacional seja mais robusto, a NB*Tree* e *Random Forest* são as mais adequadas.

Ao considerarmos uma detecção em tempo real, nosso trabalho eliminou alguns atributos referentes ao destino. Neste cenário, a técnica mais apropriada foi a NB*Tree*, embora, a técnica *Random Forest* tenha detectado melhor o tráfego normal.

A combinação do AG e das RNAs permitiu melhores resultados, formando um

Sistema Inteligente Híbrido (SIH). Neste caso, o AG encontrou melhores parâmetros para as RNAs, minimizando seus problemas em relação à escolha inicial dos parâmetros. Dessa forma, a aplicação de determinadas técnicas selecionadas permite combinar as suas vantagens e minimizar suas desvantagens, sendo uma tendência para realizar novas combinações, aumentando cada vez mais a precisão dos resultados.

Nos diversos trabalhos relacionados, mesmo utilizando bases de dados diferentes, as técnicas utilizadas são importantes para prover a segurança das redes e computadores e, o fato deste trabalho ter utilizado uma base de dados de perfis de tráfego mais recente e as técnicas terem apresentado resultados promissores, motiva a utilizar e estender estas técnicas como, por exemplo, criando outros sistemas inteligentes híbridos, que permitirão evoluir cada vez mais os sistemas de dectecção e prevenção de intrusões.

Como trabalhos futuros sugere-se a criação de um classificador que identifique cada tipo de ataque, aplicando o algoritmo que melhor apresentou respostas, além de realizar a classificação das intrusões utilizando mais técnicas de IA de forma combinada com o intuito de melhorar as taxas de classificação e tempo gasto. Pode-se também realizar a detecção de intrusões a partir de uma inspeção profunda dos dados através dos *payloads* da base ISCX 2012, e não apenas a partir de informações dos níveis de rede e de transporte. Além disso, as próprias bibliotecas em Java do software Weka, ou as bibliotecas em Python, podem ser utilizadas para desenvolver ou estender em um sistema de IDS/IPS real.

Referências Bibliográficas

Al-Janabi, S. & Saeed, H. (2011). A neural network based anomaly intrusion detection system. Em *Developments in E-systems Engineering (DeSE)*, pp. 221–226.

Artero, A. O. (2009). *Inteligência Artificial: Teórica e Prática*. Livraria da Física, 1 edição. ISBN 9788578610296.

Braga, A. d. P.; Carvalho, A. P. d. L. F. d. & Ludermir, T. B. (2007). *Redes neurais artificiais: teoria e aplicações*. LTC Editora, 2 edição. ISBN 9788521615644.

Britt, W.; Gopalaswamy, S.; Hamilton, J. A.; Dozier, G. V. & Chang, K. H. (2007). Computer defense using artificial intelligence. Em *Proceedings of the 2007 spring simulation multiconference*, volume 3 of *SpringSim '07*, pp. 378–386, San Diego, CA, USA. Society for Computer Simulation International.

Bukhtoyarov, V. & Semenkin, E. (2012). Neural networks ensemble approach for detecting attacks in computer networks. Em *Evolutionary Computation (CEC), 2012 IEEE Congress*, pp. 1–6.

CERT.br (2014). Estatísticas dos incidentes reportados. Disponível em: http://www.cert.br/stats/incidentes/.

Chang, C.-C. & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27. Disponível em: http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Devaraju, S. & Ramakrishnan, S. (2011). Performance analysis of intrusion detection system using various neural network classifiers. Em *Recent Trends in Information Technology (ICRTIT), 2011 International Conference*, pp. 1033–1038.

Douligeris, C. & Serpanos, D. (2007). *Network security: current status and future directions*. Wiley Editora. ISBN 9780471703556.

FANN (2011). Fann: Fast artificial neural network library. Acesso em: Julho de 2013. Disponível em: http://leenissen.dk/fann/wp/.

Han, J.; Kamber, M. & Pei, J. (2011). *Data Mining: Concepts and Techniques: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science Editora. ISBN 9780123814807.

KDD-Cup (1999). Kdd-cup 1999 data set. Acesso em: Julho de 2013. Disponível em: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Kurose, J. & Ross, K. (2010). *Redes de computadores e a internet: uma abordagem top-down*. Pearson Editora. ISBN 9788588639973.

Larson, R. & Farber, B. (2004). *Estatística aplicada*. Prentice Hall Editora, 2 edição. ISBN 9788587918598.

Li, F. (2010). Hybrid neural network intrusion detection system using genetic algorithm. Em *Multimedia Technology (ICMT), 2010 International Conference*, pp. 1–4.

Marhusin, M.; Cornforth, D. & Larkin, H. (2008). An overview of recent advances in intrusion detection. Em *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference*, pp. 432–437.

McHugh, J. (2000). Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262--294. ISSN 1094-9224.

Mzila, P. & Dube, E. (2013). The effect of destination linked feature selection in real-time network intrusion detection. Em *ICIMP 2013 : 8th International Conference on Internet Monitoring and Protection*.

NSL-KDD (2009). The nsl-kdd data set. Acesso em: Julho de 2013. Disponível em: http://nsl.cs.unb.ca/NSL-KDD/.

Pachghare, V. K. & Kulkarni, P. (2011). Pattern based network security using decision trees and support vector machine. Em *Electronics Computer Technology (ICECT), 2011 3rd International Conference*, volume 5, pp. 254–257.

Patcha, A. & Park, J. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks, 2007*, 51(12):3448–3470.

Rezende, S. (2005). *Sistemas Inteligentes: Fundamentos e Aplicações*. Manole Editora. ISBN 9788520416839.

Sallay, H.; Ammar, A.; Ben Saad, M. & Bourouis, S. (2013). A real time adaptive intrusion detection alert classifier for high speed networks. Em *Network Computing and Applications (NCA), 2013 12th IEEE International Symposium on*, pp. 73–80.

Shiravi, A.; Shiravi, H.; Tavallaee, M. & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security*, 31(3):357–374. ISSN 0167-4048.

Silva, J. R. C. d. (2011). Sistemas de detecção de intrusão com técnicas de inteligëncia artificial. 2011. 157 f. Dissertação de mestrado, Programa de Pós-Graduação em Ciëncia da Computação, Universidade Federal de Viçosa, Viçosa - MG.

Stallings, W. (2010). *Network Security Essentials: Applications and Standards*. Prentice Hall Editora. ISBN 9780136108054.

Tan, P.; Steinbach, M. & Kumar, V. (2009). *Introdução ao datamining: mineração de dados*. Ciëncia Moderna Editora. ISBN 9788573937619.

Tanenbaum, A. S. (2011). *Redes de computadores*. Pearson Editora. ISBN 9788576059240.

Tavallaee, M.; Bagheri, E.; Lu, W. & Ghorbani, A. (2009). A detailed analysis of the kdd cup 99 data set. Em *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium*, pp. 1–6.

Weka (2013). Weka 3: Data mining software in java. Acesso em: Julho de 2013. Disponível em: http://www.cs.waikato.ac.nz/ml/weka/.

Zhang, R.; Qian, D.; Ba, C.; Wu, W. & Guo, X. (2001). Multi-agent based intrusion detection architecture. Em *Computer Networks and Mobile Computing, 2001. Proceedings. 2001 International Conference*, pp. 494–501.

Anexo A

Código para padronização dos arquivos .csv

Este código foi desenvolvido na linguagem *Python* pelo autor deste trabalho, sendo responsável pela padronização dos arquivos .csv da base ISCX 2012, os quais possuem informações do tráfego de rede capturado e rotulado.

Estes arquivos possuiam colunas adicionais ou fora de ordem, além de inconsistências geradas por vírgulas da coluna UTF, a qual continha informações dos *payloads* dos pacotes, que geravam colunas adicionais nos registros, as quais foram removidas ou reorganizadas para que todos os arquivos tivessem o mesmo formato.

padronizacaoArquivosCSV.py

```
1 # -*- coding: utf-8 -*-
  '''
3 Created on Abr/2013

5 @author: Rafael Antonio Gonçalves Lima (rafael.a.lima@ufv.br /
      rafaelglima@gmail.com)
  '''
7
  import csv, sys, os
9
  #diretorio da base de dados e quais arquivos
11 dir_arquivos = '/home/rafael/Projeto/iscx2012/labeled_flows/'

13 #lista de arquivos a serem verificados
  arquivos_teste = [dir_arquivos + 'TestbedMonJun14.csv',     # retirou a
      coluna sourcePayloadAsUTF
15                  dir_arquivos + 'TestbedSatJun12.csv',     # nao precisa
```

```
                    de alteracao
                dir_arquivos + 'TestbedSunJun13.csv',    # nao precisa
                    de alteracao
                dir_arquivos + 'TestbedThuJun17-a.csv', # retirou a
                    coluna destinationPayloadUTF
                dir_arquivos + 'TestbedThuJun17-b.csv', # retirou a
                    coluna destinationPayloadUTF
                dir_arquivos + 'TestbedThuJun17-c.csv', # retirou a
                    coluna destinationPayloadUTF
                dir_arquivos + 'TestbedTueJun15-a.csv', # retirou as
                    colunas destinationPayloadUTF e sensorInterfaceID
                dir_arquivos + 'TestbedTueJun15-b.csv', # retirou a
                    coluna destinationPayloadUTF
                dir_arquivos + 'TestbedTueJun15-c.csv', # retirou a
                    coluna destinationPayloadUTF
                dir_arquivos + 'TestbedWedJun16-a.csv', # retirou as
                    colunas destinationPayloadUTF e startTime
                dir_arquivos + 'TestbedWedJun16-b.csv', # retirou a
                    coluna destinationPayloadUTF
                dir_arquivos + 'TestbedWedJun16-c.csv',] # retirou a
                    coluna destinationPayloadUTF

#loop para avaliacao dos arquivos
for index in range(len(arquivos_teste)):

    print 'Arquivo de teste: ', arquivos_teste[index]

    #arquivo .csv a ser criado
    arq_semUTF = open(arquivos_teste[index]+'_semUTF', "wb")
    f_semUTF = csv.writer(arq_semUTF, delimiter='\t')

    #abre o arquivo .csv em modo leitura
    f = open(arquivos_teste[index], 'rb')

    reader = csv.reader(f , delimiter=',')
    try:
        #contador
        count = 0

        for row in reader:
            #print row[0][::-1] #inverte a palavra da primeira coluna
            #print row[::-1] #inverte a linha
            #print count, '\t', row[::-1][0], '\t', row[::-1][1], '\t',
                row[::-1][2], '\t', row[::-1][3], '\t', row[::-1][4], '\t
```

```
        ', row[::-1][5], '\t', row[::-1][6], '\t', row[::-1][7],
        '\t', row[::-1][8], '\t', row[::-1][9], '\t', row
        [::-1][10]

49      #escreve no arquivo pegando as colunas de 0 à 6 pela esquerda
            , e as colunas 18 à 8 pela direita
        #eliminando assim a coluna com UTF na qual estava gerando
            inconsistências por também haver virgulas
51      if index == 0:
            f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                [4], row[5], row[6], row[::-1][10], row[::-1][9], row
                [::-1][8], row[::-1][7], row[::-1][6], row[::-1][5],
                row[::-1][4], row[::-1][3], row[::-1][2], row
                [::-1][1], row[::-1][0]])
53      elif index == 1:
            f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                [4], row[5], row[6], row[::-1][12], row[::-1][11], row
                [::-1][10], row[::-1][9], row[::-1][8], row[::-1][7],
                row[::-1][6], row[::-1][5], row[::-1][4], row
                [::-1][3], row[::-1][2], row[::-1][1], row[::-1][0]])
55      elif index == 2:
            f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                [4], row[5], row[6], row[::-1][12], row[::-1][11], row
                [::-1][10], row[::-1][9], row[::-1][8], row[::-1][7],
                row[::-1][6], row[::-1][5], row[::-1][4], row
                [::-1][3], row[::-1][2], row[::-1][1], row[::-1][0]])
57      elif index == 3:
            f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                [4], row[5], row[6], row[7], row[::-1][10], row
                [::-1][9], row[::-1][8], row[::-1][7], row[::-1][6],
                row[::-1][5], row[::-1][4], row[::-1][3], row
                [::-1][2], row[::-1][1], row[::-1][0]])
59      elif index == 4:
            f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                [4], row[5], row[6], row[7], row[::-1][10], row
                [::-1][9], row[::-1][8], row[::-1][7], row[::-1][6],
                row[::-1][5], row[::-1][4], row[::-1][3], row
                [::-1][2], row[::-1][1], row[::-1][0]])
61      elif index == 5:
            f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                [4], row[5], row[6], row[7], row[::-1][10], row
                [::-1][9], row[::-1][8], row[::-1][7], row[::-1][6],
                row[::-1][5], row[::-1][4], row[::-1][3], row
                [::-1][2], row[::-1][1], row[::-1][0]])
```

```
63              elif index == 6:
                    f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                        [4], row[5], row[6], row[7], row[8], row[::-1][10],
                        row[::-1][9], row[::-1][8], row[::-1][7], row
                        [::-1][6], row[::-1][5], row[::-1][4], row[::-1][3],
                        row[::-1][2], row[::-1][1], row[::-1][0]])
65              elif index == 7:
                    f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                        [4], row[5], row[6], row[7], row[::-1][10], row
                        [::-1][9], row[::-1][8], row[::-1][7], row[::-1][6],
                        row[::-1][5], row[::-1][4], row[::-1][3], row
                        [::-1][2], row[::-1][1], row[::-1][0]])
67              elif index == 8:
                    f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                        [4], row[5], row[6], row[7], row[::-1][10], row
                        [::-1][9], row[::-1][8], row[::-1][7], row[::-1][6],
                        row[::-1][5], row[::-1][4], row[::-1][3], row
                        [::-1][2], row[::-1][1], row[::-1][0]])
69              elif index == 9:
                    f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                        [4], row[5], row[6], row[7], row[::-1][11], row
                        [::-1][10], row[::-1][9], row[::-1][8], row[::-1][7],
                        row[::-1][6], row[::-1][5], row[::-1][4], row
                        [::-1][3], row[::-1][2], row[::-1][1], row[::-1][0]])
71              elif index == 10:
                    f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                        [4], row[5], row[6], row[7], row[::-1][10], row
                        [::-1][9], row[::-1][8], row[::-1][7], row[::-1][6],
                        row[::-1][5], row[::-1][4], row[::-1][3], row
                        [::-1][2], row[::-1][1], row[::-1][0]])
73              elif index == 11:
                    f_semUTF.writerow([row[0], row[1], row[2], row[3], row
                        [4], row[5], row[6], row[7], row[::-1][10], row
                        [::-1][9], row[::-1][8], row[::-1][7], row[::-1][6],
                        row[::-1][5], row[::-1][4], row[::-1][3], row
                        [::-1][2], row[::-1][1], row[::-1][0]])
75
                #incrementa o contador, apenas para mostrar a linha atual
77              count += 1

79      except csv.Error, e:
            sys.exit('file %s, line %d: %s:' % (arquivos_teste[index], reader
                .line_num, e))
81
```

```
  #fecha o arquivo
83 f.close()
  #finaliza o programa
85 sys.exit
```

Anexo B

Código para gerar arquivo .arff

Este código foi desenvolvido na linguagem *Python* pelo autor deste trabalho, sendo responsável por transformar os arquivos .csv no formato .arff (utilizado pelo software *Weka*), adicionando o cabeçalho com informações dos atributos, além de eliminar algumas inconsistências de registros.

geraArquivoArff.py

```
1 # -*- coding: utf-8 -*-
  '''
3 Created on Abr/2013

5 @author: Rafael Antonio Gonçalves Lima (rafael.a.lima@ufv.br /
      rafaelglima@gmail.com)
  '''
7
  import csv, sys, os
9 import re

11 def cabecalhoWeka():

13     cabecalho = """@relation \'iscx2012\'
  @attribute \'duration\' real
15 @attribute \'protocolName\' {\'tcp\',\'udp\',\'icmp\'}
  @attribute \'sourceTCPFlagsDescription\' {\'N/A\',\'SPA\', \'SRPA\',\'
      FSPA\', \'FRPA\', \'FSRPA\', \'FA\', \'FRA\', \'FPA\', \'FSA\', \'SRA
      \', \'RPA\', \'R\', \'S\', \'A\',\'RA\',\'PA\', \'SA\', \'SR\', \'FSRA
      \', \'FPU\', \'FSPU\', \'FSRPU\'}
17 @attribute \'destinationTCPFlagsDescription\' {\'N/A\',\'SPA\', \'SRPA
      \',\'FSPA\', \'FRPA\', \'FSRPA\', \'FA\', \'FRA\', \'FPA\', \'FSA\',
      \'SRA\', \'RPA\', \'R\',  \'S\', \'A\',\'RA\', \'PA\',\'SA\', \'FSRA
```

```
    \'}
@attribute \'totalSourceBytes\' real
@attribute \'totalDestinationBytes\' real
@attribute \'source\' {\'0\',\'1\', \'2\', \'3\', \'4\', \'5\', \'6\'}
@attribute \'destination\' {\'0\',\'1\', \'2\', \'3\', \'4\', \'5\',
    \'6\'}
@attribute \'sourcePort\' numeric
@attribute \'destinationPort\' numeric
@attribute \'totalSourcePackets\' real
@attribute \'totalDestinationPackets\' real
@attribute \'direction\' {\'L2R\',\'L2L\',\'R2R\',\'R2L\'}
@attribute \'tag\' {\'Normal\',\'Attack\'}
@data
"""
    return cabecalho

def calculaDuracaoSegundos(dateTime):
    data = 0
    horario = 0
    duracao = 0
    hora = 0
    minutos = 0
    segundos = 0
    try:
        data, horario = dateTime.split(' ')
        hora, minutos, segundos = horario.split(':')
        duracao = int(segundos) + int(minutos)*60 + int(hora)*360;
    except TypeError, erro:
        print "um erro ocorreu: %s" % erro
        print dateTime
        duracao = 0;

    if duracao < 0:
        return 0
    else:
        return duracao

def enderecoRede(endereco):

    result = '?'
    if endereco[0:8] != '192.168.':
        result = '0'            #endereco internet
    elif endereco[0:10] == '192.168.1.':
        result = '1'
```

```
        elif endereco[0:10] == '192.168.2.':
61          result = '2'
        elif endereco[0:10] == '192.168.3.':
63          result = '3'
        elif endereco[0:10] == '192.168.4.':
65          result = '4'
        elif endereco[0:10] == '192.168.5.':
67          result = '5'
        elif endereco[0:10] == '192.168.6.':
69          result = '6'
        return result
71
    #diretorio das entradas ISCX2012
73  dir_arquivos = '/home/rafael/iscx2012/labeled_flows/'

75  #lista de arquivos a serem verificados
    arquivos_teste = [dir_arquivos + 'TestbedWedJun16-a.csv',
77                    dir_arquivos + 'TestbedWedJun16-b.csv',
                      dir_arquivos + 'TestbedWedJun16-c.csv']
79
    #loop para avaliacao dos arquivos
81  for index in range(len(arquivos_teste)):

83      #array para entradas
        entradas = [[0 for x in xrange(14)] for x in xrange(len(
            arquivos_teste))]
85
        #print entradas
87      print 'Arquivo de teste: ', arquivos_teste[index]

89      #abre o arquivo .csv em modo leitura
        f = open(arquivos_teste[index], 'rb')
91
        #arquivo .csv a ser criado de entradas
93      fEntradas = open(arquivos_teste[index]+'.arff', "wb")
        #adiciona cabecalho Weka
95      fEntradas.write(cabecalhoWeka())
        #escreve como .csv
97      fEntradas = csv.writer(fEntradas, delimiter=',')

99      reader = csv.reader(f , delimiter='\t')
        try:
101
            #percorre o arquivo
```

```
103         for row in reader:

105             #contador
                count = 0
107
                if row[0] != 'appName':
109
                    #duration = StopTime - StartTime
111                 entradas[count][0] = calculaDuracaoSegundos(row[::-1][1])
                        - calculaDuracaoSegundos(row[::-1][2])
                    if entradas[count][0] < 0:
113                     entradas[count][0] = '?' #caso o valor duration seja
                            negativo, será considerado "nulo" pelo Weka

115                 #protocolName
                    if row[::-1][6]=='tcp_ip':
117                     entradas[count][1] = 'tcp'
                    elif row[::-1][6]=='udp_ip':
119                     entradas[count][1] = 'udp'
                    elif row[::-1][6]=='icmp_ip':
121                     entradas[count][1] = 'icmp'

123                 #sourceTCPFlagsDescription
                    if row[9] == '':
125                     entradas[count][2] = '?'
                    else:
127                     entradas[count][2] = re.sub(',','',row[9]) #remove a
                            vírgula entre as flags

129                 if entradas[count][2] == 'SRIllegal7Illegal8': #trata
                        registro errado
                        entradas[count][2] = 'SR'
131
                    #destinationTCPFlagsDescription
133                 if row[10] == '':
                        entradas[count][3] = '?'
135                 else:
                        entradas[count][3] = re.sub(',','',row[10])
137
                    if entradas[count][3] == 'FSPAIllegal8':      #trata
                        registro errado
139                     entradas[count][3] = 'FSPA'
                    elif entradas[count][3] == 'SRAIllegal8':
141                     entradas[count][3] = 'SRA'
```

```
                elif entradas[count][3] == 'RIllegal8':
143                 entradas [count][3] = 'R'
                elif entradas[count][3] == 'SPAIllegal8':
145                 entradas [count][3] = 'SPA'
                elif entradas[count][3] == 'FSRPAIllegal8':
147                 entradas [count][3] = 'FSRPA'
                elif entradas[count][3] == 'RAIllegal7Illegal8':
149                 entradas [count][3] = 'RA'
                elif entradas[count][3] == 'RAIllegal7':
151                 entradas [count][3] = 'RA'
                elif entradas[count][3] == 'RAIllegal8':
153                 entradas [count][3] = 'RA'
                elif entradas[count][3] == 'FPAIllegal8':
155                 entradas [count][3] = 'FPA'

157             #totalSourceBytes
                entradas[count][4] = row[1]
159             #totalDestionationBytes
                entradas[count][5] = row[2]
161             #source
                entradas[count][6] = enderecoRede(row[11])  #0 se for
                    endereco da internet e 2-6 se for das sub-redes
                    internas
163             #destination
                entradas[count][7] = enderecoRede(row[::-1][4]) #0 se for
                     endereco da internet e 2-6 se for das sub-redes
                    internas
165             #sourcePort
                entradas[count][8] = row[::-1][5]
167             #destinationPort
                entradas[count][9] = row[::-1][3]
169             #totalSourcePackets
                entradas[count][10] = row[4]
171             #totalDestinationPackets
                entradas[count][11] = row[3]
173
                #direction
175             if row[8] != ',L2R"':
                    x = row[8]
177                 x = x[-7:]      #pega os 7 ultimos caracteres
                        retirando o payload que veio no campo
                    match = re.search('\w\w\w', x)
179                 if match:
                        entradas[count][12] = match.group()
```

```
181                     else:
                            entradas[count][12] = x
183                 else:
                        entradas[count][12] = 'L2R'
185
                    #tag
187                 entradas[count][13] = row[::-1][0] #'Normal' ou 'Attack'

189                 fEntradas.writerow([entradas[count][0], entradas[count
                        ][1], entradas[count][2], entradas[count][3], entradas
                        [count][4], entradas[count][5], entradas[count][6],
                        entradas[count][7],entradas[count][8],entradas[count
                        ][9],entradas[count][10], entradas[count][11],
                        entradas[count][12], entradas[count][13]])

191     except csv.Error, e:
            sys.exit('file %s, line %d: %s:' % (arquivos_teste[index], reader
                .line_num, e))
193
#fecha o arquivo de leitura
195 f.close()
#finaliza o programa
197 sys.exit
```

Anexo C

Código do sistema inteligente híbrido

Este código foi desenvolvido na linguagem *Python* e criado pelo Msc. Jacson Rodrigues Correia da Silva e então adaptado para este trabalho. Ele é responsável por executar o Algoritmo Genético definindo os parâmetros iniciais para a estrutura da RNA MLP.

RNAGenetico.py

```
1 # -*- coding: utf-8 -*-
  '''
3 Created on Jan/2011

5 @author: Jacson RC Silva <jacsonrcsilva@gmail.com / jacson.silva@ufes.br>
  '''
7
  from algoritmoGenetico.AlgoritmoGenetico import AlgGenetico
9 from pyfann import libfann
  from math import sqrt
11
  logFile = "Execucao_Alg_Gen.log"
13
  qntNeuroniosEntrada  = 7 # adaptado para este trabalho utilizando 13 e 7
      entradas #
15 qntNeuroniosSaida    = 2

17 # Genes de um cromossomo
  # x*0.1 for x in (range(1,9)) = 0.1,..., 0.9
19 learningRate = [ x*0.1 for x in (range(1,9)) ]
  momentum      = [ x*0.1 for x in (range(1,9)) ]
21 lrdecay       = [ x*0.1 for x in (range(1,9)) ]
```

```
trainers      = [ libfann.TRAIN_INCREMENTAL, libfann.TRAIN_QUICKPROP,
                  libfann.TRAIN_RPROP, libfann.TRAIN_BATCH ]
camadas       = [         libfann.SIGMOID, libfann.SIGMOID_SYMMETRIC,
                          libfann.SIGMOID_STEPWISE, libfann.
                              SIGMOID_SYMMETRIC_STEPWISE,
                          libfann.LINEAR, libfann.LINEAR_PIECE,
                          libfann.LINEAR_PIECE_SYMMETRIC, libfann.THRESHOLD
                              ,
                          libfann.THRESHOLD_SYMMETRIC, libfann.GAUSSIAN,
                          libfann.GAUSSIAN_STEPWISE, libfann.
                              GAUSSIAN_SYMMETRIC,
                          libfann.ELLIOT, libfann.ELLIOT_SYMMETRIC,
                          libfann.COS_SYMMETRIC, libfann.SIN_SYMMETRIC ]

neuroniosOcultos  = [ 5, 8, 10, 12, 15,
                      int(sqrt(qntNeuroniosEntrada+qntNeuroniosSaida)*2),
                      int((qntNeuroniosEntrada+qntNeuroniosSaida)/2),
                            (2*qntNeuroniosEntrada+1) ]
epocasTreinamento = [ 100, 200, 300 ]

# Arquivo que conterá toda a saída da avaliação algoritmo genético

open( logFile , "w").close()

def avaliacaoRNA(cromo):
    ''' Função responsável pela avaliação da RNA
    '''
    try:
        log = open( logFile , "a")
        c = cromo.getCromossomo()
        log.write("Treinando %s"%c)
        from os import popen
        resultado = popen(
                "python Avaliador_Fann.py %s %s %s %s %s %s %s %s" % \
                (c[0], c[1], c[2], c[3], c[4], c[5], c[6], c[7] )).
                    readlines()

        for i in resultado:
            log.write(i)

        log.write("_"*50+'\n')
        log.close()
        return float(resultado[-1].split()[-1])
    except:
```

```
            log.write("Resultado: 999.0 " \
63                      "(ERRO: Combinação de rede errada)\n")
            log.write("_"*50+'\n')
65          log.close()
            del resultado
67          return 999.0


69
    # Matriz com os tipos de genes
71  tipoGenes = [ camadas, camadas, neuroniosOcultos, trainers,
                  learningRate, momentum, lrdecay, epocasTreinamento ]
73  ''' Conteúdo do vetor tipoGenes:
        [0] Função da camada oculta
75      [1] Função da camada de saída
        [2] Quantidade de Neurônios Ocultos
77      [3] Algoritmo de Aprendizado
        [4] Learning Rate
79      [5] Momentum
        [6] Lr Decay
81      [7] Quantidade de épocas a treinar
    '''
83
    # Instância do Algoritmo Genético:
85  AG = AlgGenetico(tipoGenes, 20, avaliacaoRNA, criterioSatisfacao=0.001,
                     considMaiorAvaliacao=False, maxGeracoes=100,
87                   verboso=True)
    # Obtendo os resultados:
89  resultado = AG.evoluir()

91  # Salvando os resultados:
    arqResultado = open("Resultado_Alg-Genetico.txt","w")
93  for i in resultado:
        for c in i.getCromossomo():
95          arqResultado.write(str(c)+' ')
        arqResultado.write('\n'+str( i.getAvaliacao() )+'\n')
97  arqResultado.write('\n')
    arqResultado.close()
```

Anexo D

Código do avaliador da RNA

Este código foi desenvolvido na linguagem *Python* e criado pelo Msc. Jacson Rodrigues Correia da Silva e então adaptado para este trabalho. Ele é responsável por receber os parâmetros do Algoritmo Genético e criar a RNA MLP referente, realizando as etapas de treinamento e testes da RNA.

No final do código são retornadas as taxas de detecções como Falsos Positivos (FP), Falsos Negativos (FN), Verdadeiros Positivos (VP) e Verdadeiros Negativos (VN), além da expressão "Resultado:", seguida da taxa classificação incorreta (*error rate*).

Avaliador_Fann.py

```
# -*- coding: utf-8 -*-
'''
Created on Jan/2011

@author: Jacson RC Silva <jacsonrcsilva@gmail.com / jacson.silva@ufes.br>
'''

from sys import argv, exit

if len(argv) < 8:
    print 'Utilize: %s função-Cam.Oculta função-Cam.Saída Qnt-Neur.Ocultos \\' % argv[0]
    print 'Alg.Aprendizado LearningRate Momentum Decay' \
          'epocasTreinamento'
    exit (1)

from pyfann import libfann

cromo = argv
```

```
qntNeuroniosEntrada = 7 # adaptado para este trabalho utilizando 13 e 7 entradas #
qntNeuroniosSaida   = 2

# genes do cromossomo
nomeArqTreino = "20_percent.train"
nomeArqTeste  = "80_percent.train"
fcOculta      = int   (cromo[1])
fcSaida       = int   (cromo[2])
neurOcultos   = int   (cromo[3])
algAprend     = int   (cromo[4])
learnR        = float(cromo[5])
moment        = float(cromo[6])
decay         = float(cromo[7])
epocasTreino  = int   (cromo[8])

# abrindo arq. treino e testes
dadosTreino = libfann.training_data()
dadosTeste = libfann.training_data()
dadosTreino.read_train_from_file( nomeArqTreino )
dadosTeste.read_train_from_file ( nomeArqTeste  )

# criando rede
RNA = libfann.neural_net()
RNA.create_sparse_array(1,
                        (qntNeuroniosEntrada, neurOcultos,
                           qntNeuroniosSaida))
RNA.set_learning_rate( learnR )
RNA.set_learning_momentum( moment )

# treinamento
RNA.set_activation_function_hidden( fcOculta )
RNA.set_activation_function_output( fcSaida  )
RNA.set_training_algorithm( algAprend )
if RNA.get_training_algorithm() == libfann.TRAIN_QUICKPROP:
    RNA.set_quickprop_decay(decay)

maxIterations =epocasTreino
iterationsBetweenReports = 50
desiredError = 0.00001
try:
    RNA.train_on_data( dadosTreino, maxIterations,
            iterationsBetweenReports, desiredError )
except:
```

```
        import sys
62      print "Resultado: 999.0"
        sys.exit(1)
64
    # Avaliação pela taxa de acerto
66  entrada = dadosTeste.get_input()
    saida   = dadosTeste.get_output()
68  Normal = 0
    Normal_Ataque = 0
70  Normal_Normal = 0
    Ataque = 0
72  Ataque_Normal = 0
    Ataque_Ataque = 0
74  for i in range( len(entrada) ):
        try:
76          resposta = RNA.run(entrada[i])
            if saida[i][0] == 1.0:
78              Ataque+=1
                if resposta[0] > resposta[1]:
80                  Ataque_Ataque+=1
                else:
82                  Ataque_Normal+=1
            else:
84              Normal+=1
                if resposta[0] > resposta[1]:
86                  Normal_Ataque+=1
                else:
88                  Normal_Normal+=1
        except:
90          import sys
            print "Resultado: 999.0"
92          sys.exit(1)

94  dadosTeste.destroy_train()

96  try:
        print "FP: %d FN: %d VP: %d VN: %d (Total: %d)" %   \
98              (Normal_Ataque, Ataque_Normal, Ataque_Ataque, Normal_Normal,
                    Normal+Ataque)

100     # o resultado é a taxa de erro
        resultado = (float(Normal_Ataque)+float(Ataque_Normal))/(float(Normal
            )+float(Ataque))
102     # devolve a taxa de erro
```

```
        print "Resultado:", resultado
104 except:
        print "Resultado: 999.0"
106
    RNA.destroy()
```