ESCOLA POLITÉCNICA
DE PERNAMBUCO

DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES UTILIZANDO CLASSIFICADORES ONE-CLASS

Trabalho de Conclusão de Curso

Engenharia da Computação

Daniel dos Anjos de Oliveira Gomes
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, junho de 2006

Daniel dos Anjos de Oliveira Gomes

DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES UTILIZANDO CLASSIFICADORES ONE-CLASS

Resumo

Cada vez mais as organizações utilizam as redes de computadores para fazer diversas aplicações e obter vários objetivos. Ameaças, tentando prejudicar os computadores e as próprias redes, estão cada vez mais especializadas e difíceis de serem detectadas ou descartadas. Como componente meramente essencial em uma rede de computadores, o *firewall* é um filtro de pacotes que não tem a capacidade de detectar o mau uso da rede se esse for através de algum serviço permitido na organização, como a WEB, por exemplo. Sistemas de Detecção de Intrusão (*Intrusion Detection System* - IDS) têm a função de identificar estas ameaças para impedir o ataque analisando o conteúdo dos pacotes. Neste trabalho, propomos a utilização de classificadores *one-class,* que são técnicas de aprendizado de máquina, para utilização em IDS. Os classificadores *one-class* têm a característica de distinguir entre apenas duas classes, "normal" e "novidade", e seu treinamento é feito apenas com exemplos normais. Uma vez que, para o problema de intrusões em redes, não é preciso saber qual é a intrusão que a rede está sofrendo e sim se é ou não intrusão, esses tipos de classificadores se adequam perfeitamente ao problema. Além disso, a cada dia novos ataques são criados e, se os classificadores convencionais forem utilizados, esses ataques devem ser constantemente modelados nos sistemas IDS. Comparamos quatro técnicas levando-se em conta a métrica de Área Sob a Curva ROC (*Receiver Operating Characteristic*), AUC, e verificamos que esses classificadores têm um bom desempenho para aplicação em IDS. Os classificadores *one-class* utilizados foram Parzen, NNd, kNNd e Kmeans, onde esse último foi o que apresentou os melhores resultados para os testes realizados.

Abstract

Organizations use more and more computer networks for several applications and objectives. Threats, trying to harm the computers and the networks, are even more specialized and difficult to be found or dropped. As an essential component of a computer network, the firewall is only a packet filter that does not detect the bad use of the network if it is using an allowed service, like WEB, for example. Intrusion Detections System (IDS) has the role of identifying the threats and of avoiding an attack looking at its content. In this work, we propose the use of one-class classifiers, which are machine learning techniques that distinguish between only two classes, the "normal" and the "outliers" class. Training is done only with normal examples. In a intrusion detection problem, it is not necessary to know which is the intrusion attack that the network is undergoing. To the problem tacked here the simple fact of knowing whether the data is normal or attack is sufficient. In this way, one-class classifiers perfectly fit to this problem. Furthermore, new attacks are created every day and, if conventional classifiers are used, these classes (attacks) should be constantly modeled at the classifiers. We compared four techniques using the Area Under Curve ROC (Receiver Operating Characteristic), AUC, as the metric and we found that this classifiers have a good performance when applied to this problem. The one-class classifiers used was Parzen, NNd, kNNd e Kmeans. The Kmeans classifier was the one with the bests results to the tests.

Sumário

Índice de Figuras

Índice de Tabelas

Agradecimentos

À minha família, pelo incentivo, pelo amor e por ter sempre me apoiado, fornecendo ferramentas para o meu desenvolvimento.
Aos meus amigos e minha namorada, pela sincera amizade e amor, companheirismo, dedicação e atenção em todos os momentos.
Ao meu orientador, por ter me orientado, por esclarecer muitas vezes problemas que pareciam ser uma enorme dificuldade na confecção deste projeto, além de sempre me passar tranqüilidade e confiança na consecução do objetivo final.
A todos os professores do Departamento de Sistemas Computacionais (DSC) pelo permanente empenho em alcançar o melhor nível para o curso de Engenharia da Computação, seja no ensino, seja na infra-estrutura, seja em atividades acadêmicas, tais como pesquisas, palestras, simpósios, etc. A evolução do curso em seus primeiros seis anos de vida é, sem dúvida, surpreendente.
Por último, e mais importante, a Deus, por ter colocado todas as pessoas acima citadas no meu caminho e me iluminar em todos os momentos da minha vida.

Capítulo 1

Introdução

Este capítulo contém uma introdução ao projeto proposto, citando o problema e a motivação, o objetivo e as ferramentas utilizadas para a execução deste trabalho.

1.1 Considerações iniciais sobre o problema e conceitos utilizados neste trabalho

Cada vez mais, as aplicações computacionais das organizações estão sendo baseadas em soluções que sejam disponíveis via Internet, obtendo ganhos no mercado competitivo, adquirindo mais receita e tornando-as cada vez mais eficientes em seus procedimentos. Entretanto, essa facilidade de acesso pode comprometer todas as fontes de receita daquela organização, caso não seja dado o correto tratamento aos procedimentos de segurança inerentes, que são políticas, processos e produtos de segurança apropriados para o ambiente. Fornecer acesso remoto confiável para clientes, parceiros, funcionários remotos e móveis é um papel da segurança da informação e provê a vantagem de aumentar as fontes de receita e a produtividade dos funcionários pois provê mobilidade na medida que permite o acesso às informações de forma mais fácil e de qualquer local. As informações trocadas devem ser confidenciais, porém elas trafegam através de meios públicos ou privados os quais, às vezes, fogem do controle da organização. Possuir controles, políticas e aplicações de segurança para a redução do risco é fundamental, para garantir a proteção das informações.

Com o grande crescimento das redes de computadores nos últimos anos, a segurança passou a ser uma das maiores preocupações das organizações na área de tecnologia da informação e comunicação - TIC. Casos como perda de arquivos, roubo de informações, danos a equipamentos, dentre outros, contribuem para tornar esse quadro cada vez mais crítico. Ameaças como vírus de computadores, mensagens indesejadas (*spam*), *spywares*[1], etc. são cada vez mais freqüentes na Internet e nas redes privadas, tendendo, inclusive, a aumentar, devido ao crescimento do número de computadores interligados em rede e ao número de serviços nela disponibilizados.

[1] Programa de computador que tenta obter dados pessoais dos usuários para executar fraudes.

Segundo o *CERT*[2] *Information Center,* em dados divulgados em 2004 [5] , o número de incidentes de segurança informados aumenta significativamente a cada ano, como podemos ver na Figura 1, onde verifica-se que há um crescimento exponencial na quantidade de incidentes de segurança entre os anos de 1998 e 2003. Esses incidentes podem ser em computadores individuais, em redes pequenas ou até mesmo em conjuntos de redes, abrangendo milhares de computadores. Para cada um desses, pode haver um demasiado tempo para solucionar o problema e fazer voltar ao estado anterior, gerando perdas para a instituição.

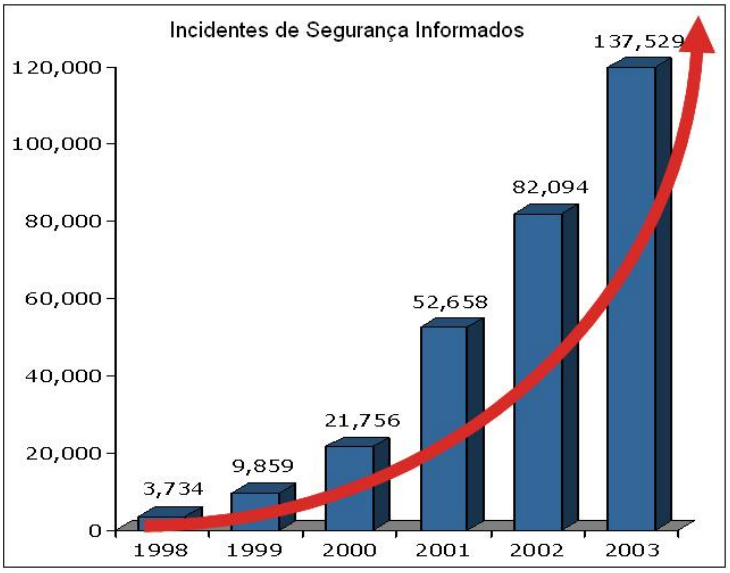

Figura 1. Incidentes de Segurança Reportados para o CERT de 1998 a 2003 [5].

Simultaneamente, os tipos de ataques às redes estão cada vez mais especializados do ponto de vista dos danos causados. Podemos ver na Figura 2 que houve uma mudança no objetivo dos ataques: se antes tínhamos ameaças à máquina local, hoje em dia as ameaças visam a prejudicar o funcionamento da infra-estrutura das redes, gerando um verdadeiro colapso, uma vez que quase todas as operações hoje ligadas a computadores são feitas através de redes. Além disso, o tempo de impacto para a ameaça se alastrar pela infra-estrutura reduziu de semanas para minutos, tendendo a ser em segundos futuramente.

Para se ter uma idéia mais precisa da evolução das ameaças, podemos citar dois casos de *worms*[3] que se espalharam recentemente pela Internet: o *Code Red* e o *SQL Slammer*. Vemos na Tabela 1 que o tempo de propagação dessas ameaças aos seus alvos diminuiu de 24 horas para apenas 30 minutos. Além do mais, a diferença de tempo entre essas duas ameaças é de apenas dois anos.

Todas essas ameaças resultam, principalmente, em risco ao mais importante fator para a maioria das organizações: receita. Como podemos citar com os dados da última pesquisa[20] realizada pela Módulo *Security*[4]*,* em outubro de 2003, onde 35% das empresas pesquisadas reconheceram que tiveram perdas financeiras no mesmo ano (as demais não souberam quantificar). Desse total, 22% registraram perdas de até R$ 50 mil, 8% entre R$ 50 mil a R$ 500 mil e 4% de R$ 500 mil a R$ 1 milhão.

---

[2] CERT – *Center of Internet Security Expertise*. http://www.cert.org.

[3] *Worm*: programa auto-replicante, semelhante a um vírus.

[4] Módulo *Security*: www.modulo.com.br

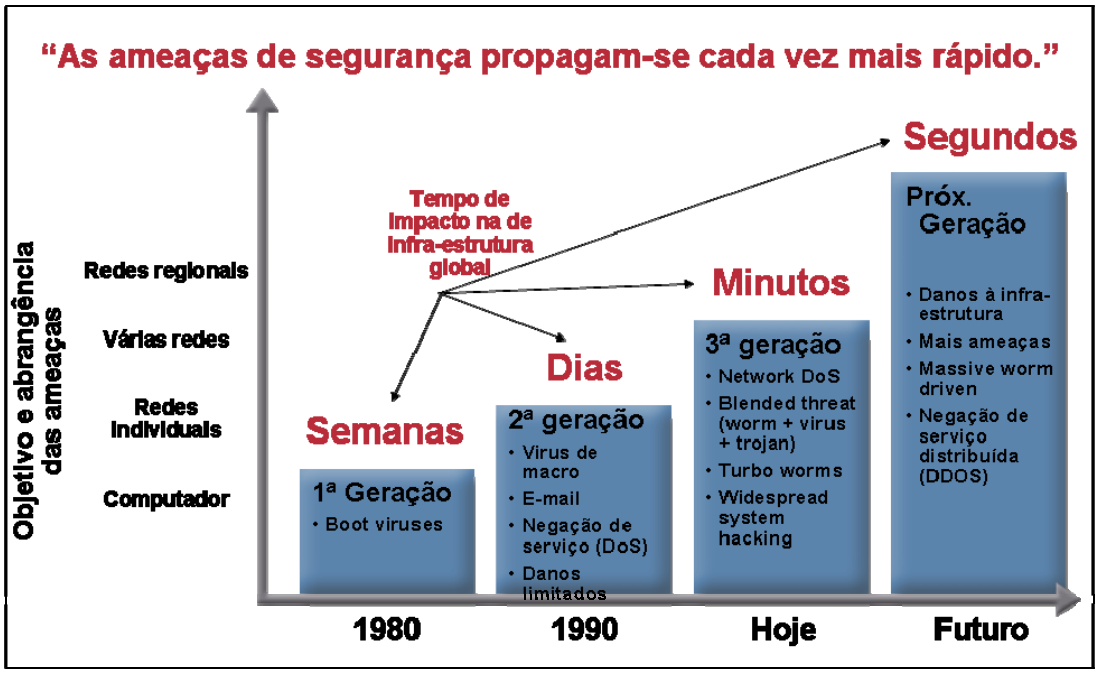

Figura 2. Evolução dos tipos de vírus levando em consideração a velocidade de propagação.

Tabela 1. Evolução das ameaças levando em consideração a velocidade de propagação.

	2001 - Code Red	2003 - SQL Slammer
Taxa inicial de infecção por hora	1,8 PCs	420 PCs
Tempo para duplicar número de PCs	37 minutos	8,5 segundos
Tempo para infectar todos os alvos vulneráveis	24 horas	30 minutos

Em se tratando de fraudes pela Internet, os brasileiros têm uma preocupação a mais: segundo dados divulgados pela Polícia Federal em 2004 [1], 80% dos hackers do mundo todo são brasileiros. Nessa mesma pesquisa, foi divulgado que, no Brasil, as fraudes financeiras que utilizam a Internet e correios eletrônicos já superam, em valores financeiros, os prejuízos de assalto a banco.

Algumas técnicas são utilizadas para proteger informações importantes, tais como *firewall*, criptografia, Redes Privadas Virtuais (*Virtual Private Networks* - VPNs), sistemas detecção de intrusos, etc. O *firewall* age como uma defesa para proteger dados sensíveis, mas é meramente um redutor de exposição, não monitorando ou eliminando as vulnerabilidades nos sistemas de computadores. Algumas ameaças como *worms*, *spyware/adware*, vírus de rede e abuso de aplicações ou negação de serviço, nem sempre são detectados pelos *firewalls*, pois utilizam portas padrões de serviços existentes na rede, como a WEB, por exemplo. Qualquer nova técnica de segurança pode ter falha de projeto. Pior que isso, a cada novo aplicativo criado, há uma grande probabilidade desse ter uma falha de segurança. Com a criptografia, podem-se cifrar dados e evitar visualizações não-autorizadas aos dados. Por sua vez, as VPNs podem ser utilizadas para interligar duas empresas fisicamente distantes e permitir acesso aos recursos uma da outra de forma segura, podendo inclusive utilizar a criptografia para atingir esse objetivo. Os sistemas de detecção de intrusão, como sugere o nome, visam a detectar conexões não permitidas na rede.

Uma intrusão na rede pode ser definida como qualquer conjunto de ações que tentem comprometer a integridade, confidencialidade ou disponibilidade dos dados e/ou do sistema [16].

Detecção de intrusão é o processo de identificar e relatar atividade maliciosa agindo em computadores e recursos da rede [2]. Sistemas de Detecção de Intrusão (*Intrusion Detection Systems* - IDS) são responsáveis por detectar tal tipo de tráfego.

Um sistema IDS é um aplicativo de monitoramento e detecção de tráfego de dados e comportamento de usuários com o intuito de identificar comportamento anormal (e em alguns casos prevenir). Existem dois métodos básicos para detectar tal comportamento:

1. Detecção de mau-uso.
2. Detecção de anomalia.

A detecção do mau-uso primeiro tenta modelar padrões específicos de intrusões num sistema. Então, sistematicamente, filtra o tráfego na rede para encontrar essas ocorrências. Uma vez que o sistema deve conhecer o ataque antes de modelá-lo, esse método é, na maioria das vezes, usado para detectar intrusões já previamente conhecidas pelo IDS.

Detecção de anomalia cria um perfil de atividades de tráfego ou comportamento de usuários normais, então compara com o perfil da rede para decidir se o tráfego é normal ou não. Essa decisão é baseada num limiar que pode ser ajustado para chegar ao melhor desempenho da rede. A detecção de anomalias na rede é um problema de detecção de ataques que não estão pré-definidos no IDS.

Um sistema IDS bem projetado deve ter a facilidade de detectar tanto mau uso quanto ataques de anomalias, ou seja, ataques “desconhecidos”.

Este projeto tem o objetivo de fazer um estudo sobre os classificadores *one-class* (classificadores de uma única classe) aplicados à detecção de intrusão em redes de computadores. Esses classificadores têm a função de distinguir entre objetos de uma determinada classe e todos os outros objetos. Nesse caso, é assumido que somente exemplos da classe normal estão disponíveis para treinamento. Assim, todo o treinamento é feito com apenas uma classe e o classificador deve aprender o comportamento normal e distingui-lo das demais classes de comportamentos anormais.

A justificativa da escolha dos classificadores *one-class* neste trabalho deve-se ao fato de que, em problemas de detecção de intrusão em redes de computadores, não necessariamente se precisa saber qual o tipo de ataque que a rede está sofrendo. Para o objetivo do IDS, apenas identificar um ataque e ter alguma maneira de bloqueá-lo, já é suficiente. Levando-se em conta essa premissa, torna-se mais fácil utilizarem-se classificadores de uma única classe, ao invés de classificadores convencionais, por diversos motivos. Como exemplo, pode-se citar a maior facilidade no treinamento, pois só precisa-se treinar o classificador com amostras de uma classe (normal), não precisando se preocupar, além disso, com classes desbalanceadas (poucas amostras de um tipo e várias de outro), uma vez que o treinamento é feito com uma só classe. Uma outra vantagem da utilização de classificadores *one-class* no problema de detecção de intrusão em redes de computadores, é que o aplicativo IDS com esse classificador é muito “estático” pois não é necessária a sua atualização com novas vacinas e assinaturas tornando-o mais simples, na prática.

Em pesquisas preliminares, não encontramos nenhum outro trabalho que tenha utilizado os classificadores *one-class* utilizados neste trabalho para a aplicação no problema de detecção de intrusão em redes de computadores, o que pode ser justificado pelo fato de que a utilização de classificadores *one-class* para problemas de aprendizado de máquina ainda não está difundida, por ser uma aplicação recente. Não foi possível a comparação dos dados obtidos neste projeto com outros, pelo mesmo motivo. Assim, esta monografia é apresentada como uma novidade no âmbito do aprendizado de máquina aplicado na detecção de intrusão em redes de computadores.

Utilizamos quatro classificadores distintos aplicados a uma base de dados de detecção de intrusão que contém dados de tráfego normal e também 22 tipos de intrusão em redes. Para avaliação de desempenho, utilizamos a curva *Receiver Operating Characteristic* (ROC) para

comparação do classificador com outros, que será explicada na Seção 3.1. A curva ROC fornece uma medida de eficiência para problemas de classificação binários. O trabalho explora a utilização de classificadores *one-class*, que são métodos de classificação binários, por distinguirem entre apenas duas classes, e utiliza curvas ROC para ilustrar a eficiência dessa técnica e calcular a área sob a curva (AUC), que é utilizada como a métrica de comparação entre os classificadores neste trabalho.

1.2 Roteiro deste trabalho

No Capítulo 2, apresentamos conceitos teóricos de sistemas de detecção de intrusão, além de apresentar uma solução atual de mercado para tais sistemas.

No Capítulo 3, apresentamos curvas ROC e classificadores *one-class*, incluindo os mais utilizados métodos de classificação: Densidade, Fronteira e Reconstrução. Além disso, esse capítulo contempla uma justificativa da escolha de classificadores *one-class* para a aplicação de detecção de intrusão.

No Capítulo 4, apresentamos os experimentos realizados nesse projeto utilizando classificadores dos 3 métodos acima, com uma base de dados de detecção de intrusão.

Finalmente, no Capítulo 5, colocamos nossa conclusão sobre este projeto além de sugestões de novas pesquisas nessas aplicações.

Capítulo 2

Sistemas de Detecção de Intrusão

Este Capítulo fornece uma visão geral de Sistemas de Detecção de Intrusão, mostrando componentes, funcionamento e apresentando uma solução real disponível no mercado, atualmente.

2.1 Conceito

Detecção de Intrusão pode ser definida como um conjunto de técnicas e métodos que são utilizados para detectar atividades suspeitas, tanto na rede como na máquina a que ela está ligada. Pode-se enquadrar em duas categorias:

a. Detecção baseada em assinaturas
b. Detecção baseada em anomalias.

Técnicas de intrusão baseadas em assinaturas têm padrões que podem ser detectados utilizando aplicativos, assim como os vírus. Tenta-se encontrar pacotes de dados que contêm qualquer assinatura relacionada à intrusão ou anomalias relacionadas a protocolos de rede. Baseado em um conjunto de assinaturas e regras, um sistema de detecção é capaz de encontrar e relatar atividades suspeitas, podendo, inclusive, gerar alertas.

Detecção de intrusão baseada em anomalia normalmente depende das anomalias de pacotes presentes em partes dos cabeçalhos do protocolo. Em alguns casos, esse método produz resultados melhores, comparados aos sistemas baseados unicamente em assinaturas. Usualmente, um sistema de detecção de intrusão captura dados da rede e os submetem a regras para depois verificar se há anomalias nesses dados.

2.2 Definições:

IDS: Sistema de Detecção de Intrusão ou *Intrusion Detection System* é um software ou conjunto de hardware e software que é usado para detectar atividades de intrusos. Um IDS pode ter diferentes funcionalidades, dependendo de quão complexos e sofisticados os seus componentes são. Podem, por exemplo, além de detectar atividades anormais, prevenir sua atuação caso esteja configurado de forma que assim o permita.

IDS de Rede ou *Network IDS* (NIDS): São sistemas de detecção de intrusão que capturam pacotes de dados trafegando na rede e os compara com uma base de dados de assinaturas. Caso essa comparação tenha um resultado positivo, pode ser gerado um alerta e um registro que pode resultar em uma ação.
IDS de *Host*: Também chamados de *Host IDS* (HIDS), são instalados como agentes em um host. São responsáveis por procurar no sistema e nos logs de aplicativos para detectar alguma atividade de intrusos.
Assinaturas: Uma assinatura é um padrão presente em um pacote que pode indicar atividades suspeitas. Uma assinatura é utilizada para detectar um ou múltiplos tipos de ataques. Por exemplo, a presença do comando "cmd.exe[5]" numa requisição a um servidor WEB, pode significar uma atividade de intrusão.

Assinaturas podem estar presentes em várias partes de um pacote, dependendo da natureza do ataque. Por exemplo, é possível haver assinaturas em um cabeçalho IP, em um cabeçalho TCP ou num cabeçalho de um pacote HTTP, por exemplo. Dessa forma, obviamente, um IDS deve trabalhar nas camadas 2 a 7 (enlace a aplicação) do modelo OSI[6].

As assinaturas de um IDS são atualizadas com novos tipos de ataques na medida que esses vão sendo descobertos e implementados nos sistemas.
Alerta: É qualquer tipo de notificação aos usuários acerca de atividades de intrusos. Quando um IDS detecta uma intrusão, esse tem que informar ao responsável pela segurança utilizando esses alertas. Várias são as maneiras de transmissão desses mecanismos. Podem ser por janelas de *pop-up,* registrando em uma console, enviando um e-mail, etc. Esses alertas também são armazenados em registros no banco de dados interno dos IDS, para que possam ser analisados posteriormente ou enviados a ferramentas específicas para tal, como veremos adiante na arquitetura da Cisco para soluções de IDS.
Alarmes Falsos: São alertas falsos sobre uma suposta atividade de intrusão que o IDS detectou como tal, devido a um comportamento que foi classificado, erroneamente, como ataque.
Sensor: A máquina que o IDS está sendo executado. No caso de NIDS, é um hardware feito com fins exclusivos para essa aplicação. Para o HIDS, o sensor é o próprio computador que o IDS está instalado.

2.3 Localização do IDS na rede

Dependendo da topologia da rede, podemos colocar o IDS em um ou mais locais. Isso também depende de quais tipos de atividades de intrusões desejamos detectar: se interna, externa ou ambas. Tomamos como exemplo o fato de querermos monitorar apenas as atividades de intrusões externas e supondo que tenhamos apenas um roteador conectando-nos à Internet. Nesse cenário, o melhor local para o IDS seria logo após o roteador ou o firewall (caso exista) no sentido da Internet para a rede interna. Num outro cenário, se há várias interligações à Internet ou a outras redes consideradas inseguras, podemos colocar IDS em cada um desses pontos. Se, por outro lado, desejássemos monitorar atividade interna da rede também (tráfego entre estações, por

[5] CMD.EXE é o programa do Microsoft Windows que permite ao usuário executar comandos no *prompt* de comando do sistema.

[6] *Open System Interconnection*: Arquitetura em camadas para referência de padronização para equipamentos de redes de computadores.

exemplo), poderíamos colocar um IDS em um segmento de rede, monitorando uma porta espelhada[7] de um *switch*.

Como pode-se ver na Figura 3, deve-se colocar o IDS depois do firewall ou roteador no sentido Internet - Rede Interna. O motivo pelo qual não é recomendável colocar o equipamento antes do firewall é que este já tem capacidade de bloquear grande parte das tentativas de ataque, ou seja, colocando o IDS nesse local, aumentaríamos a necessidade de capacidade de processamento nele, por um motivo desnecessário.

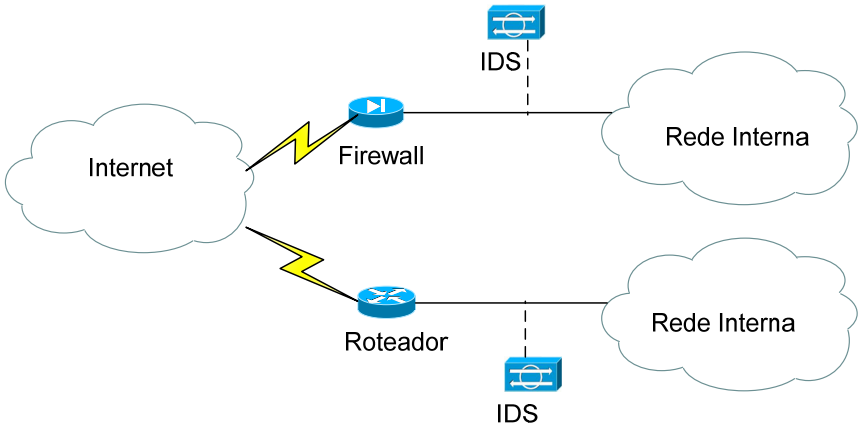

Figura 3. Localização correta de um IDS na topologia da rede.

2.4 Componentes de um IDS

Um IDS é logicamente dividido em vários componentes, que trabalham em conjunto, para detectar ataques e gerar saídas em formato específico, que pode ser utilizada por outros equipamentos como firewalls, aplicativos de correlação de dados entre vários IDS, dentre outros.

O IDS contém basicamente os cinco componentes a seguir, que também estão mostrados na Figura 4:

- Decodificador de pacotes;
- Pré-processador;
- Motor de detecção;
- Sistema de registros e alertas;
- Módulo de saída .

Decodificador de Pacotes:
Recebe os pacotes através das interfaces de rede e os prepara para serem pré-processados. Essa preparação é, geralmente, o desencapsulamento do pacote da tecnologia recebida que pode ser *Ethernet*, PPP, etc.

Pré-processador:
É o componente que permite fazer alguma modificação ou arranjo nos dados antes que estes passem para o motor de detecção. Alguns pré-processadores podem também encontrar anomalias

[7] Alguns switches (comutadores de rede) têm a funcionalidade de espelhar o tráfego de uma ou mais portas em uma outra porta que é chamada de porta de monitoramento. Esta porta recebe todos os datagramas que estão sendo recebidos e enviados através das portas monitoradas.

em cabeçalhos de pacotes e gerar alertas. Alguns tipos de ataques são camuflados por mudanças que o atacante pode fazer com relação a código de comandos, sintaxe, etc. Por exemplo, ao invés de enviar uma mensagem contendo o comando "scripts/iisadmin[8]" para um servidor, o pacote pode vir com o Identificador de Recurso Uniforme (*Uniform Resource Identifier – URI*) correspondente ao comando citado. Nesse exemplo, o pré-processador seria responsável por traduzir o URI para o comando.

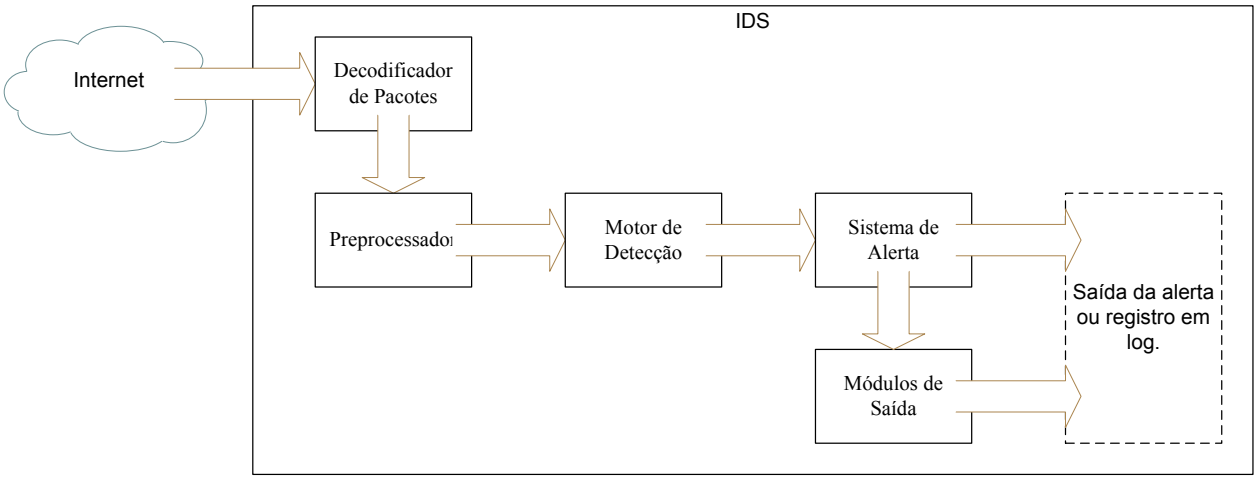

Figura 4. Os cinco componentes lógicos de um IDS.

Motor de Detecção:
Responsável por detectar se alguma atividade de intrusão existe no(s) pacote(s), é a mais importante parte do IDS. Dependendo do IDS, conforme já dito anteriormente, o método utilizado pode ser baseado em assinaturas, em anomalias ou ambos. Se baseado em regras (assinaturas), os pacotes são comparados com cada uma delas, para, se positivo, realizar uma ação. Se baseado em anomalias, o pacote passa pelo teste, que é baseado na técnica utilizada para essa detecção.

Esse é o componente em que o pacote leva mais tempo para ser processado. Dependendo da capacidade de processamento da máquina que o IDS está sendo executado, pode ser gerado um gargalo e conseqüente atraso, no tráfego de dados da rede. Assim, os IDS, em geral, têm a característica de não impedir que o tráfego flua, caso ele esteja sobrecarregado. O que ele faz é não verificar os pacotes que estão passando, quando o seu processamento está no limite. Isso, por sua vez, pode ocasionar a passagem de dados contendo intrusões.

Sistema de Alerta:
Para atingir o objetivo final de uma solução de segurança, que é evitar intrusões, o IDS precisa alertar algum outro componente (que pode estar até nele mesmo) que será responsável por executar alguma ação, caso o sistema esteja bem configurado. Baseado no retorno que o Motor de Detecção passar, o sistema de alerta poderá enviar tal mensagem.

Módulos de Saída:
Presentes em alguns sistemas, esses módulos podem gerar ações diferentes de simples mensagens a serem enviadas para outros sistemas (função do sistema de alerta). Para sistemas que também

[8] IISADMIN é uma interface WEB que compreende o Internet Information Services (IIS) da Microsoft que pode ser utilizada para administrar o serviço. O IIS é o serviço de divulgação na Internet da Microsoft que pode ser instalado nos Windows 2000, XP ou 2003 e publicam sites, provêem serviço FTP, etc.

têm a funcionalidade de prevenir ataques, evitando deixar que o tráfego passe através dele, há um módulo que bloqueia os pacotes, por exemplo. Além disso, esses módulos podem, dependendo do sistema, enviar alertas através do protocolo SNMP[9], enviar mensagens para banco de dados, modificar configurações de roteadores, *firewalls* e *switches*, dentre outras várias funcionalidades.

2.5 Solução de mercado atualmente disponível: arquitetura da solução Cisco

Nesta Seção mostraremos a arquitetura de uma solução de IDS, disponibilizada atualmente pelo fabricante Cisco, que é um dos mais reconhecidos na área.

Atualmente, os principais produtos do fabricante Cisco para sua solução de detecção de intrusão são os da série Cisco IPS 4200[10]. Esses produtos oferecem proteção para as redes, ajudando a detectar, classificar e parar ameaças à rede, incluindo *worms, spyware/adware,* vírus de rede e abuso de aplicações ou negação de serviço. Os equipamentos citados têm a função exclusiva de detectar e/ou prevenir os ataques. Funcionando *inline,* isto é, no "meio" do canal de dados, o IPS 4200 pode também bloquear o tráfego malicioso. Assim, funciona, não só como um sistema de detecção, mas também de prevenção de intrusos (*Intrusion Prevention System* - IPS).

Além da série IPS 4200, a Cisco oferece a opção de colocar o IDS integrado no software dos roteadores. Dessa forma, o roteador, além de fazer a função de roteamento da rede, implementa segurança contra tráfego malicioso.

Por último, há os equipamentos da família *Cisco Adaptative Security Appliance (ASA) 5500.* Esses hardwares podem implementar funções de firewall, IPS, antivírus de rede e servidor de VPN (redes privadas virtuais – *virtual private networks*) num mesmo equipamento. Para as estações de trabalho e servidores, a Cisco fornece o *Cisco Security Agent*[7], que é um software para proteção de ameaças diretamente instaladas no sistema operacional da máquina.

Para ambientes mais complexos e maiores, onde são necessários vários equipamentos de segurança entre firewalls, IDS, roteadores com IDS etc., o fabricante oferece *softwares* que servem para configurar automaticamente os equipamentos de segurança da rede, seja por uma intervenção do operador, seja por uma detecção de alarme que o aplicativo de gerência entende como uma ameaça. Nesse caso, o sistema envia mensagens para os equipamentos de segurança afetados, reconfigurando-os, para evitar maiores danos à infra-estrutura da organização. São exemplos destes softwares:

- *Cisco Security Monitoring, Analysis, and Response System (MARS)*: Solução baseada em hardware que permite aos administradores de rede e de segurança monitorar, identificar, isolar e contar ameaças de segurança[8]. Essa ferramenta tem a função de analisar eventos dos dispositivos de segurança da rede, para detectar anomalias e eventos de segurança. Após a análise, a ferramenta define ações a serem tomadas que pode gerar a emissão de alarme, reconfiguração automática de equipamentos, dentre outras.
- *Cisco Incident Control System (ICS):* A partir de uma parceria entre a Cisco e a Trend Micro, um dos líderes mundiais na fabricação de antivírus, o ICS mantém-se atualizados

[9] *Simple Network Management Protocol*: é um protocolo de camada de aplicação que facilita a troca de informações de gerenciamento entre equipamentos de rede. Ele nos permite gerenciar o desempenho da rede, encontrar e resolver problemas nela, dentre outras várias funcionalidades.

[10] Site do produto disponível em: http://www.cisco.com/en/US/products/hw/vpndevc/ps4077/index.html.

com as mais recentes vacinas disponíveis, todos os equipamentos IDS instalados em uma organização[6]. Na Figura 5, podemos ver uma topologia de uma solução com o Cisco ICS. Como pode-se ver, existe um servidor na rede com o aplicativo ICS, que é responsável por manter atualizados todos os IDS e/ou equipamentos de segurança presentes na rede, estejam eles embutidos nos switches, nos roteadores ou sejam eles os próprios IDS/IPS. Essas atualizações podem ser tanto reconfigurações de regras de acesso, tais como em filtros de pacotes (*firewall*), como também podem ser atualizações das vacinas/assinaturas presentes nos sistemas IDS. As novas assinaturas são fornecidas pelo fabricante TrendMicro em uma parceria com a Cisco.

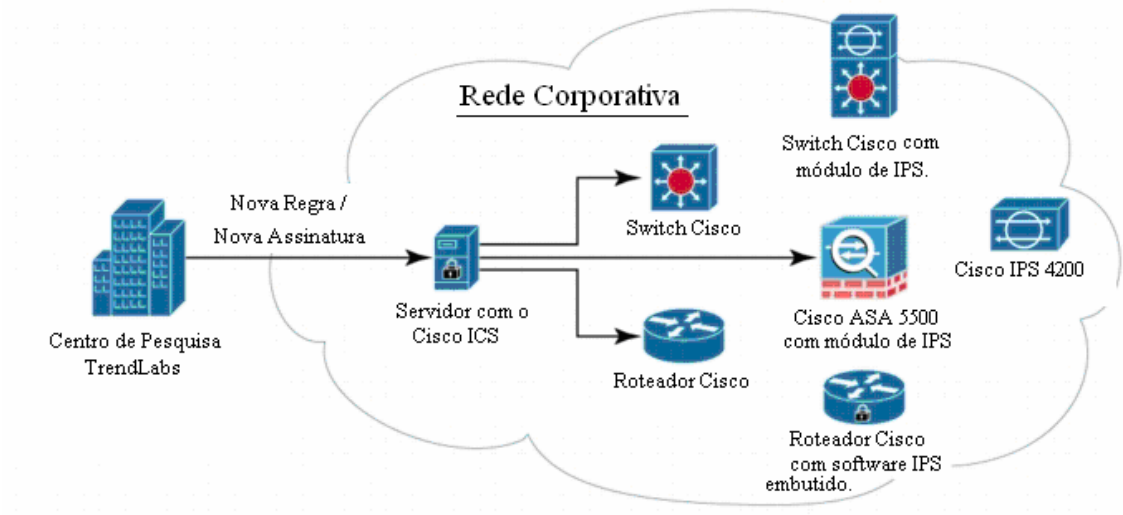

Figura 5. Topologia da arquitetura da solução de mercado Cisco ICS.

Existem outros softwares disponibilizados pelo fabricante Cisco que têm funções semelhantes, mas que não serão citados aqui por não ser este o objetivo principal deste trabalho.

Para redes maiores e complexas, como exemplo, uma arquitetura de segurança da Cisco pode ser composta de:

1 - Equipamento de IDS: equipamentos da série IPS 4200, roteadores com função de IPS, os equipamentos da família ASA 5500 e/ou os agentes para estações e servidores, o *Cisco Security Agent* [7].

2 – Aplicação de correlação de eventos e atualização de vacinas: Cisco MARS e Cisco ICS.

Na Figura 6, damos um exemplo da topologia da solução citada para grandes empresas, baseado em soluções de segurança do fabricante Cisco. Verificamos uma integração entre os componentes da solução, de forma que são gerenciados centralizadamente. No exemplo da figura, temos um servidor com o Cisco MARS, que recebe alarmes vindos de todos os equipamentos de segurança da rede. O aplicativo faz uma correlação desses alarmes para definir ações a serem tomadas e as envia para os equipamentos da rede. Algumas vezes, um simples alarme em um único equipamento pode não significar alguma coisa. Entretanto, vários alarmes, vindos de diferentes origens, podem indicar uma ação de intrusão, com a necessidade de novas configurações para a impedir. Neste cenário que está baseada a solução da Figura 6. Essa solução tem um ciclo permanente onde o Cisco MARS se mantém realizando:

1. Agregação e pré-processamento dos alarmes;
2. Detecção de ações suspeitas na rede;
3. Correlação dos eventos recebidos;
4. Reconfiguração de equipamentos da rede.

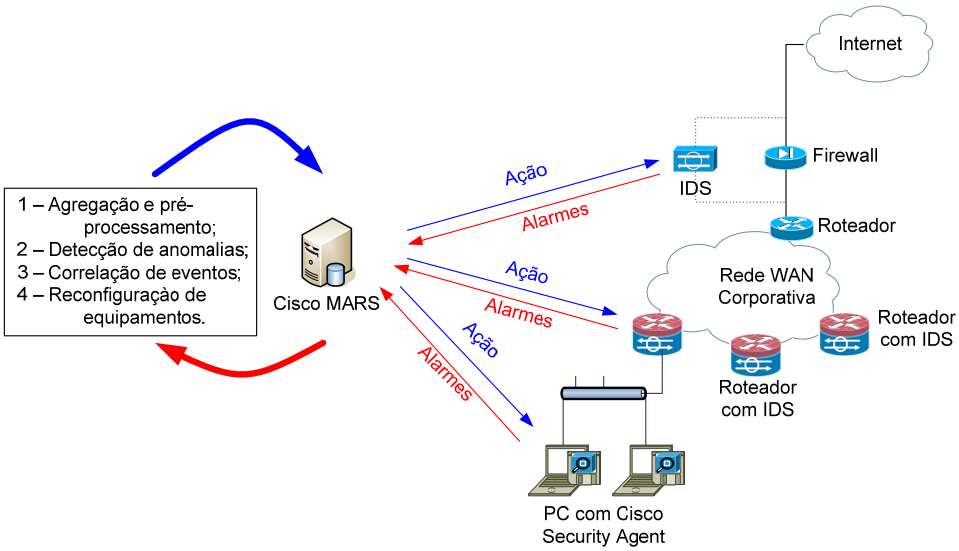

Figura 6. Exemplo de topologia utilizando Cisco IPS, Roteadores com IDS e os agentes *Cisco Security Agents*.

Apesar da utilização de todas as técnicas e aplicativos mencionados acima para a detecção de intrusão em redes, a Cisco ainda não utiliza Detecção de Anomalias através de aprendizado de máquina, em suas arquiteturas de IDS. O fabricante utiliza reconhecimento de padrões como principal meio para a aplicação do IDS. Esse fato pode ser considerado uma limitação do sistema, uma vez que sempre que novos ataques são criados, estes devem ser descobertos, modelados no sistema operacional dos equipamentos (geração das assinaturas) de segurança e enviados a esses, para suas atualizações. Os argumentos da Cisco para a não utilização de detecção de anomalias através de aprendizado de máquina em seus produtos são:

1. Sistemas baseados nessas técnicas não informam a granularidade do ataque que foi detectado. Em outras palavras, o sistema não diferencia entre ataques com uma baixa criticidade e ataques seriamente críticos.
2. A flexibilidade dos sistemas é muito pequena com relação à detecção de ataques ou tráfego normal.
3. O método é altamente dependente do ambiente a que está instalado. Dessa maneira, torna-se difícil definir o que é considerado tráfego normal.

Embora seus argumentos sejam coerentes, existem formas de minimizar os problemas citados pela Cisco, por exemplo, combinando IDS baseado em assinaturas e também baseado em anomalia. Outra opção, é combinar vários tipos de classificadores baseados em anomalias para aumentar a eficiência e segurança dos resultados. Esse último exemplo é um trabalho futuro proposto nesta monografia, como explicado na Seção 5.3. Dessa forma, é possível sim, além de viável, a utilização de detecção de intrusão baseada em aprendizado de máquina.

Capítulo 3

Classificadores *One-Class*

Este Capítulo tem o intuito de explanar conceitos sobre classificadores *One-Class* em geral, sobre os classificadores *one-class* utilizados neste trabalho e sobre o método para avaliação de desempenho dos classificadores, que foi a Área sob a Curva (*Area Under Curve* - AUC) baseada em curvas ROC. Por fim, neste capítulo apresentamos nossa proposta de tratar o problema de Detecção de Intrusão como um problema de classificação *one-class*.

3.1 Curva ROC e Área Sob a Curva (*area under curve* - AUC)

Curvas *Receiver Operating Characteristics* (ROC) é uma técnica para visualizar, organizar e selecionar classificadores, baseada em seus desempenhos [12]. Esta técnica já é utilizada em outras áreas há algum tempo e tem crescido a sua utilização em aplicações de aprendizado de máquina.

Curvas ROC têm sido utilizadas há muito tempo na teoria dos sinais para descrever o compromisso entre taxas de acerto (probabilidade de um acerto) e taxas de falsos alarmes[11] (probabilidade de um falso alarme) de classificadores [10][29]. Uma outra área que vem utilizando Curvas ROC há muito tempo é a área médica. Sua análise é utilizada para visualizar e analisar o comportamento de sistemas de diagnóstico bem como a utilização em aplicativos de decisões médicas [30].

Ultimamente, há um aumento na utilização de ROC em aplicações de aprendizado de máquina devido a pesquisas realizadas que mostram que a métrica convencional de precisão de classificação (cálculo simples da taxa de erro de classificação) não é a mais apropriada para medições de desempenho nessas aplicações de aprendizado [25][26].

Devido ao fato de só termos duas classes possíveis (Verdadeiro ou Falso) para a elaboração da curva ROC, essas só podem ser utilizadas para problemas de duas classes. Para utilizar curvas ROC em problemas multiclasses, devemos reduzi-los a vários problemas de duas classes. Em [14], são apresentados mais detalhes de como fazer essa operação.

[11] Falso Alarme é quando uma amostra falsa é classificada como verdadeira por algum classificador.

Na Figura 7, temos uma matriz de confusão para possíveis classificações para problemas de aprendizado de máquina. Dado um padrão (exemplo) de teste e um classificador, a classificação se enquadra em quatro possíveis categorias:

- Se o exemplo for positivo e assim for classificado, temos um verdadeiro positivo (*true positive* – **TP**).
- Caso o exemplo seja positivo, mas tenha sido classificado como negativo, temos um falso negativo (*false negative* – **FN**).
- Por outro lado, sendo o exemplo negativo e assim sendo classificado, temos um negativo verdadeiro (*true negative* – **TN**).

Se o exemplo negativo for classificado positivo, é chamado de falso positivo (*false positive* – **FP**).

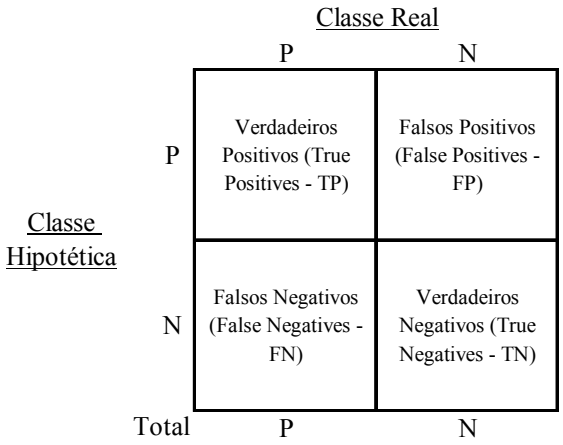

Figura 7. Matriz de confusão com possibilidades de classificação para um classificador.

Com essas classificações, temos duas taxas que serão utilizadas na confecção da curva ROC:

$$taxa\,FP = \frac{FP}{N} \tag{3.1}$$

$$taxa\,TP = \frac{TP}{P} \tag{3.2}$$

onde, P e N são, respectivamente, o número total de amostras positivas e negativas na classe real.

$$P = TP + FN \tag{3.3}$$

$$N = TN + FP \tag{3.4}$$

Pelas equações 3.3 e 3.4 acima, fica clara a composição de P (amostras positivas) e de N (amostras negativas).

Curvas ROC são gráficos bi-dimensionais onde a taxa TP de verdadeiros positivos - TP (*tp rate)* é mapeada no eixo Y e a taxa FP de falsos positivos - FP (*fp rate)* é mapeada no eixo X. Assim, um gráfico ROC representa compromissos entre benefícios (taxa TP) e custos (taxa FP) [12], como podemos ver no exemplo dado na Figura 8. Observando esse exemplo, citamos alguns pontos importantes de saber no espaço ROC:

- O ponto (0,0) representa um ponto onde o classificador não dará nenhuma classificação FP. Da mesma forma, esse classificador não fornecerá nenhuma

classificação TP, significando que não dará classificações positivas. Assim, vemos que todas as amostras nesse caso serão classificadas como negativas, sejam elas verdadeiramente negativas (TN) ou sejam elas positivas mas que foram classificadas negativamente (FN). Assim, caso esse classificador analise uma base com amostras positivas, teremos sempre estas classificadas erroneamente (FN).

- Ao contrário desse, o ponto (1,1) representa o local onde o classificador fornecerá classificação positiva de qualquer forma, seja um TP ou um FP, não classificando nenhuma amostra negativamente. Para o caso do conjunto haver amostras negativas (o que é muito provável), esse ponto é ruim, pois vai classificar todas como positivas, ou FP, pois, pela equação 3.4, vemos que se o FP for igual a 1, o TN será sempre igual a 0.
- O ponto (0,1) é a classificação ideal, ou seja, máxima quantidade de amostras positivas classificadas verdadeiramente (TP) e nenhuma amostra negativa classificada como positiva (FP). Isso não quer dizer, no entanto, que o classificador não tenha classificado nenhuma amostra como negativa. Na verdade, o fato de TP ser igual a 1 e FP ser igual a 0, significa que TN tem que ser igual a 1, como podemos verificar nas equações 3.3 e 3.4. Assim, qualquer amostra negativa será classificada como tal, nesse ponto.

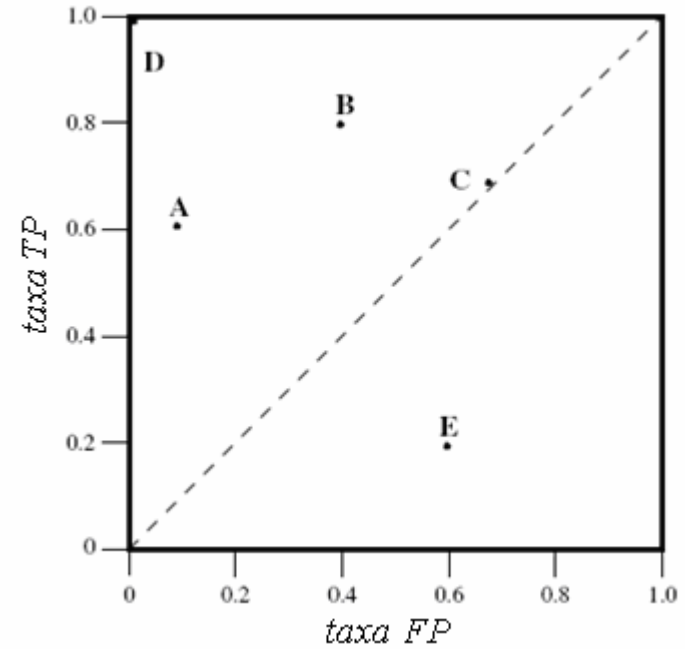

Figura 8. Exemplo de gráfico ROC apresentando cinco classificadores discretos.

Dessa forma, podemos dizer que um ponto é melhor que outro em um espaço ROC se ele está mais à esquerda e para cima desse, significando que a taxa TP é maior e/ou a FP menor. No exemplo da Figura 8, onde temos pontos de classificadores discretos, o classificador A é melhor que o C, por exemplo. Ambos classificadores A e C são melhores que o classificador E. No mesmo exemplo, o classificador D é considerado o melhor classificador (mais à esquerda e acima).

Ainda podemos dizer que classificadores que estão do lado esquerdo do gráfico, perto do eixo X podem ser considerados “conservadores”, pois fazem classificações positivas somente com alta evidência fazendo, assim, poucos FP. Classificadores no lado direito superior do gráfico podem ser chamados de “liberais”, pois fazem classificações positivas com pouca evidência

tendo, entretanto, altas taxas de FP [12]. Voltando ao exemplo da Figura 8, o ponto A é mais conservador que B (que, consequentemente, é mais liberal que o anterior).

A linha y=x (diagonal reta no gráfico) na Figura 9 representa um classificador que classificaria aleatoriamente os exemplos, isto é, tem as mesmas taxas de TP e de FP. O ideal é que todos os classificadores estejam no triângulo esquerdo superior do gráfico, pois, como visto, os que estiverem no triângulo direito inferior classificarão com um desempenho inferior a um classificador aleatório. Qualquer classificador na diagonal y=x pode ser dito que não tem nenhuma informação sobre a classe. De um classificador abaixo da diagonal, pode ser dito que tem informações úteis, porém está aplicando-as incorretamente [13].

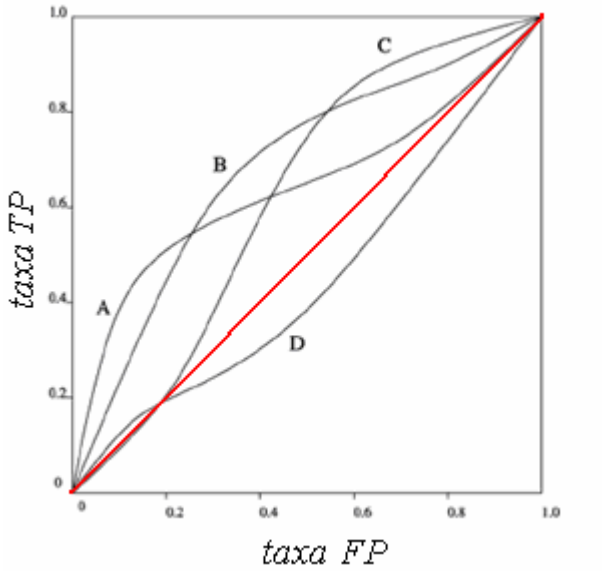

Figura 9. Gráfico ROC apresentando quatro curvas relativas a classificadores e a diagonal y=x, que representa um classificador que classifica aleatoriamente.

Classificadores como os utilizados na Figura 8, nos fornecem somente uma distribuição de classes e matrizes de confusão, permitindo definir um ponto apenas referente às suas taxas TP e FP. Por isso, são chamados de classificadores discretos. Já para modelos que produzem uma saída contínua, como a estimativa de probabilidade em Redes Bayesianas [11], as taxas de TP e FP variam à medida que o limiar da saída varia entre seus extremos 0 e 1, gerando uma curva ROC. Na Figura 9, temos quatro exemplos de classificadores desse tipo que são representados, um a um, pelas curvas A, B, C e D.

Na Figura 9, podemos dizer que a curva B domina a curva D, pois ela está mais à esquerda e acima que esta última. Isso significa que o classificador B terá um custo menor (menor taxa FP) que o D. Já entre as curvas A e C não existe uma relação clara de qual delas domina, pois, em um determinado intervalo, A está mais à esquerda e acima e, em outro intervalo, isso acontece com C. Nesse caso, não podemos comparar visivelmente os classificadores A, B e C.

A maneira mais utilizada para comparar curvas ROC é o cálculo da área sob a curva (*area under curve* – AUC) [4][15]. AUC é uma medida escalar variável compreendida no intervalo entre 0 e 1 que torna a comparação de curvas uma medida muito simples. A partir desse número, experiências podem ser realizadas em busca do classificador que produza o maior valor de AUC, que é considerado o melhor classificador. Dados dois exemplos aleatoriamente escolhidos, um positivo e um negativo, AUC representa a probabilidade de o exemplo negativo ter uma

probabilidade menor de pertencer à classe positiva que o positivo vai ter de pertencer à classe negativa.

Voltando ao cenário de aprendizado de máquina, em [17] é mostrado teoricamente e empiricamente que o AUC é uma medida melhor que a de precisão (cálculo simples da taxa erro de classificação) na comparação de algoritmos nessa área.

3.2 Classificadores *one-class*.

Estamos acostumados com o modo de classificação para aprendizado de máquina onde todos os exemplos de treinamento são, necessariamente, rotulados como sendo de alguma classe. Nesse tipo de classificação convencional existem duas ou mais classes. Na classificação *one-class* (uma classe), o objetivo é definir uma classe de objetos e distinguir essa de quaisquer outros objetos que não fazem parte dela. Nesse tipo de classificação, após o treinamento, um novo padrão (exemplo), se semelhante aos objetos treinados, é classificado como normal (chamado de *target*) ou como novidade (ou *outlier*), caso não seja condizente com os exemplos de treinamento.

Na fase de treinamento de classificadores *one-class*, apenas exemplos *targets* (normais) são utilizados. O classificador *one-class* tem a função de aprender suas características ou, em outras palavras, fazer uma descrição do conjunto de objetos normais. Desse modo, resolve-se um grande problema comum que acontece em aplicações de aprendizado de máquina: nem sempre nós temos amostras de todas as classes disponíveis para o treinamento. Um exemplo que pode ser citado é o caso de detecção de intrusão, que é o tema deste trabalho: não conhecemos todos os tipos de ataques existentes. A cada dia, novos tipos de ataques são criados. Num outro exemplo, na área médica, ocorre que, em aplicações de diagnósticos, a população saudável é significativamente maior que a população anormal. Isso é um caso típico de classe não balanceada para treinamento, isto é, com várias amostras de um tipo e poucas de outro. Assim, conforme os exemplos citados, seria muito mais fácil treinar o classificador apenas com exemplos normais. Classificadores *one-class* se adequam a esse problema, ou seja, sua principal aplicação é a detecção de novidades.

Em aplicações de aprendizado de máquina para reconhecimento de padrões, é assumido que dois objetos próximos no espaço de características são semelhantes também na realidade. Isso nos leva a assumir que consideramos que os objetos estão espalhados no espaço de características e divididos em conjuntos onde nesses estão objetos semelhantes. Dessa forma, sabemos que o aprendizado se torna mais difícil quando temos poucos exemplos, pois, quando estamos fazendo classificações, devemos ter informações suficientes que nos levem a distinguir entre os objetos.

Na Figura 10, damos um exemplo representativo onde são mostrados dois classificadores: o convencional, representado pela linha contínua, e o *one-class*, com a linha pontilhada. O problema citado é a classificação de bailarinas e halterofilistas utilizando os atributos Altura (eixo x) e Peso (eixo y). Como visto, bailarinas e halterofilistas estão separados pelo classificador convencional através da linha reta, que define bem a fronteira entre as duas classes. Quando colocamos um terceiro elemento (lutador de sumô representado com um círculo verde), de classe desconhecida para o classificador desse problema, o elemento é classificado como halterofilista por estar à direita da linha contínua. Claramente vemos que, para o classificador convencional, existem apenas duas classes: ou bailarina, ou halterofilista, e qualquer novidade tem que ser um desses dois. No classificador *one-class* da figura, este distingue entre duas classes: “bailarinas ou halterofilistas”, que é a classe normal (ou *target)* ou uma outra classe que é a de novidades (ou *outliers)*. Verifica-se que é nessa última classe, a de novidades, que ficou representado o lutador de sumô.

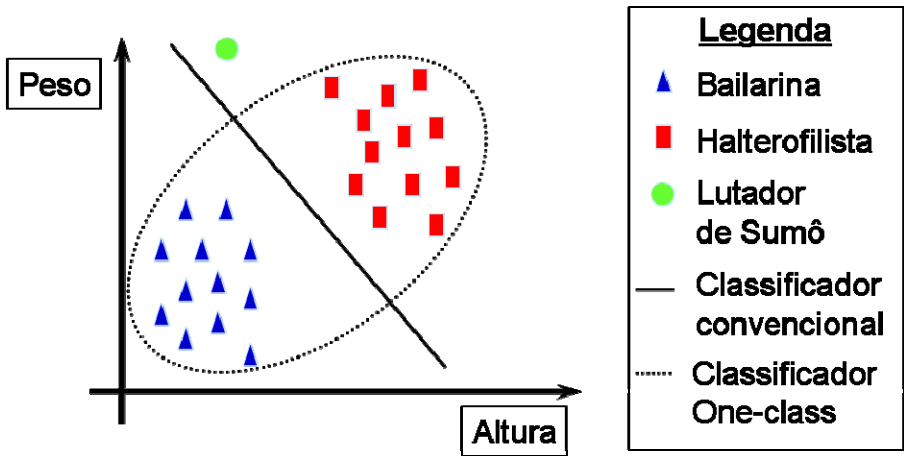

Figura 10. Exemplo de classificador *one-class* comparado com um classificador convencional para um conjunto de atributos bi-dimensionais.

No caso de classificador *one-class*, na maioria das vezes, o tipo de função *f* utilizado pelo classificador é escolhido antes do treinamento e somente alguns parâmetros têm que ser determinados. Exemplos dessas funções *f* são classificadores lineares, redes neurais artificiais, máquinas de vetor suporte (SVMs), etc. [32].

Nos classificadores *one-class*, também há problemas, assim como nos convencionais: ruídos[12], definição de erro e de outros parâmetros. No caso do *one-class*, temos um dificultador adicional: como calcular os limites entre normais e novidades se há apenas exemplos normais na fase de treinamento. Além disso, quê características devemos utilizar que distinguem melhor as classes. Essas são as dificuldades maiores desses classificadores.

Existem vários tipos de classificadores *one-class*. A maioria se encaixa em três métodos principais, baseados na forma de abordagem:

1. Avaliadores de densidade;
2. Métodos de reconstrução;
3. Métodos de criação de limites (também chamados de fronteiras ou perímetros).

Conforme veremos, utilizamos classificadores desses 3 tipos neste trabalho. Esses métodos de classificação se diferenciam na habilidade de lidar ou explorar as diferentes características dos dados [32]. São exemplos dessas características: grupamentos de objetos em clusters, convexidade da distribuição de dados, suas localizações nos subespaços definidos na classificação, etc.

3.2.1 Conceitos sobre classificação *one-class*

Os classificadores *one-class* têm dois elementos a serem definidos:

1. A medida da distância d(z) ou semelhança p(z) de um objeto de teste z em relação à normal.
2. Um gatilho θ que regula a distância ou semelhança aceitável (para estar na classe normal) de d(z) ou p(z).

Assim, temos que:

$$f(z) = I(d(z) < \theta) \qquad (3.5)$$

[12] Ruídos são dados do conjunto de treinamento que estão rotulados erroneamente.

para o caso de distância e,

$$f(z) = I(p(z) > \theta) \tag{3.6}$$

para a utilização de semelhança. Onde I(x) é uma função indicadora simples tal que:

$$I(x) = \begin{cases} 1 & \textit{se } x \textit{ é verdadeiro,} \\ 0 & \textit{caso contrário.} \end{cases} \tag{3.7}$$

A otimização dos parâmetros d(z) ou p(z) e θ é feita levando-se em consideração o compromisso entre a taxa TP (verdadeiros positivos) e a taxa FP (falsos positivos) (vide Figura 7 para conceitos de TP e TN). Porém, tem-se um problema para descobrir a taxa de FP (que envolve novidades) uma vez que no treinamento só se apresenta exemplos normais. Para isso, é necessário estimar uma densidade de novidades na fase de treinamento. Essas novidades são espalhados através do conjunto Z (Figura 11) que compreende todo o espaço de parâmetros, incluindo o conjunto de amostras normais (região em cinza claro na Figura 11) e o volume capturado pelo classificador (região circular em cinza escuro na Figura 11). Esse volume é justamente referente ao espaço definido onde as amostras que ali dentro estiverem serão consideradas normais e as demais serão novidades. Dessa forma, temos que amostras positivas que, após o treinamento, ficam fora desse volume, são consideradas FN (falsos negativos), pois deveriam ter sido consideradas positivas. Assim, podemos imaginar um espaço n-dimensional com um conjunto de exemplos normais em algum local e uma estimativa de volume que tenta englobá-los. Aumentando este volume pode fazer com que normais que tenham ficado fora passem a estar dentro do espaço volumétrico. Entretanto, é certo que outras amostras novidades que estavam fora desse espaço passarão a estar dentro também. Assim, verificamos uma conseqüente proporcionalidade entre as taxas TP e FP. Mais detalhes sobre o cálculo desse volume estão descritos em [32].

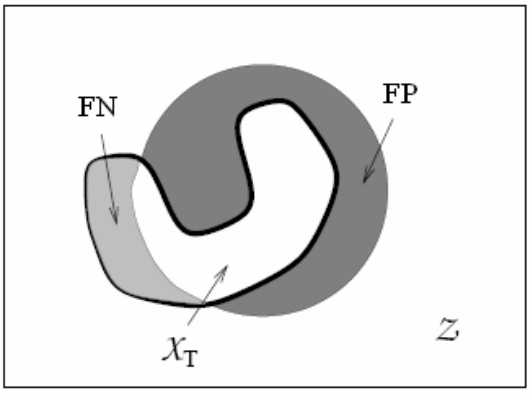

Figura 11. Regiões possíveis que compõem um problema bi-dimensional com classificador one-class

3.2.2 Métodos de classificação *one-class* utilizados

Os métodos de classificação *one-class* diferem entre si nas suas definições de p(z) ou d(z), na otimização desses parâmetros e nos limiares θ para o treinamento.

Os métodos listados a seguir possuem características diferentes baseadas no número de parâmetros livres, no número de parâmetros a serem inseridos pelo usuário,

suas flexibilidades para adaptar-se a mudanças e suas robustez a novidades no conjunto de treinamento.

Métodos de Densidade

Como o nome sugere, esses métodos são baseados na estimativa de densidade dos dados de treinamento. Com essa estimativa, o classificador estipula um limiar para a densidade. Neste projeto, consideramos o uso de um classificador baseado nesse método, o de densidade Parzen [24]. Esses métodos requerem vários exemplos normais para treinamento, para calcular a correta dimensionalidade dos dados e a complexidade do modelo de densidade. Dessa forma, o tamanho do conjunto de treinamento tem que ser relativamente grande. Caso contrário, esses métodos não devem ser considerados na escolha dos classificadores.

Com a otimização do limiar, nesse método, automaticamente é encontrado o mínimo volume de aceitação para o modelo de densidade. Na construção do modelo, o classificador ignora áreas com baixas densidades de dados por considerar que essas podem ser ruídos. Assim, apenas áreas com altas massas de exemplos são inclusas.

Estimativa de Densidade Parzen

Esse método, baseado em densidade, pode ser usado para estimar a densidade de probabilidade $\rho(x)$ de um vetor X de variáveis aleatórias contínuas. Isso envolve a sobreposição de uma função janela normalizada centrada em um conjunto de exemplos [23][24].

$$p(x) = \frac{1}{N}\sum_i p_N(x; x_i, hI) \tag{3.8}$$

Na equação 3.8, mostramos o método de classificação do classificador Parzen. *hI* é referente às matrizes de covariância diagonais utilizadas no treinamento. *xi* é referente ao i-ésimo exemplo de treinamento. O p_N é a distribuição de probabilidade para um objeto *x* com *d* dimensões e é dada por:

$$p_N(z; \mu, \sum) = \frac{1}{(2\pi)^{d/2} |\sum|^{1/2}} \exp\left\{-\frac{1}{2}(z-\mu)^T \sum^{-1} (z-\mu)\right\} \tag{3.9}$$

Onde μ é a média, $\sum$ é a matriz de covariância e *z* é o dado de entrada de teste.

O único parâmetro ajustável no treinamento desse classificador é o *h* referente à largura do núcleo *h* por pelo qual serão aceitos ou não os exemplos. Em [18], é apresentado um método para realizar o ajuste otimizado da largura *h*, o qual utilizamos neste trabalho.

Pelo fato do Parzen só precisar de um parâmetro regulável e inserido pelo usuário, ele é bastante fácil de utilizar na prática. Isso, inclusive, é uma vantagem da maioria dos classificadores *one-class* existentes: o usuário precisa ajustar poucos parâmetros.

Métodos de Fronteira

Esses métodos são muito bem aplicáveis a problemas com poucos exemplos para o treinamento. Nesses casos, somente um limite (ou fronteira) ao redor do conjunto de treinamento é criado e otimizado.

A maioria dos classificadores de fronteira utiliza a métrica da distância para definir os limites entre normais e novidades. Por causa disso, a arrumação dos dados no espaço (que é referente aos valores dos parâmetros) influencia nos casos com poucos exemplos.

Neste trabalho, foram considerados os classificadores NNd e kNNd pertencentes a essa técnica.

Método do vizinho mais próximo (*Nearest neighbor method description* - NNd)

O conceito do NND é relativamente simples: cada novo objeto é avaliado armazenando a sua distância euclidiana ao seu vizinho mais próximo do conjunto de treinamento. Essa distância é a NN(x). Ela é normalizada pela distância do seu vizinho mais próximo, que foi previamente calculada, a NN(NN(x)). Um objeto de teste z é considerado normal se a sua distância ao vizinho mais próximo for menor que a distância desse ao seu vizinho mais próximo.

Pode-se ver na equação 3.10 a forma de calcular, no classificador NNd, se um padrão de teste é normal ou não. Verifica-se que, se a distância do padrão de teste ao vizinho mais próximo (numerador) for menor que a distância do padrão mais próximo até o seu mais próximo, o valor da fração vai ser menor que 1, fazendo com que o f(z) seja igual a 1, ou seja, classificado como normal (ou *target*). Os valores entre || e || informam que devem ser calculados através de distância euclidiana.

$$f_{NN^{tr}}(z) = I\left(\frac{\left\|z - NN^{tr}(z)\right\|}{\left\|NN^{tr}(z) - NN^{tr}(NN^{tr}(z))\right\|} \leq 1\right) \tag{3.10}$$

Nesse método, não há nenhum parâmetro adicional de regulação que deva ser inserido pelo usuário.

kNNd

Esse método é semelhante ao NNd com uma única diferença: ao invés de, a cada novo exemplo, ser calculado sua distância para o objeto mais próximo e comparar com a distância desse para o mais próximo dele, nesse outro método calcula-se a distância dele para o(s) k-mais próximos e compara-se essa com a distância desses k-mais próximos até o(s) k-mais próximos deles.

Nota-se, na equação 3.11, a semelhança que há com o NNd (equação 3.10), mudando apenas pelo k.

$$f_{NN^{tr}}(z) = I\left(\frac{\left\|z - NN_k^{tr}(z)\right\|}{\left\|NN_k^{tr}(z) - NN_k^{tr}(NN_k^{tr}(z))\right\|} \leq 1\right) \tag{3.11}$$

Com relação aos parâmetros de entrada, aqui precisamos definir o *k* que é relativo à quantidade de exemplos que serão utilizados no cálculo da distância euclidiana.

Métodos de Reconstrução

Este método, como sugere o nome, tenta reconstruir o modelo que originou aqueles dados que estão sendo analisados para serem aprendidos. Utilizando os dados de treinamento, um modelo é escolhido para reconhecer os dados nos testes. Geralmente, esses métodos utilizam características pré-definidas para encontrar o modelo tais como características de clusters que são definidas no espaço de exemplos. Assim, protótipos são definidos no espaço e o erro de reconstrução deve ser minimizado. O classificador desse método que utilizamos foi o Kmeans [3] que será explanado a seguir.

Kmeans

Também chamado de Kmeans clustering, leva em consideração que os dados estão agrupados no espaço e podem ser caracterizados por alguns objetos pré-definidos, chamados de protótipos. Se o método já contempla, de antemão, o protótipo, resta a ele definir a(s) correta(s) localização(ões) deste(s) protótipo(s), que é feita minimizando o erro a seguir:

$$\mathcal{E}_{k-m} = \sum_i (\min \| x_i - \mu_k \|^2), \tag{3.12}$$

onde k é a quantidade de protótipos que devem ser inseridos no espaço de treinamento. O μ_k, por sua vez, é a localização do protótipo k que está sendo calculada naquele momento. O x_i é o i-ésimo padrão de treinamento que estiver sendo treinado. As barras verticais || indicam o cálculo da distância euclidiana.

O Kmeans utiliza a distância entre os protótipos e todos os exemplos para poder encontrar o melhor lugar para a localização do protótipo. Fazendo isso, torna o método robusto a novidades uma vez que distâncias de minorias (que seriam as novidades) não são tão influentes na decisão devido à quantidade.

Para facilitar o entendimento, na Figura 12 é mostrado um exemplo de classificador Kmeans. Para esse exemplo, o k utilizado teve o valor de 5. Podemos ver que existem 5 protótipos que se sobrepõem e geram um perímetro delimitando a classe de exemplos normais.

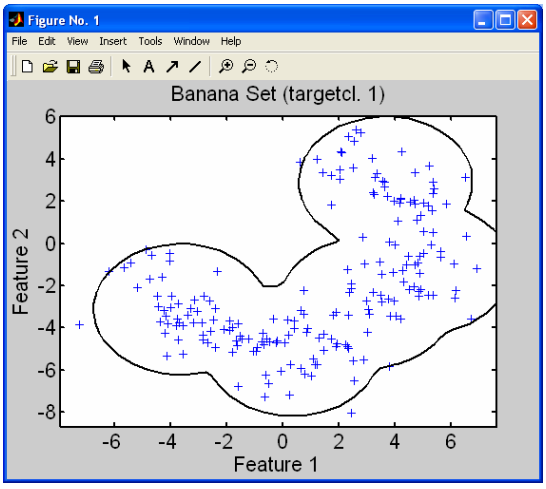

Figura 12. Exemplo de classificador kMeans num problema de classificação bi-dimensional.

Em linhas gerais, o funcionamento do Kmeans é da seguinte forma:

1 – Os k protótipos são espalhados aleatoriamente no espaço;

2 – Todos os objetos de treinamento x são atribuídos ao protótipo mais próximo através do cálculo da distância euclidiana;

3 - A localização do protótipo é atualizada, utilizando a média do conjunto de seus padrões de entrada x.

4 – Os passos acima são repetidos até que a localização do protótipo não seja mais alterada.

O cálculo da distância d de um objeto z para o conjunto de exemplos normais é definido da seguinte forma:

$$d_{k-m}(z) = \min \| z - \mu_k \|^2 \tag{3.13}$$

Onde z é um padrão de entrada de teste e μ_k é a localização do protótipo k. Temos, assim, que a distância d de um novo padrão de teste no classificador kMEANS é dada pela menor distância entre o padrão e os k protótipos, como vemos na equação 3.13. Se o d(z), após calculado, for menor que o d especificado como limiar, z é considerado normal. Caso contrário, é considerado novidade (*outlier*).

3.3 Classificadores *one-class* aplicados em detecção de intrusão em redes de computadores (IDS)

Já falamos de segurança em redes, de detecção de intrusão e de classificadores *one-class*. Entretanto, uma coisa interessante a falar é o que nos levou a aplicar tais classificadores no problema citado. Como já visto, não temos modelos definidos para técnicas de intrusão. Detectá-las então, torna-se uma tarefa difícil. Também já foi visto, e não é novidade, que as ameaças mudam constantemente por conseqüência da criação de novos tipos de ataques.

Em classificadores convencionais, quando aplicados à detecção de intrusão, a base de dados para treinamento deve ter diversos exemplos de vários tipos de ataques distintos além de conexões normais. Isso faz com que, dado um exemplo de teste, esse seja classificado como algum tipo de ataque ou como uma conexão normal. Isso acontece mesmo que o exemplo citado não faça parte de nenhum dos tipos de ataques nem tampouco seja uma conexão normal. Ou seja, é um ataque, porém não foi previsto na fase de treinamento o que leva o classificador a classificá-lo erroneamente em alguma classe!

O problema citado é muito comum. É muito difícil que alguém conheça todos os ataques disponíveis e ainda tenha exemplos desses em quantidade suficiente para treinar um classificador convencional e esse aprender os comportamentos de cada um deles. Além disso, fica inviável constantemente remodelarmos e re-treinarmos o classificador criando mais classes à medida que novos tipos de ataques forem sendo criados.

O problema da classificação errada para um exemplo pode trazer danos nos casos de sistemas IDS: alguns comportamentos inusitados podem acontecer como, por exemplo, algum novo tipo de ataque ser classificado como um já existente no sistema e uma ação[13] indesejada ser disparada. Ou, pior ainda, um novo tipo de ataque ser classificado como normal e isto não gerar nenhuma ação. Em [21], [22] e [27], são utilizados classificadores convencionais para a detecção

[13] Em sistemas IDS, configuramos ações a ser realizarem em casos de ocorrência de algum tipo de alarme. Por exemplo: envio de mensagem por e-mail ao administrador da rede, envio de mensagem ao *firewall* para bloqueio do tráfego detectado, etc.

de intrusão utilizando a mesma base de dados de detecção de intrusão utilizada nesse trabalho. Nesses trabalhos, há um desempenho muito bom nos experimentos. Entretanto, não é previsto que possa haver novidades em relação a ataques. Ou seja, não é prevista a ocorrência de novos tipos de ataques.

Na prática, não necessariamente precisamos saber qual o ataque que está sofrendo a rede. A simples detecção desse ataque e a conseqüente execução de alguma ação já são suficientes para o objetivo que se propõe o sistema, que é a detecção da intrusão, e não a classificação desta.

No caso de classificadores *one-class*, utilizamos os exemplos considerados normais como *targets* durante o treinamento. Qualquer exemplo que fugir da generalização criada pelo classificador será classificado como uma novidade (*outlier*), que é uma intrusão.

Capítulo 4

Experimentos e Resultados

Neste capítulo, explana-se a formatação da base de dados utilizada, as ferramentas, os experimentos, a metodologia e os resultados obtidos nos experimentos.

4.1 Base de Dados

Os dados utilizados para este trabalho foram os dados disponibilizados no banco de dados do repositório UCI [33] que foram utilizados na Terceira Competição Internacional de Descobrimento de Conhecimento e Ferramentas de Mineração de Dados (*Third International Knowledge Discovery and Data Mining Tools Competition*). Nessa competição, a tarefa foi construir um IDS com modelo preditivo (classificador) que fosse capaz de distinguir entre conexões normais e não normais (ataques ou intrusões). Essa base de dados contém um conjunto padrão de dados para serem auditados, que inclui uma grande variedade de intrusões simuladas num ambiente de rede militar [28]. A utilização dessa base em avaliação de desempenho para aplicações de aprendizado de máquina é comum. Exemplos disso são [21] e [22].

A base de dados original contém quase cinqüenta milhões de padrões, que são conexões de rede, distribuídas entre normais e diversos tipos de ataques. Existe uma versão reduzida da base do mesmo site da UCI, que possui apenas 10% dos dados da base original. Essa foi a versão que utilizamos neste trabalho. A divisão exata dos exemplos da base dados com suas classes é mostrada na Figura 13.

Verificamos, a partir da divisão original da base de dados, que a maioria das conexões é referente a ataques. Apenas aproximadamente 20% dos exemplos é que são relativos a conexões normais. Esse número elevado de ataques na base de dados deve-se ao fato de ser uma base gerada a partir de experiências com o fim de pesquisas sobre os ataques. Nesses experimentos, o sistema IDS foi "bombardeado" com vários tipos de ataques para gerar os dados necessários para a competição. Usualmente, entretanto, o número de ataques a uma rede de computadores não é tão alto quanto essa taxa.

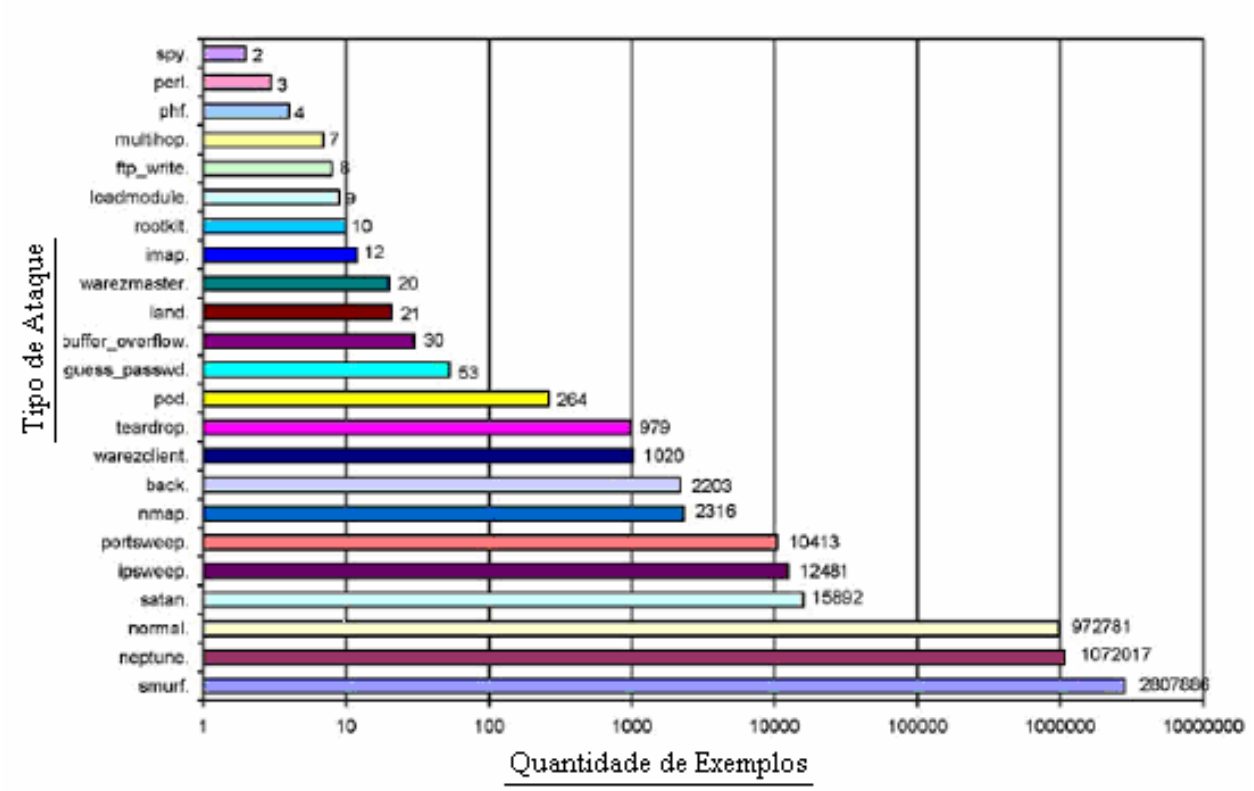

Figura 13. Distribuição dos dados da base de dados utilizada neste trabalho com 4.898.431 exemplos [21].

Para cada conexão (cada padrão), tem-se 41 atributos quantitativos e qualitativos. Uma conexão é uma seqüência de pacotes TCP começando e terminando em períodos definidos, onde os dados fluem com um endereço IP de origem e um de destino. Existem quatro categorias principais de ataques na base:

- **DOS** (Denial of Service - negação de serviço): é um ataque onde se tenta levar o sistema à exaustão através da alta utilização de processamento, memória ou largura de banda, fazendo com que os recursos fiquem ocupados e deixem de responder a usuários legítimos.
- **R2L** (Remote to local): Acesso não autorizado a uma máquina remota, sem uma conta válida, tentando ganhar acesso de usuário explorando alguma vulnerabilidade.
- **U2R** (User to root): Quando um usuário inicia utilizando uma conta normal na máquina e explora alguma vulnerabilidade para ganhar acesso de super-usuário.
- ***Probing***: Inspeções e sondagens não autorizadas a fim de descobrir alguma vulnerabilidade ou pegar alguma informação. Com um mapa das máquinas da rede, o atacante pode partir para tentar outros tipos de ataques.

Essas categorias são apenas para classificação teórica já que nos nossos experimentos utilizamos as classificações dos próprios ataques, assim como veremos na Tabela 2 que apresenta os grupos de ataques, os tipos que os compõem e uma descrição do ataque com os efeitos causados. Todos os ataques citados na tabela estão presentes na base de dados utilizada.

Tabela 2. Grupos, nomes e tipos de ataques no conjunto de testes.

Grupo	Nome do Ataque (ameaça)	Serviço	Mecanismo utilizado	Efeito do ataque
DOS	*back*	http	abuso/ vulnerabilidade	Diminui o tempo de resposta do servidor
DOS	*land*	http	vulnerabilidade	Trava a máquina
DOS	*neptune*	icmp	abuso	Diminui o tempo de resposta da máquina
DOS	*pod (Ping of Death)*	icmp	vulnerabilidade	Trava a máquina
DOS	*smurf*	icmp	abuso	Diminui o tempo de resposta da rede
DOS	*teardrop*	IP	vulnerabilidade	Trava a máquina
PROBE	*ipsweep*	icmp	abuso	Identifica máquinas ativas
PROBE	*nmap*	vários	abuso	Identifica portas abertas na máquina
PROBE	*portsweep*	vários	abuso	Identifica portas abertas na máquina
PROBE	*satan*	vários	abuso	Procura por vulnerabilidades conhecidas
R2L	*ftp_write*	ftp	configuração errada	Toma o acesso da máquina
R2L	*guess_passwd*	telnet, rlogin, pop, ftp, imap	abuso	Toma o acesso da máquina
R2L	*imap*	imap	vulnerabilidade	Toma acesso de root na máquina
R2L	*multihop*	tcp	vulnerabilidade	Toma o acesso da máquina
R2L	*phf*	http	vulnerabilidade	Executa comandos como usuário http
R2L	*spy*	ftp	configuração errada	Toma o acesso da máquina
R2L	*warezclient*	tcp	vulnerabilidade	Toma o acesso da máquina
R2L	*warezmaster*	tcp	vulnerabilidade	Toma o acesso da máquina
U2R	*buffer_overflow*	sessão do usuário	estouro de pilha	Toma o acesso de root da máquina
U2R	*loadmodule*	sessão do usuário	configuração errada	Toma o acesso de root da máquina
U2R	*perl*	sessão do usuário	configuração errada	Toma o acesso de root da máquina
U2R	*rootkit*	sessão do usuário	vulnerabilidade	Toma o acesso de root da máquina

Os atributos dos exemplos da base são as características das conexões detalhados na Tabela 3.

Tabela 3. Atributos (características) da base de dados utilizada neste trabalho.

Característica	Descrição	Tipo
duration	tempo de duração da conexão (em segundos)	contínuo
protocol_type	protocolo utilizado	discreto
service	serviço de rede	discreto
flag	status da conexão (normal ou erro)	discreto
Src_bytes	quantidade de bytes de dados da origem para o destino	contínuo
Dst_bytes	quantidade de bytes de dados do destino para a origem	contínuo
land	1 se a conexão é de ou para o mesmo host ou porta 0 caso contrário	discreto
wrong_fragment	quantidade de fragmentos errados	contínuo
urgent	quantidade de pacotes urgentes	contínuo
hot	quantidade de indicadores "hot"	contínuo
num_failed_logins	quantidade de tentativas com login falhando	contínuo
logged_in	1 se o login obtiver êxito 0 caso contrário	discreto
num_compromised	quantidade de condições "comprometidas"	contínuo
root_shell	1 se obteve acesso de root 0 caso contrário	contínuo
Su_attempted	1 se tentou tornar-se super-usuário 0 caso contrário	contínuo
num_root	quantidade de acessos como root	contínuo
num_file_creations	quantidade de operações de criação de arquivos	contínuo
num_shells	quantidade de prompts shell	contínuo
num_access_files	quantidade de operações de controle de acesso em arquivos	contínuo
num_outbound_cmds	quantidade limite de comandos em uma sessão ftp	contínuo
is_host_login	1 se o login pertencer à lista "hot" 0 caso contrário	discreto
is_guest_login	1 se o login é de um convidado 0 caso contrário	discreto
count	quantidade de conexões para um mesmo host como a conexão corrente nos últimos 2 segundo	contínuo
Srv_count	quantidade de conexões para um mesmo serviço como conexão correntes nos últimos 2 segundos	contínuo
serror_rate	porcentagem de conexões que possuem erros "SYN"	contínuo
Srv_serror_rate	porcentagem de conexões que possuem erros "SYN"	contínuo
rerror_rate	porcentagem de conexões que possuem erros "REJ"	contínuo
Srv_rerror_rate	porcentagem de conexões que possuem erros "REJ"	contínuo
same_srv_rate	porcentagem de conexões para o mesmo serviço	contínuo
diff_srv_rate	porcentagem de conexões para diferentes serviços	contínuo
Srv_diff_host_rate	porcentagem de conexões para diferentes serviços	contínuo

4.2 Ferramentas Utilizadas

Para o treinamento, testes e comparações foi utilizado o Matlab e os pacotes *PRTools* e *DDTools* para Matlab responsáveis por, respectivamente, fornecer funções para aprendizado de máquina e para classificadores *one-class*.

4.2.1 MATLAB 6.5.0

O nome Matlab[19] vem de "MATrix LABoratory". Matlab é um aplicativo destinado a, entre outras coisas, realizar operações matemáticas com matrizes. O programa provê um ambiente para aplicações em computação, onde podemos criar rotinas e funções que nos permitem criar pacotes semelhantes a aplicativos dentro da ferramenta Matlab. Os comandos do MATLAB são semelhantes à forma como escrevemos expressões algébricas, tornando simples o seu uso.

Na Figura 14, é dado um exemplo de inserção de uma matriz no programa Matlab. Uma das formas é a mostrada na figura.

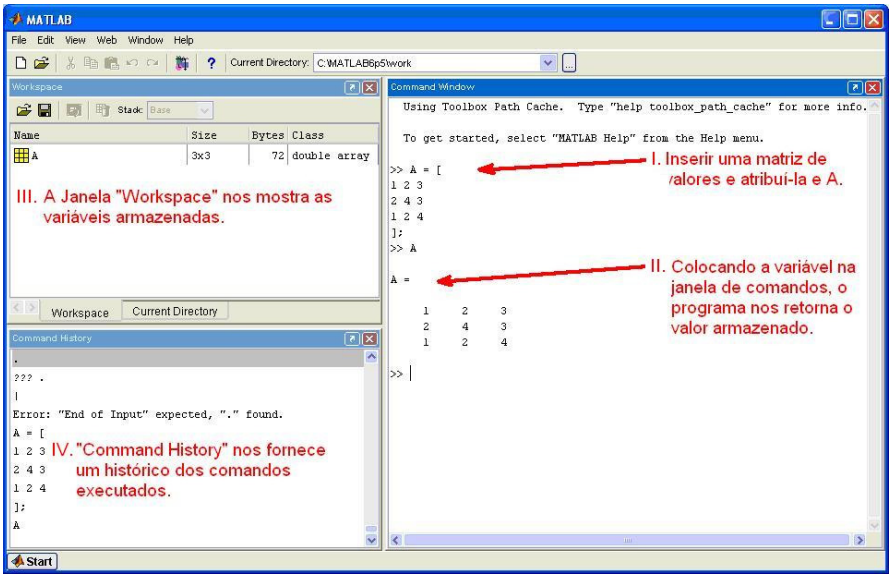

Figura 14. Exemplo de inserção de uma matriz no MATLAB

4.2.2 *PRTools* 4.0

O *PRTools* é uma ferramenta, baseada no Matlab, para aplicações de reconhecimento de padrões [9]. Sua utilização é livre para pesquisas acadêmicas. Suas funcionalidades incluem geração de bases de dados aleatórias para testes e treinamentos, além de várias funções de mapeamentos das bases de dados para aplicações de aprendizado de máquina, como, por exemplo, funções de treinamento de Redes Neurais Artificiais, classificadores lineares, etc. Para avaliação dos dados, o *PRTools* permite-nos encontrar o erro de teste, dentre várias outras medidas para verificação da eficiência de funções.

Na Figura 15, mostramos uma tela do *PRTools* sendo utilizado para gerar bases de dados aleatórias. É função do *PRTools* gerar gráficos com as bases de dados e com os classificadores utilizados. Na Figura, vemos como criar uma base de dados aleatória, dividi-la em outras duas, e desenharmos as amostras dessa base no gráfico do Matlab.

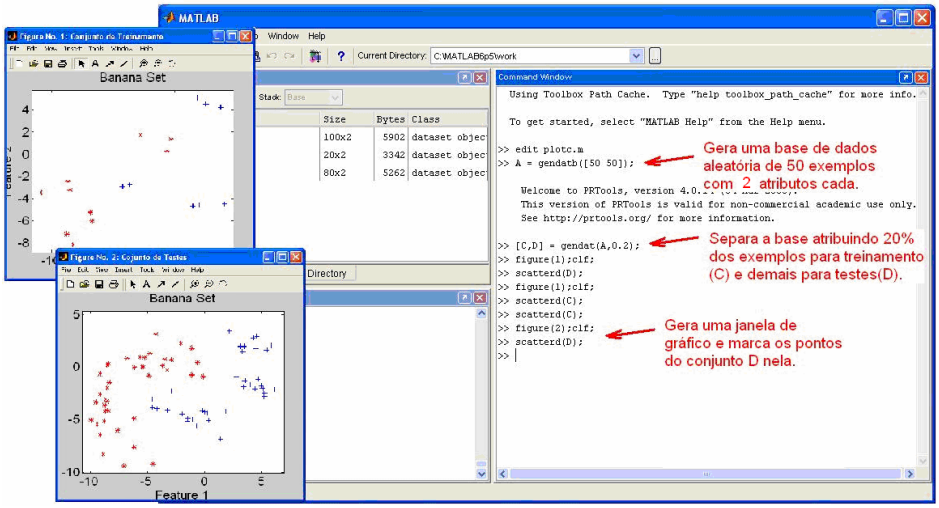

Figura 15. PRTools – gerando bases de dados aleatórias

4.2.3 *Data Description Toolbox* (*DD_Tools*) 1.4.1

Data Description (ou Descrição de Dados) é um outro nome dado à classificação *one-class*. O *DD_Tools* [31] é um pacote de funções que é executado em conjunto com o *PRTools*, no Matlab. O pacote provê ferramentas, classificadores e funções de avaliação para pesquisas e experimentos, no Matlab, utilizando classificadores *one-class*. Com o *DD_tools*, podemos criar bases de dados para serem utilizadas com classificadores *one-class*, treinar bases de dados com classificadores *one-class*, gerar novidades artificiais, para caso não os tenhamos, verificar os erros dos classificadores através de várias métricas presentes nele. E, por último, podemos estimar a Curva ROC e o erro AUC de vários classificadores. O pacote já vem com vários classificadores *one-class* implementados e, entre eles, estão os que foram utilizados neste projeto.

Na Figura 16, temos um exemplo da ferramenta *DDTools*. No caso, geramos duas bases de dados aleatórias, definimos os conjuntos normais e novidades e aplicamos a estes dois classificadores: NNd e kMEANS (note que existem 5 círculos que representam justamente o k=5 do Kmeans). Na Figura, temos um gráfico com os exemplos do conjunto de testes e uma demonstração dos limites de perímetros dos classificadores, sendo a linha contínua referente ao NNd e a linha pontilhada referente ao classificador kMEANS.

Na Figura 17, vemos um exemplo do cálculo de Curvas ROC do *DD_Tools*. Em sua plotagem, podemos, nesse caso, comparar visualmente as duas curvas (lembrando que quanto mais à esquerda e na parte superior, melhor). Além dessa comparação, temos também o AUC que calculamos (vide figura) para a estimativa de erro e classificação. Vemos que, para o conjunto de testes no exemplo, o classificador NNd obteve um AUC de 0,7313 enquanto que o kMEANS alcançou o AUC de 0,9932, sendo mais eficiente que o primeiro.

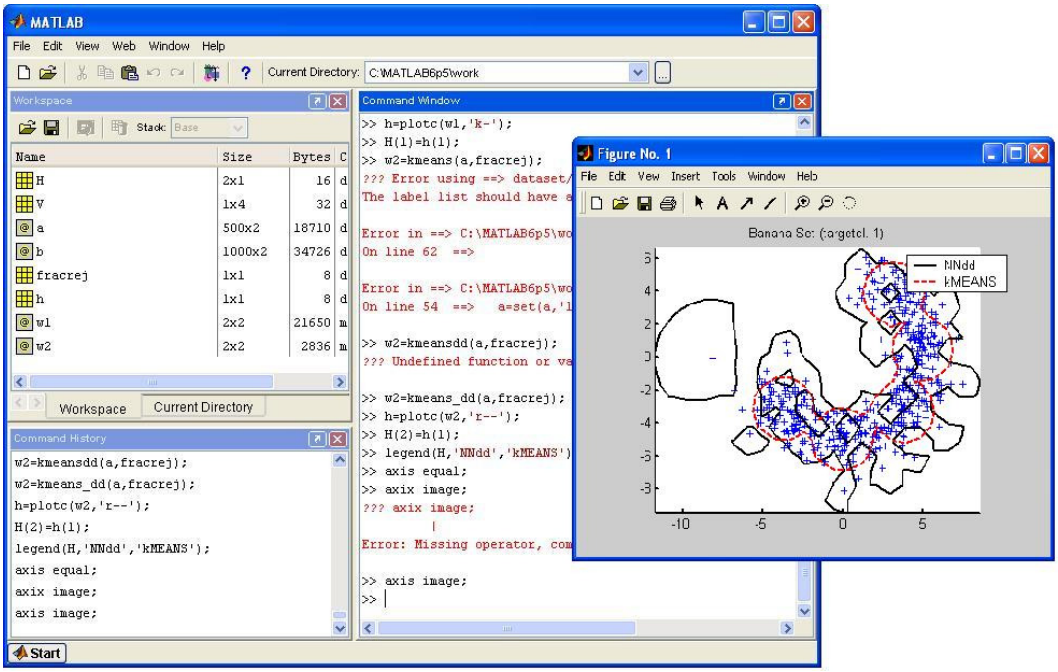

Figura 16. Exemplo *DDTools* – representando, no gráfico, dois classificadores *one-class, o NNd e o kMEANS.*

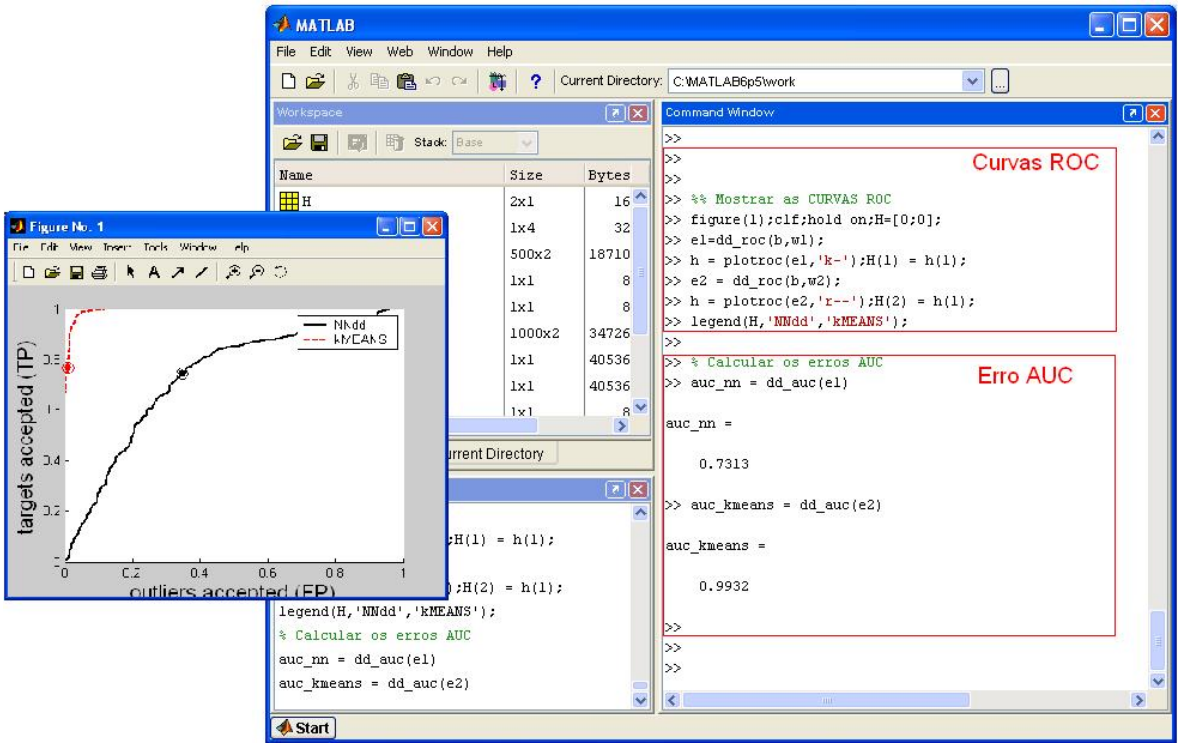

Figura 17. Exemplo do *DDTools* – cálculo e demonstração gráfica das curvas ROC e AUC.

4.3 Treinamento

Com o intuito de avaliar o desempenho de classificadores *one-class* para detecção de intrusão em redes de computadores, escolhemos quatro classificadores, que contemplam os três métodos citados para, assim, além de avaliar a viabilidade da utilização desse tipo de classificadores, fazermos uma comparação entre eles para apontar qual o melhor para essa aplicação. Como já dito anteriormente, utilizamos os classificadores Parzen, NNd, kNNd e kMEANS, que contemplam os três métodos descritos neste projeto: densidade, fronteira e reconstrução.

Para todos os experimentos, executamos treinamento, testes e verificação de desempenho (através da AUC). Foi definido que iriam-se treinar os classificadores com amostras de 500, 1000, 2.000 e 3.000 exemplos normais (lembrando que, em treinamento *one-class*, só se utiliza exemplos normais). O motivo da quantidade máxima de exemplos no conjunto de treinamento ser de 3.000 é referente a problemas de alta necessidade de hardware para a utilização desses classificadores no pacote *DDTools*. Um computador com 4 GB de memória RAM física não foi suficiente para bases com mais de 3.000 amostras, por exemplo. Dessa forma, por tentativas de obter a máxima quantidade de amostras de treinamento para ser possível a realização dos experimentos, chegou-se ao valor de 3.000.

A razão pela qual decidimos ter 4 conjuntos de treinamento (de 500 a 3.000 exemplos), inicialmente, foi para verificar a influência do número de amostras do conjunto de treinamento no desempenho. Em outras palavras, queríamos saber se existia uma diferença de desempenho em treinar a base com diferentes quantidades de amostras no treinamento, no resultado final. É importante verificarmos essa influência para saber, quando estiver utilizando o *one-class* na prática, quantas amostras normais são necessárias para que o treinamento forneça as classificações mais eficientes.

Retiramos os primeiros 3.000 exemplos classificados como normais, na base de dados original, para serem utilizados como conjunto de treinamento. Ainda dividimos esse conjunto para termos vários conjuntos de treinamento, como citado, que seriam utilizados nos experimentos. Na Tabela 4, vemos as bases de dados criadas inicialmente, para treinamento.

Tabela 4. Conjuntos de Treinamento para a Fase 1.

Nome do conjunto	Quantidade de exemplos	Tipo
Ids_treina_500	500	Normais
Ids_treina_1k	1.000	Normais
Ids_treina_2k	2.000	Normais
Ids_treina_3k	3.000	Normais

A inserção de dados para treinamento no Matlab é da forma de inserção de matriz. Por exemplo, o conjunto ids_treina_500.m tem a formatação apresentada na Figura 18. Foi preciso fazer um pré-processamento na base de dados para colocá-la em formato aceito pelo DDTools. Primeiro, todos os valores têm que ser numéricos. Dessa forma, pegamos os valores discretos nominais que existiam na base e fizemos um mapeamento nome-número, conforme mostrado no Apêndice A. Além disso, outros detalhes foram ajustados tais como, separação de atributos através de “espaços”, etc.

Na Figura 18, vemos um exemplo de inserção dos dados de treinamento no DDTools/Matlab. Verifica-se que não precisamos colocar, no final de cada linha, o tipo de conexão, se ataque ou normal, pois, por ser este um conjunto de treinamento, já é entendido que todos os exemplos correspondem a amostras normais.

```
ids_treina_500 = [
0 1 9 8 181 (...) 1.00 0.00 0.11 0.00 0.00 0.00 0.00 0.00
0 1 9 8 239 (...) 1.00 0.00 0.05 0.00 0.00 0.00 0.00 0.00
0 1 9 8 235 (...) 1.00 0.00 0.03 0.00 0.00 0.00 0.00 0.00
(...)
];
```

Figura 18. Exemplo de base de dados de entrada para o treinamento, Fase 1.

4.4 Testes

Na Figura 19, vemos um exemplo de inserção dos dados de um conjunto de testes no DDTools/Matlab. Diferentemente da entrada para o treinamento, precisamos colocar duas matrizes, uma para os atributos e uma outra referente às classes de cada exemplo. Após isso, juntamos essas duas matrizes no Matlab e atribuímos à classe normal a classificação de normais e às demais, novidades. Dessa forma, vemos que, mesmo na fase de testes, os classificadores *one-class* não precisam saber a classificação exata (i.e. o tipo do ataque) do exemplo. Basta saber se é normal ou novidade.

```
ids_teste_atibutos = [
0 1 9 5 181 (...) 1.00 0.00 0.15 0.00 0.00 0.00 0.00 0.00
0 1 5 8 239 (...) 1.00 0.00 0.05 0.00 0.00 0.00 0.00 0.00
0 2 9 2 235 (...) 1.00 0.00 0.08 0.00 0.00 0.00 0.00 0.00
(...)
];

ids_teste_outputs = [
5
1
7
(...)
];
```

Figura 19. Exemplo de base de dados de entrada para os testes, Fase 1.

Para os testes dos experimentos com os classificadores, utilizamos 3.000 exemplos, onde 1.500 foram novidades (*i.e.* ataques) citados na Tabela 2 e 1.500 exemplos foram conexões normais. O critério utilizado para a divisão dos ataques dentro desses 1.500 exemplos teve o objetivo de colocar todos os tipos de ataques da base de dados em quantidades razoáveis para cada um, de forma a evitar um conjunto de testes desbalanceado, isto é, com várias amostras de um tipo e poucas ou nenhuma de outro. Dessa forma, utilizamos o modelo da Tabela 5 para inserir ataques na base.

Tabela 5. Critério para inserção de exemplos do tipo ataque no conjunto de testes para a Fase 1 e Fase 2.

Quantidade de Exemplos na base original:		Quantidade Utilizada na base de teste	Observação
De	Até		
1	100	todas	
101	5.000	60	
5.001	20.000	100	
20.001	1.500.000	250	
1.500.001	Em diante	471	* para completar 1.500

Assim sendo, utilizamos um conjunto com exemplos de teste com exatamente a divisão mostrada na Tabela 6. O nome dado a esse conjunto foi o ids_teste_3k.m.

Tabela 6. Quantidade de exemplos utilizados no conjunto de testes **ids_teste_3k.m**

Nome do Ataque (ameaça)	Base Original	Base para testes
spy	2	2
perl	3	3
phf	4	4
multihop	7	7
ftp_write	8	8
loadmodule	9	9
rootkit	10	10
imap	12	12
warezmaster	20	20
land	21	21
buffer_overflow	30	30
guess_passwd	53	53
pod (Ping of Death)	264	60
teardrop	979	60
warezclient	1.020	60
back	2.203	60
nmap	2.316	60
portsweep	10.413	100
ipsweep	12.481	100
satan	15.892	100
neptune	1.072.017	250
smurf	2.807.886	471
Subtotal (ataques)	**3.925.650**	**1.500**
normal	972.781	1.500
Total	**4.898.431**	**3.000**

A metodologia utilizada para testes foi a seguinte: treinamos os 4 conjuntos de dados da Tabela 4 com os quatro classificadores escolhidos. Para todos esses treinamentos, executamos testes com o conjunto de teste fixo ids_teste_3k.m. Essa primeira fase chamamos de Fase 1. A métrica para os testes, conforme já dito, foi a AUC baseada em Curvas ROC. De posse dos resultados desses treinamentos e testes iniciais, obtemos as melhores curvas ROC e AUCs. Após isso, escolhemos o conjunto que forneceu os melhores resultados e executamos mais 10 treinamentos, com 10 novos conjuntos de treinamento escolhidos aleatoriamente, porém mantendo o conjunto de teste fixo. A essa fase, demos o nome de Fase 2. O objetivo dos experimentos da Fase 2 foi verificar se a escolha das amostras que compuseram o conjunto de treinamento tem influência significativa no resultado. Nesses experimentos, tiramos as médias e desvio padrão dos resultados obtidos para ajudar-nos da conclusão.

4.4.1 Testes utilizando o classificador NND

O classificador NND é uma técnica não-parametrizada, isto é, que não precisamos inserir parâmetros livres para ajustar o treinamento. Uma vez que os classificadores NND têm, como treinamento, a responsabilidade de armazenar as localizações de todo o conjunto de treinamento, seu treinamento se torna um pouco longo, dependendo da quantidade de exemplos. Devido a isso, no nosso caso, o NNd foi o método que levou mais tempo para ser treinado. Esse tempo de treinamento variou de 4 segundos, para as 500 amostras, até 630 segundos quando treinado com 3.000 amostras.

Além disso, o método NND demonstrou apresentar alto consumo de memória física da máquina utilizada, sendo essa a razão pela qual restringimos o treinamento a 3.000 amostras, no máximo.

Na Tabela 7, temos os resultados dos treinamentos para os quatro conjuntos iniciais de testes utilizando o classificador NNd. Vemos que o AUC obtido para o teste com o conjunto de 2.000 amostras se apresenta como a melhor opção. Apesar disso, o de 3.000 amostras apresenta um valor muito próximo, o que pode levar-nos a concluir que isso é uma diferença relativa a exemplos específicos dos conjuntos, assumindo que a diferença é pouca de forma que pode ser considerada irrelevante. O AUC citado é de 0,8375 e foi obtido para o conjunto de testes de 3.000 exemplos quando o conjunto de treinamento contemplava 2.000 amostras.

Tabela 7. Resultados obtidos na Fase 1 para o NNd

FASE 1 - NNd	
	NNd
Número de Exemplos	AUC
500	0,7166
1000	0,7638
2000	**0,8375**
3000	0,8324
Média	**0,7876**
Desvio Padrão	**0,0580**

Na Figura 20, estão mostradas as curvas ROC para os quatro experimentos demonstrados anteriormente. Tanto nesses experimentos quanto nos demais, plotamos as curvas ROC no mesmo gráfico com o intuito de comparar graficamente essas entre si. Podemos ver na Figura que as curvas ROC para 2.000 e 3.000 amostras são muito semelhantes, com a curva vermelha

pontilhada (2.000 exemplos) sendo superior em desempenho que as demais. Com relação à proporcionalidade da Curva com o erro AUC, visivelmente verificamos que a ordem das curvas coincide com os AUC obtidos nos experimentos, em ordem decrescente, da mais eficiente para a menos eficiente.

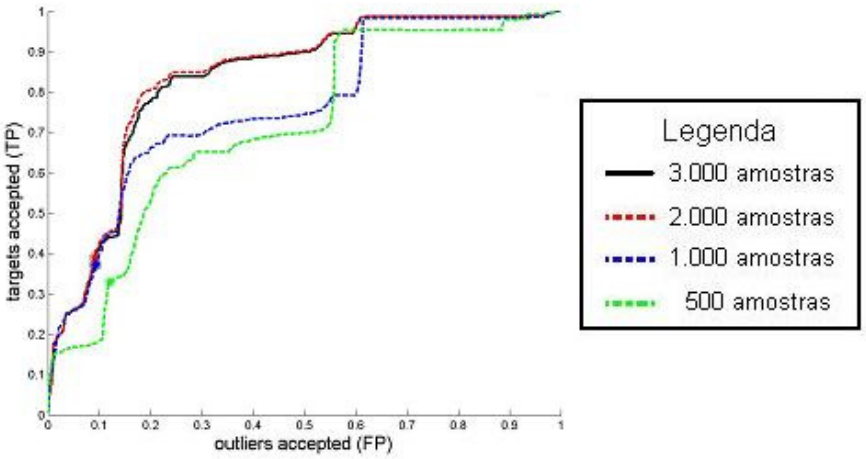

Figura 20. Curva ROC para o classificador NNd, Fase 1.

4.4.2 Testes utilizando o classificador kNND

No kNNd, temos apenas um parâmetro para regular, que é o k. Enquanto que no NNd este k é igual a 1, nesse é ajustável.

Para o valor de k nos experimentos com esse classificador, variamos para k = 2 e k = 5. Obviamente, não colocamos valor de k = 1, pois caso acontecesse, estaríamos igualando este experimento ao NNd, onde k é igual a 1 e não é ajustável.

Verificamos, conforme Tabela 8, que para ambos os valores k, a maior quantidade de amostras (3.000) obteve o melhor desempenho. Tivemos em k=5 os melhores resultados para esse classificador, obtendo uma AUC de 0,7105 quando testado com o conjunto de treinamento maior.

Tabela 8. Resultados obtidos na Fase 1 para o kNNd

FASE 1 - kNNd		
	k = 5	k = 2
Número de Exemplos	AUC	AUC
500	0,6135	0,567
1000	0,7033	0,6777
2000	0,7073	0,6890
3000	**0,7105**	**0,6932**
Média	**0,6837**	**0,6567**
Desvio Padrão	**0,0469**	**0,0602**

Na Figura 21, apresentamos um comparativo gráfico entre as quatro curvas ROC para o melhor valor de k encontrado, onde k = 5. Pode-se ver que temos a melhor curva como sendo a para 3.000 amostras, assim como é o AUC. O fato das curvas de 2.000 e 3.000 amostras serem quase sobrepostas, apenas diferenciando para pequenos valores de TP, justifica-se porque as 2.000 amostras também estão inseridas nesse conjunto de 3.000 amostras. Essa pequena diferença entre as duas curvas é relativa, justamente, às demais 1.000 amostras que contemplam o maior conjunto.

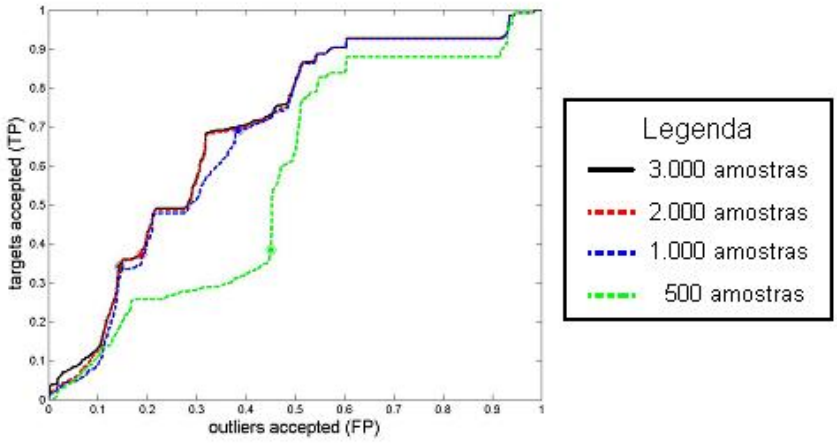

Figura 21. Curva ROC para o classificador kNNd, Fase 1, k=5.

4.4.3 Testes utilizando o classificador kMEANS

Neste método, temos um parâmetro a ser ajustado, que é a quantidade k de protótipos que queremos que sejam calculados no conjunto de treinamento.

Na Tabela 9, apresentamos o desempenho obtido para esse método. O melhor resultado foi encontrado quando k = 1, que foi de 0,8007 para um conjunto de treinamento de 1.000 exemplos. Um fato importante que podemos notar aqui é que, nos três casos, tivemos um desvio padrão relativamente pequeno, o que indica que a quantidade de amostras, no conjunto de treinamento, não influencia tanto como no NNd e kNNd, para esse problema, no desempenho final. Isto significa que, mesmo que tivéssemos uma quantidade bem maior de exemplos de treinamento, a diferença no desempenho provavelmente não seria visível.

Tabela 9. Resultados obtidos na Fase 1 para o kMEANS

FASE 1 – kMEANS			
	k = 5	k = 2	k = 1
Número de Exemplos	AUC	AUC	AUC
500	0,7219	0,7789	0,7922
1000	0,7326	0,7824	**0,8007**
2000	0,7297	0,785	0,7975
3000	**0,7676**	**0,7911**	0,7955
Média	0,7380	0,7844	**0,7965**
Desvio Padrão	0,0203	0,0051	**0,0036**

O baixo valor do desvio padrão citado se torna evidente quando verificamos, através da Figura 22, que as curvas ROC são muito semelhantes, se sobrepondo em quase todos os pontos do gráfico.

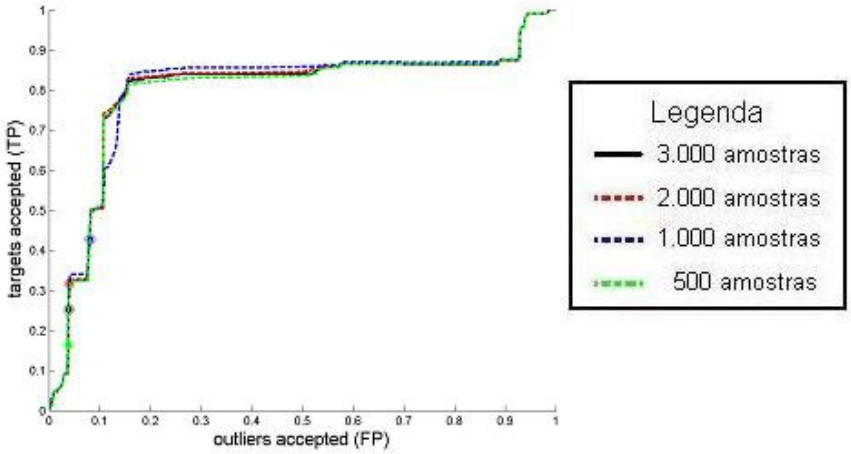

Figura 22. Curva ROC para o classificador kMEANS, Fase 1, k=1

4.4.4 Testes utilizando o classificador PARZEN

Para o método PARZEN, temos um parâmetro a ajustar. Esse é o parâmetro de largura *h* do classificador. A ferramenta DD_Tools nos fornece uma facilidade de otimizar essa medida para obter o melhor desempenho, que foi citada na seção 3.2.2 deste trabalho.

Na Tabela 10, vemos que o melhor AUC foi obtido para o conjunto com apenas 500 exemplos. De fato, todos os experimentos dessa fase apresentaram um baixo valor do AUC, comparado com os outros classificadores.

Tabela 10. Resultados obtidos na Fase 1 para o PARZEN

FASE 1 - PARZEN	
	PARZEN
Número de Exemplos	AUC
500	**0,6966**
1000	0,6508
2000	0,6864
3000	0,6917
Média	**0,6814**
Desvio Padrão	**0,0208**

Esse classificador, embora seja muito rápido para treinar (menos de um segundo em todas as fases), leva um tempo maior na fase de testes, pois todos os objetos do treinamento devem ser armazenados e suas distâncias são calculadas para todos os objetos de treinamento. No nosso caso, esse teste levou em torno de 20 segundos. Em algumas aplicações, entretanto, esse tempo é inviável, podendo limitar a aplicação.

Para visualização gráfica e comparação, disponibilizamos as curvas ROC para a Fase 1 com o classificador PARZEN na Figura 23. Um fato interessante que podemos verificar analisando esta curva é que, apesar do experimento com conjunto de treinamento de 500 amostras

ter obtido o melhor AUC, ele não é superior aos demais para todos os valores da taxa TP. Podemos ver que para a taxa FP entre 0,05 e 0,3 o classificador para os 500 exemplos é melhor que os demais. Entretanto, isso não é válido para os demais pontos, onde os com mais exemplos são melhores que o com menos.

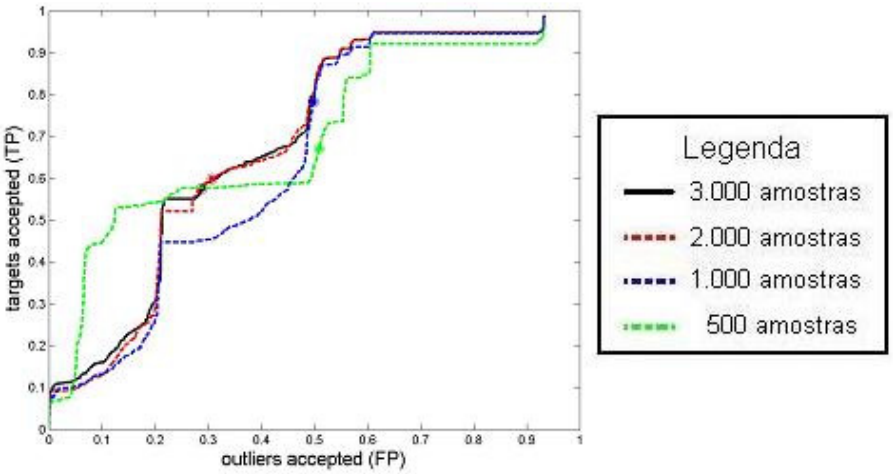

Figura 23. Curva ROC para o classificador PARZEN, Fase 1

4.4.5 Fase 2 – Influência da escolha dos padrões de treinamento

Como dito anteriormente, na Fase 2 utilizamos dez conjuntos de treinamento com quantidade igual à dos conjuntos que forneceram os melhores resultados na Fase 1. O objetivo desses experimentos foi de verificar a influência das amostras de treinamento no desempenho dos classificadores.

Baseando-se nos resultados obtidos na Fase 1, verificamos que, em média, o conjunto com 3.000 amostras obteve os melhores resultados. Dessa forma, extraímos da base de dados original mais dez conjuntos de treinamento com 3.000 amostras normais, cada, para treinarmos novamente. Esses conjuntos são disjuntos entre si, ou seja, cada um deles apresenta amostras diferentes que não aparecem nos demais.

Para cada classificador, apresentamos os melhores valores obtidos, além da média aritmética dos valores e os desvios padrões. Os resultados obtidos na Fase 2 estão dispostos na Tabela 11. Verificamos, de acordo com a Tabela, que o melhor AUC ocorreu no 4º conjunto de treinamento da 2ª Fase, para o classificador NNd. Apesar disso, a melhor média aritmética entre os experimentos de um mesmo classificador foi obtida pelo classificador kMEANS. Nesse caso, consideramos que a média dos experimentos é mais importante que o maior valor isolado do AUC e isso nos leva a concluir que o kMEANS pode ser considerado melhor para o problema citado. Notamos também que o método PARZEN, embora tenha obtido resultados inferiores aos demais, obteve o menor desvio padrão indicando uma regularidade nos AUC dos experimentos. Isso indica que a utilização de diferentes bases de treinamento não tem grande impacto no resultado de sua classificação nesse problema.

Nas Figura 24, Figura 25, Figura 26 e Figura 27, apresentamos as curvas ROC do melhor conjunto de treinamento (maior AUC) para cada um dos 4 classificadores. Na Figura 25, temos a curva ROC para o melhor AUC obtido, o resultado dos experimentos da Fase 2 utilizando o quarto conjunto de treinamento para o método NNd. Verificamos que a linha está bem localizada à esquerda e para cima, indicando um bom desempenho do classificador. Nas demais figuras,

notamos que os classificadores tiveram uma boa curva ROC também, com todas elas acima da diagonal y=x e próximas do ponto (0,1) que é o ponto de melhor eficiência para os classificadores.

Tabela 11. Resultados obtidos na Fase 2 para todos os classificadores

FASE 2 - Só conjuntos de 3.000 exemplos				
Conjunto	NNDD	PARZEN	KMEANS, K = 1	KNNDD, K = 5
	AUC	AUC	AUC	AUC
1	0,8324	0,6917	0,7955	0,7302
2	0,7993	0,7196	0,7971	0,7399
3	0,8289	0,6552	0,8042	0,7438
4	**0,8468**	0,6826	0,798	0,755
5	0,7118	0,6972	0,8028	0,7032
6	0,5781	0,6891	**0,8078**	0,6953
7	0,8049	0,7057	0,8035	0,7888
8	0,7858	0,672	0,8063	**0,8254**
9	0,6012	0,6895	0,791	0,6911
10	0,6475	**0,7443**	0,6734	0,7748
Média	0,7437	0,6947	**0,7880**	0,7448
Desvio Padrão	0,1013	0,0247	**0,0406**	0,0431

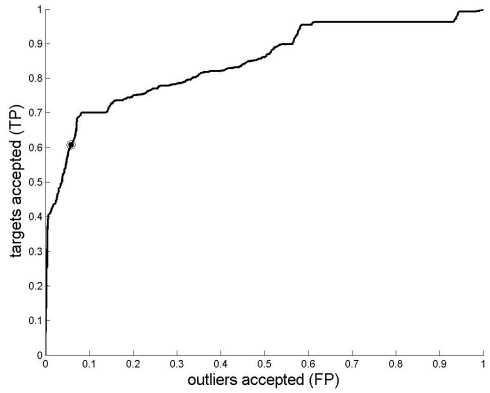

Figura 24. Curva ROC para a Fase 2, NNd, 4º conjunto de treinamento

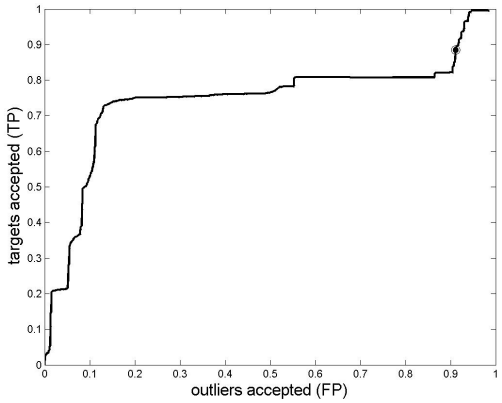

Figura 25. Curva ROC para a Fase 2, PARZEN, 10º conjunto de treinamento

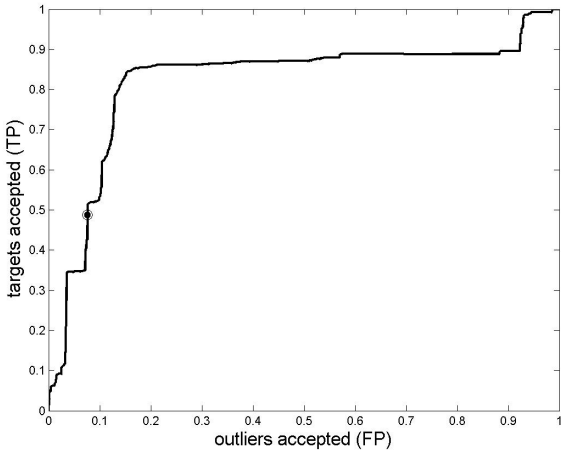

Figura 26. Curva ROC para a Fase 2, kMEANS, 6º conjunto de treinamento

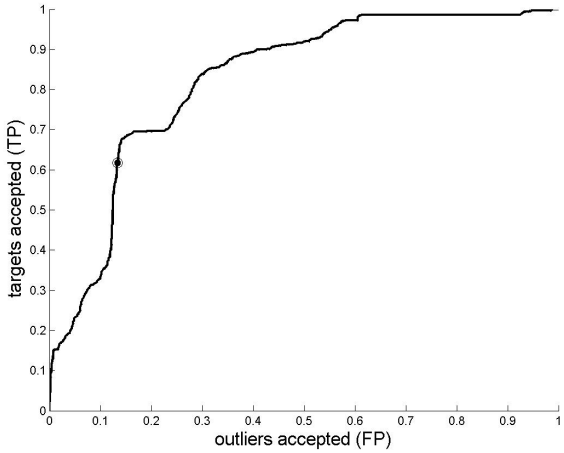

Figura 27. Curva ROC para a Fase 2, kNNd, 8º conjunto de treinamento

Os resultados obtidos foram considerados bons, pois tiveram medidas AUC altas, quando verificamos nas tabelas Tabela 7, Tabela 8, Tabela 9, Tabela 10 e Tabela 11, conforme já mostrado anteriormente. Além disso, as curvas ROC apresentadas, para todos os classificadores, apresentaram uma boa disposição dos pontos no espaço de possibilidades entre classificações positivas e falsas. Nessas curvas, verifica-se que passam, na maioria das vezes, em pontos superiores à diagonal x = y, que representa um classificador aleatório, sendo, portanto, bons classificadores.

Um exemplo de implementação do algoritmo *one-class* kMeans, que foi realizada durante este projeto, está disponível no Apêndice B.

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho apresentou um estudo sobre classificadores *one-class* aplicando-os ao problema de detecção de intrusão em redes de computadores. Foram utilizados quatro classificadores – NNd, kNNd, Parzen e kMEANS - que são contemplados em três métodos distintos de classificação *one-class*: método de fronteira, método de densidade e método de reconstrução. Além disso, explanamos conceitos sobre métodos de avaliação de desempenho baseados em curvas ROC, que é uma metodologia recente no campo de aplicação do aprendizado de máquina, apesar de já ser, a muito tempo, utilizada em outras áreas como a médica, para diagnósticos. É provável que, cada vez mais, os pesquisadores da área de aprendizado de máquina passem a utilizar a métrica de desempenho da área sob a curva ROC (*area under curve* - AUC) ao invés de utilizar o cálculo simples da taxa de erro de classificação causada pela aplicação, pois, mostrou-se que AUC é uma medida melhor para avaliação. Para avaliar o melhor classificador *one-class*, dentre os testados, fizemos um estudo comparativo entre os quatro classificadores *one-class* aplicados ao problema de detecção de intrusão em redes de computadores.

5.1 Contribuições

- Reiteramos, através deste trabalho, que técnicas de aprendizado de máquina aplicadas à detecção de intrusão são opções muito boas, face ao bom desempenho de classificação obtido. Outros trabalhos já haviam deduzido essa eficiência tais como [21], [22] e [27].
- Não encontramos, no entanto, nenhum trabalho que tenha utilizado os classificadores *one-class* utilizados nesse trabalho para essa aplicação, de forma que este apresenta-se como uma novidade.
- Mostramos que o uso desses classificadores é uma opção muito boa para a aplicação em detecção de intrusão em redes pois esse é um problema que basicamente precisa distinguir entre a classe normal e as demais, não necessitando, obrigatoriamente, classificar o tipo de ataque.
- A partir deste projeto, é possível fazer outros tipos de experimentos comparando com os resultados aqui obtidos.

5.2 Discussão

O melhor AUC encontrado neste trabalho, embora seja bom (0,8468), poderia ser ainda melhorado caso o IDS pudesse levar em conta, além da detecção de novidades baseado em aprendizado de máquina, a detecção de ataques baseados em assinaturas previamente encontradas e formuladas. Além disso, a composição da base de dados de testes de trabalho, com 50% das amostras normais e 50% anormais, é, na prática, uma coisa que não acontece, pois a relação entre tráfego normal e ataques não chega a tanto. Dessa forma, na prática provavelmente o desempenho deste sistema seria ainda maior que os valores encontrados neste trabalho.

5.3 Trabalhos Futuros

Nos trabalhos [21], [22] e [27] utilizam-se classificadores convencionais para a obtenção de desempenho utilizando-se a mesma base de dados utilizada neste trabalho. Nessas referências, desempenhos muito bons foram obtidos. Em alguns casos, chegaram a quase 100% de acerto. No entanto, o treinamento e os testes foram realizados utilizando-se as mesmas classes de ataques (mas não os mesmo exemplos), além da classe normal. Ou seja, não é previsto, nesses trabalhos, a ocorrência de novas classes (novos tipos de ataques) além das treinadas. Uma sugestão de trabalho futuro poderia ser a comparação entre esses dois tipos de classificadores, *one-class* e multiclasses, utilizando a mesma métrica de desempenho utilizada nesse trabalho (AUC), porém aplicando novas classes de ataques aos classificadores convencionais, que não tenham sido previamente treinadas. Uma vez que a visualização de desempenho através de curvas ROC só pode ser realizada para problemas com duas classes, deve-se reduzir o problema multiclasses para problemas com duas classes para viabilizar a comparação. Com essa pesquisa, será possível verificar qual método, se *one-class* ou classificadores convencionais, é melhor aplicável para a identificação de novas classes.

Um outro trabalho futuro que pode ser proposto é a combinação de classificadores *one-class* com classificadores convencionais para a detecção de intrusão em redes de computadores. Como vemos na Figura 28, o IDS seria composto de dois classificadores, um *one-class* e outro convencional. O *one-class* seria responsável por detectar se o tráfego é normal ou intrusão (exatamente o que foi proposto nessa monografia). O classificador convencional serviria unicamente para classificar os ataques atribuindo-os às suas classes de ataques corretas para posteriores estatísticas ou ações. Dessa forma, o papel desempenhado por esse último classificador não seria tão fundamental na detecção de intrusão de forma que a sua eventual desatualização (não conhecer algum tipo de ataque) não resultaria em danos expressivos para a organização.

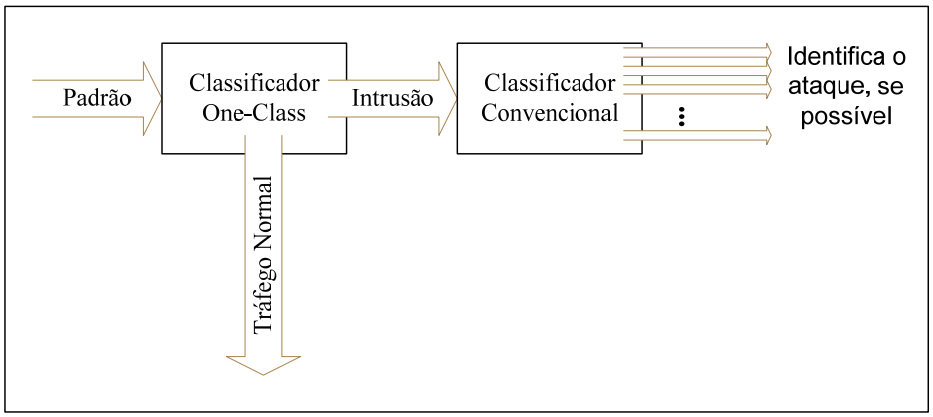

Figura 28. Trabalho futuro: Combinação de classificador *one-class* com convencional para a detecção e classificação de intrusão

Bibliografia

[1] AGÊNCIA BRASIL, Brasil está na rota dos crimes na internet, 2004. Artigo disponível em http://old.idgnow.com.br/AdPortalv5/SegurancaInterna.aspxGUID=5C172E89-0A38-4929-A764-1B0B5AC5AB3E&ChannelID=21080105. Acessado pela última vez em maio de 2006.

[2] BARBOSA, A. S. e MORAIS, L. F. M., Sistemas de Detecção de Intrusão, Seminários Ravel, 2000. CPS760, UFRJ.

[3] BISHOP, C., *Neural Networks for Pattern Recognition*, 1995. Oxford University Press.

[4] BRADLEY, A.P., *The use of area under the ROC curve in the evalution of machine leaning algorithms*. Pattern Recognition, volume 30, páginas 1145-1159, 1997.

[5] CERT. *Incidents Reported.* Documento disponível em http://www.cert.org/stats/#incidents. Acessado pela última vez em abril de 2006.

[6] CISCO*, Cisco Incident Control System - ICS*. Documentação técnica do produto disponível em http://www.cisco.com/en/US/products/ps6542/index.html. Acessado pela última vez em abril de 2006.

[7] CISCO, *Cisco Security Agent*. Documentação técnica do produto disponível em http://www.cisco.com/en/US/products/sw/secursw/ps5057/index.html. Acessado pela última vez em abril de 2006.

[8] CISCO, *Cisco Security Monitoring, Analysis and Response System 4.2*. documentação técnica do produto disponível em http://www.cisco.com/en/US/products/ps6241/products_data_sheet0900aecd80272e64.html. Acessada pela última vez em abril de 2006.

[9] DUIN, R., JUSZCZAK, P., PACLIK, P., PEKALSKA, E., de RIDDER, D., TAX, D., *PRTools4, A Matlab Toolbox for Pattern Recognition*, Delft University of Technology, 2004. Disponível em http://www.prtools.org. Acessado pela última vez em abril de 2006.

[10] EGAN, J.P., *Signal detection thory and ROC analysys*. Series in Cognition and Perception, Academic Press, 1975.

[11] ELKAN, C., *Naive Bayesian Learning*, Department of Computer Science, Harvard University, 1997. Disponível em http://citeseer.ist.psu.edu/30545.html. Acessado pela última vez em maio de 2006.

[12] FAWCETT, T., *An Introduction to ROC Analysis*. Institute for the Study of Learning and Expertise, 2005. Pattern Recognition Letters, Volume 27, páginas 861-874, 2006. Disponível em http://www.sciencedirect.com. Acessado pela última vez em abril de 2006.

[13] FLACH, P., WU, S., *Repairing concavities in ROC curves*. Em: Proc. 2003 UK Workshop on Computational Intelligence. University of Bristol, páginas 38-44, 2003.

[14] FUKANAGA, K., *Statistical Pattern Recognition*, 1990. Academic Press.

[15] HANLEY, J.A., McNEIL, B.J., *The meaning and use of the area under a receiver operating characteristic (ROC) curve*. Radiology volume 143, páginas 29-36, 1982.

[16] HEADY, R., *The Architecture of a Network Level Intrusion Detection System,* Technical Report CS90-20, Department of Computer Science, University of New Mexico, 1990.

[17] HUANG, J., LING, C.X., *Using AUC and Accuracy in Evaluating Learning Algorithms*, IEEE Transactions on Knowledge and Data Engineering, volume 17, No. 3, páginas 299-310, 2005. Disponível em http://www.ieeexplore.ieee.org. Acessado pela última vez em abril de 2006.

[18] KRAAIJVELD, M., DUIN, R., *A criterion for the smoothing parameter for parzen-estimators of probability density functions*., 1991. Relatórios técnicos, Delft University of Technology.

[19] MATHWORKS, The., Página WEB do MATLAB. Disponível em http://www.mathworks.com. Acessado pela última vez em maio de 2006.

[20] MÓDULO SECURITY, 9ª Pesquisa Nacional de Segurança da Informação, 2003. Documento disponível em http://www.modulo.com.br. Acessado pela última vez em abril de 2006.

[21] MUKKAMALA, S., SUNG, A., ABRAHAM, A., *Intrusion detection using na ensemple of intelligent paradigms*. Journal of Network and Computer Applications volume 28, páginas 167-182, 2004. Disponível em http://www.sciencedirect.com. Acessado pela última vez em maio de 2006.

[22] MUKKAMALA,S., SUNG, A., GUADALUPE, J., *Intrusion Detection Using Neural Networks and Support Vector Machines*, Department of Computer Science, New Mexico Institute of Mining and Technology, 2002.

[23] NANNI, L., *Experimental comparison o one-class classifiers for online signature verification*., Neurocomputing volume 69, páginas 869-873, 2006. Disponível em http://www.sciencedirect.com. Acessado pela última vez em maio de 2006.

[24] PARZEN, E. *On Estimation of a probability density function and mode*. Annals of Mathematical Statistics. Volume 33, páginas 1065-1076, 1962

[25] PROVOST, F., FAWCETT, T., *Analysis and Visualization of classifier performance: Comparison under imprecise class and cost distributions*. Third International Conference on Knowledege Discovery and Data Mining (KDD-97), AAAI Press, Menlo Park, CA, páginas 43-48, 1997.

[26] PROVOST, F., FAWCETT, T., KOHAVI, R., *The case against accuracy estimation for comparing induction algorithms*. In: Shavlik, J. (Ed.), Proc. ICML-98. Morgan Kaufmann, San Francisco, CA, páginas 445-453., 1998. Disponível em http://www.purl.org/NET/tfawcett/papers/ICML98-final.ps.gz.

[27] ROCHA, T., Estudo Comparativo entre Técnicas de Aprendizagem de Máquina para Sistemas de Detecção de Intrusão (IDS), 2005. Trabalho de conclusão de curso de Engenharia da Computação do Departamento de Sistemas Computacionais (DSC) da UPE.

[28] STOLFO, S. J.; WEI, F.; LEE, W.; PRODOMIDIS, A.; CHAN, P.K.: KDD CUP – *knowledge discovery and data mining competition,* 1999. Disponível em http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Acessado pela última vez em abril de 2006.

[29] SWETS, J.A., DAWES, R.M., MONAHAN,J., *Better Decisions Through Science*. Scientific American 283, páginas 82-87, 2000.

[30] SWETS, J.A., *Measuring the accuracy of diagnostics systems*. Science volume 240, páginas 1285-1293, 1988.

[31] TAX, D., *DDtools, the Description Toolbox for Matlab*, vesão 1.4.1, 2005. Ferramenta disponível em http://www-ict.ewi.tudelft.nl/~davidt/dd_tools.html. Acessado pela última vez em maio de 2006.

[32] TAX, D., *One-class classification*., Tese de doutorado, 2001. Disponível em http://www-ict.ewi.tudelft.nl/~davidt/oneclass.html. Acessado pela última vez em maio de 2006.

[33] UCI KDD ARCHIVE, Disponível em http://kdd.ics.uci.edu/datasets. Acessado pela última vez em abril de 2006.

Apêndice A

Pré-processamento – mapeamento nome-número de atributos discretos nominais

Apresentaremos aqui o mapeamento referente ao pré-processamento que foi realizado na base de dados original, tanto conjuntos de treinamento quanto testes, para viabilizar a utilização dos dados no Matlab. As tabelas abaixo apresentam, cada uma, os conjuntos de atributos discretos nominais contidos na base de dados utilizada neste projeto.

Tipo de protocolo	
NOME	CÓDIGO Utilizado
icmp	0
tcp	1
udp	2

Flag	
NOME	CÓDIGO Utilizado
REJ	0
RSTO	1
RSTOS0	2
RSTR	3
S0	4
S1	5
S2	6
S3	7
SF	8
OTH	9
SH	10

Serviço	
NOME	CÓDIGO Utilizado
auth	0
domain	1
domain_u	2
eco_i	3
ecr_i	4
finger	5
ftp	6
ftp_data	7
gopher	8
http	9
link	10
mtp	11
name	12
ntp_u	13
other	14
pop_3	15
private	16
remote_job	17
rje	18
smtp	19
ssh	20
telnet	21
time	22
whois	23
ctf	24
daytime	25
hostnames	26
http_443	27
imap4	28
IRC	29
login	30
nnsp	31
pm_dump	32
shell	33
sunrpc	34
systat	35
uucp	36

Classificação	
NOME	CÓDIGO Utilizado
buffer_overflow.	0
guess_passwd.	1
ipsweep.	2
loadmodule.	3
neptune.	4
normal.	5
perl.	6
pod.	7
portsweep.	8
smurf.	9
teardrop.	10
back.	11
ftp_write.	12
imap.	13
land.	14
multihop.	15
nmap.	16
phf.	17
rootkit.	18
satan.	19
spy.	20
warezclient.	21
warezmaster.	22

Apêndice B

Implementação do algoritmo kMeans

Apresentamos aqui a implementação, em linguagem Java 1.5.0, do algoritmo kMeans. A escolha desse algoritmo para implementação deveu-se ao fato desse ter sido o que obteve os melhores resultados para o problema apresentado. O código abaixo está comentado para melhor entendimento dos passos.

Classe KMeans

```
package kmeans;

/*
 * Projeto TCC - Daniel Gomes
 * "Deteccao de intrusao em redes de computadores
 * utilizando Classificadores One-Class".
 *
 * Daniel dos Anjos de O. Gomes
 * Departamento de Sistemas Computacionais - DSC/UPE
 *
 *
 */

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Implementacao do algoritmo kMeans
```

```
 * @author          Daniel
 */
public class kMeans {

        /** k eh sempre o numero de clusters */
        private int k;
        /** Array de clusters */
        public cluster[] clusters;
        /** Quantidade de Iteracoes */
        private int nIteracoes;
        /** Vetor de todos os pontos com seus respectivos clusters */
        private ArrayList pontosKMeansList;
        /** Nome do arquivo de entrada */
        private String arquivoEntrada;
        /** String contendo dados a serem gravados no arquivo */
        private static String strFile = "0\t0\n";
        /** Variável fracao ROC */
        private double fracROC;
        /** Array contendo todos os limiares de todos os clusters para posterior ordenacao */
        private double[][] arrayLimiares;


        /**
         * Retorna uma nova instancia do algoritmo kMeans
         *
         * @param       k                   numero de clusters
         * @param       arquivoEntrada      nome do arquivo contendo os dados de treinamento.
         */
        public kMeans(int k, String arquivoEntrada) {

                this.k = k;
                this.arquivoEntrada = arquivoEntrada;
                this.clusters = new cluster[this.k];
                this.nIteracoes = 0;
                this.pontosKMeansList = new ArrayList();

        }


        /**
         * Retorna uma nova instancia do algoritmo kMeans
         *
         * @param       k                   numero de clusters
         * @param       pontosKMeans        lista contendo objetos do tipo kMeansPoint
         */
        public kMeans(int k, List pontosKMeans) {

                this.k = k;
                this.arquivoEntrada = arquivoEntrada;
                this.clusters = new cluster[this.k];
                this.nIteracoes = 0;
                this.pontosKMeansList=new ArrayList(pontosKMeans);

        }
```

```
/**
 * Le os dados de entrada, a partir do arquivo, e armazena os pontos no vetor.
 */
public void readData() throws IOException{
	String line = null;
	InputStreamReader inputStream   = null;
	BufferedReader    bufReader   = null;
	// Cria um padrao onde diz que digitos ou um ponto fazem parte de um token.
	Pattern patPattern = Pattern.compile("[\\d.]+");

	try {
	  inputStream = new InputStreamReader( new FileInputStream(new File(this.arquivoEntrada)));
	  bufReader    = new BufferedReader(inputStream);

		while(true) {
			ArrayList<Float> pattern = new ArrayList<Float>();
			line = bufReader.readLine();

			// Le os padroes ate a ultima linha do arquivo.
			if(line == null)
				break;

			// procura os padroes (que sao os atributos) de cada linha.
			Matcher patternMatcher = patPattern.matcher(line);
			while(patternMatcher.find()){
				// Para cada padrao encontrado (que eh um token), adiciona
				// aa variavel pattern
				pattern.add(Float.parseFloat(patternMatcher.group()));
			}
			// Pega o valor de pattern e atribui a um ponto no vetor.
			PontoKMeans dp = new PontoKMeans(pattern);
			dp.assignToCluster(0);
			this.pontosKMeansList.add(dp);
		}

		bufReader.close();
		inputStream.close();
	} catch (FileNotFoundException e) {
		e.printStackTrace();
	}catch (IOException e){
		e.printStackTrace();
	}
}
```

```
/**
 * Executa o kMeans nos dados de entrada.
 */
public void runKMeans(double _fracROC) throws IOException{

        fracROC = _fracROC;

        // Seleciona k pontos como centros para os clusters iniciais.
        for (int i=0; i < k; i++){
          // cria os k clusters
                this.clusters[i] = new cluster(i);
                // seta os pontos iniciais dos k clusters,
                // atribuindo aleatoriamente pontos de exemplos ja existentes
                this.clusters[i].setCentro((PontoKMeans)(this.pontosKMeansList.get(
                        (int)(Math.random()         * this.pontosKMeansList.size())));
                // imprime as posicoes dos clusters iniciais
                //System.out.println("cluster inicial " + i + ": " + this.clusters[i].toString() );
        }

        do {
                // chamadas para o treinamento.
                Iterator i = this.pontosKMeansList.iterator();
                // hasNext retorna 1 se a iteracao tem mais elementos
                while (i.hasNext())
                        this.assignToCluster((PontoKMeans)(i.next()));

                this.nIteracoes++;

        }
        // Repetir ate que as medias parem de mudar.
        while (this.updateCentro());

/*
 * computa os limiares para cada pontoKmeans.
 */
        Iterator i = this.pontosKMeansList.iterator();
        cluster clusterAtual = null;
        PontoKMeans dp = null;
        double limiar = 0;
        int it = 0;
        arrayLimiares = new double[clusters.length][this.pontosKMeansList.size()];
        // Calculo do limiar
        while (i.hasNext()) {
                dp = (PontoKMeans)(i.next());
                int numClusterAtual = dp.getNumeroCluster();

                clusterAtual = clusters[numClusterAtual];

                limiar = dp.distancia(clusterAtual.getCentro(),dp);

                arrayLimiares[numClusterAtual][it] = limiar;

           it++;
        }
        /*
         * Ordena os limiares em ordem crescente
         */
        for(int j = 0;j < clusters.length;j++){
                int ct1 = 0;
```

```
                int ct2 = 0;
                double[] ar = arrayLimiares[j];
                double temp = 0;

                while(ct1 < ar.length-1){
                        ct2 = ct1 + 1;
                        while(ct2 < ar.length){
                                if(ar[ct1] > ar[ct2]){
                                        temp = ar[ct1];
                                        ar[ct1] = ar[ct2];
                                        ar[ct2] = temp;

                                }
                                ct2++;
                        }
                        ct1++;
                }

        }

        /*
         * Elimina o ruído no treinamento a taxa de 5%, que posso variar.
         */
        for(int j = 0;j < clusters.length;j++){

                int indiceLimiar = (int) (arrayLimiares[j].length * 0.95);

                /*
                 * Seta um limiar de acordo com a fracao ROC para geraçao da curva ROC
                 */
                clusters[j].setLimiarCluster(arrayLimiares[j][indiceLimiar]*fracROC);

        }

        /*
         * Executa o teste do modelo gerado
         */
        testKMeans t = new testKMeans(new File("ids_teste_3k.txt"),clusters);
        this.strFile += t.getStrFile();
}

/**
 * Atribui um ponto de dados para um dos k clusters baseado na sua distancia
 * do centro dos clusters (a menor distancia)
 *
 * @param       dp      ponto a ser atribuido
 */
private void assignToCluster(PontoKMeans dp) {
        // variavel clusterAtual é o cluster que esta sendo trabalhado no momento
        int clusterAtual = dp.getNumeroCluster();
        // distancia do ponto em questao ate o centro respectivo
        double distanciaMinima = PontoKMeans.distancia(dp, this.clusters[clusterAtual].getCentro());;

        // Para todos os clusters...
        for (int i=0; i < this.k; i++){
                // Se a distancia para o cluster k, que esta sendo calculada no momento,
                // for menor que a minima de outro cluster, atribui a nova distancia como
                // minima e o cluster como clusterAtual.
                if (PontoKMeans.distancia(dp, this.clusters[i].getCentro() ) < distanciaMinima ) {
```

```
                        distanciaMinima = PontoKMeans.distancia(dp, this.clusters[i].getCentro());
                        clusterAtual = i;
                        }
                }
                // atribui o ponto ao cluster mais proximo, encontrado nos calculos acima.
                dp.assignToCluster(clusterAtual);
                // imprime cada ponto e a que cluster ele pertence.
//              System.out.println("KMeansPoint: " + dp.toString());

}


/**
 * Atualiza as medias de todos os clusters e retorna se ele mudou
 *
 * @return          booleano - informa se as medias dos clusters mudaram ou nao.
 */
private boolean updateCentro() {
                // Cria um array chamado dimensions com tamanho = k.
                ArrayList[] dimensions = new ArrayList[this.k];
                boolean reply = false;
                float[] x = new float[this.k];
                int[] y = new int[this.k];
                int[] size = new int[this.k];
                PontoKMeans[] pastMeans = new PontoKMeans[this.k];


                for (int i=0; i<this.k; i++) {
                  ArrayList<Float> dimSum = new ArrayList<Float>(clusters[0].getCentro().getDim().size());
                  ////// qtd de colunas (atributos)
                  for (int j = 0 ; j < 41; j++) {
                                dimSum.add(j,Float.parseFloat("0"));
                  }
                  dimensions[i] = dimSum;
                  x[i] = 0;
                  y[i] = 0;
                  size[i] = 0;
                  pastMeans[i] = this.clusters[i].getCentro();
                }

                Iterator i = this.pontosKMeansList.iterator();
                // Calcula a soma de cada coluna para cada cluster, pois vamos calcular a media
                // mais na frente.
                while (i.hasNext()) {
                        PontoKMeans dp = (PontoKMeans)(i.next());
                        int clusterAtual = dp.getNumeroCluster();
                        size[clusterAtual]++;
                        for(int j = 0; j < dimensions[clusterAtual].size(); j++){
                                float sum = (Float) dp.getDim().get(j) + (Float) dimensions[clusterAtual].get(j);
                                dimensions[clusterAtual].set(j,sum);
                        }

                }
                // Calcula as medias em cada coluna (que eh a soma feita anteriormente)
                for (int j=0; j < this.k; j++ )
                        if(size[j] != 0) {
                                for(int z = 0; z < dimensions[j].size(); z++){
                                        dimensions[j].set(z,(Float) dimensions[j].get(z)/size[j]);
                                }
```

```
                    PontoKMeans temp = new PontoKMeans(dimensions[j]);
                    temp.assignToCluster(j);
                    this.clusters[j].setCentro(temp);
                    double distancia = PontoKMeans.distancia(pastMeans[j], this.clusters[j].getCentro());
                    // Quando a distancia for zero, eh pq nao esta mudando mais, ou seja,
                    // acabou o treinamento.
                    if (distancia !=0 )
                    reply = true;
                }

        // Se a distancia for igual a zero, vai retornar FALSE
        return reply;

}


/**
 * Retorna o valor de k
 *
 * @return          o valor de k
 */
public int getK() {

        return this.k;

}


/**
 * Retorna o cluster identificado pelo indice (index)
 *
 * @param           index     indice do cluster a ser retornado
 * @return          retorna o cluster identificado pelo indice
 */
public cluster getCluster(int index) {

        return this.clusters[index];

}


/**
 * Retorna os pontos para serem impressos
 *
 * @return  a string de saida dos pontos
 */
public String toString(){
        String cluster = "";
        for(int i = 0 ; i < k; i++ ){
                cluster = cluster + clusters[i].toString() + "\n";
        }
        return this.pontosKMeansList.toString() + "\n" + cluster;

}


/**
 * Retorna os pontos de dados
 *
```

```
 * @return  os pontos de dados
 */
public ArrayList getDataPoints() {

	return this.pontosKMeansList ;

}


/**
 * Retorna a String contendo os dados para gravacao no arquivo
 *
 * @return strFile
 */
public static String getStrFile() {
	return strFile;
}


/**
 * Main
 *
 * @param   args    argumentos
 */
public static void main(String[] args) throws IOException{

	/** Número de Clusters */
	int k = 5;

	/** Número de pontos para geracao da curva ROC */
	int nPts = 25;

	/** Limite inferior do parametro fracao ROC */
	double limInferior = 0.8;

	/** Limite superior do parametro fracao ROC */
	double limSuperior = 1.2;

	/** Intervalo de incremento do parametro fracao ROC */
	double intervalo = (limSuperior-limInferior)/nPts;

	/** Variavel de incremento do parametro fracao ROC */
	double incremento = limInferior;

	kMeans km = new kMeans(k, "ids_treina_500.txt");

	try {
		km.readData();
	} catch (Exception e) {
		e.printStackTrace();
		}

	/*
	 * iteracao para variacao do parametro fracao ROC
	 */
	while(incremento <= limSuperior){
		km.runKMeans(incremento);
		incremento += intervalo;
	}
```

```
		FileWriter fw = new FileWriter(new File("saida.txt"));

		strFile+= "1\t1";
		fw.write(getStrFile());
		fw.flush();


	}

}
```

Classe testKMeans

```
package kmeans;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class testKMeans {

	/** Vetor de todos os pontos com seus respectivos clusters */
	private ArrayList pontosKMeansTeste;

	/** Vetor de todos os pontos com seus respectivos clusters */
	private ArrayList pontosKMeansTesteResultantes;

	/** Array de clusters */
	public cluster[] clusters;

	/** String contendo os valores para concatenacao e geracao do arquivo de testes */
	private String strFile;


	public testKMeans(File _file, cluster[] _cluster) throws IOException{
		clusters = _cluster;
		pontosKMeansTeste = new ArrayList();
		pontosKMeansTesteResultantes = new ArrayList();
		readData(_file);
		testaKMeans();

	}

	/**
	 * Le os dados de entrada, a partir do arquivo, e armazena os pontos no vetor.
	 */
	public void readData(File arquivoEntrada) throws IOException{
		String line = null;
		InputStreamReader inputStream   = null;
```

```
		BufferedReader	bufReader	= null;
		boolean novelty;
		// Cria um padrao onde diz que digitos ou um ponto fazem parte de um token.
		Pattern patPattern = Pattern.compile("[\\d.]+");

		try {
			inputStream = new InputStreamReader( new FileInputStream(arquivoEntrada));
			bufReader	= new BufferedReader(inputStream);


			while(true) {
				ArrayList<Float> pattern = new ArrayList<Float>();
				line = bufReader.readLine();
				novelty = false;

				// Le os padroes ate a ultima linha do arquivo.
				if(line == null)
					break;

				// procura os padroes (que sao os atributos) de cada linha.
				Matcher patternMatcher = patPattern.matcher(line);
				while(patternMatcher.find()){
					// Para cada padrao encontrado (que eh um token),
					// adiciona aa variavel pattern
					pattern.add(Float.parseFloat(patternMatcher.group()));
				}
				int ir = 0;

				// Seta o padrao de entrada de teste como novidade, caso o seu ultimo
				// parametro seja igual a 1.
				if(((Float) pattern.get(pattern.size()-1)).equals(new Float(1))){
					novelty = true;
				}


				pattern.remove(pattern.size()-1);


				// Pega o valor de pattern e atribui a um ponto no vetor.
				PontoKMeans dp = new PontoKMeans(pattern);
				dp.assignToCluster(0);
				dp.setNovelty(novelty);
				this.pontosKMeansTeste.add(dp);
			}

			bufReader.close();
			inputStream.close();
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}catch (IOException ioe){
			ioe.printStackTrace();
		}
	}

	/**
	 * Método responsável pelo teste dos pontosKMeans em relaçao a todos clusters
	 */
	public void testaKMeans(){
		Iterator itPtKMeansTeste = pontosKMeansTeste.iterator();
```

```
		PontoKMeans ptTeste = null;
		int posVerdadeiros = 0;
		int falsosPositivos = 0;
		int nPontosNormais = 0;
		int nPontosNovidades = 0;
		double taxaTP = 0;
		double taxaFP = 0;
		while(itPtKMeansTeste.hasNext()){
			ptTeste = (PontoKMeans) itPtKMeansTeste.next();

			// Seta o padrao como normal, caso a sua distancia ao cluster seja menor que o limiar.
			//
			for(int i = 0;i < clusters.length;i++){
			  if(ptTeste.distancia(clusters[i].getCentro(),ptTeste) < clusters[i].getLimiarCluster() ){
				ptTeste.setClassificacaoNovelty(false);
				}
			}

			pontosKMeansTesteResultantes.add(ptTeste);
		}

		// Calculo de Taxa FP e Taxa TP.
		//
		Iterator it = pontosKMeansTesteResultantes.iterator();
		while(it.hasNext()){
			ptTeste = (PontoKMeans)it.next();

			if(!ptTeste.isNovelty()){
				nPontosNormais++;
			}else{
				nPontosNovidades++;
			}

			if(!ptTeste.isNovelty() && (ptTeste.isClassificacaoNovelty() == ptTeste.isNovelty()))
				posVerdadeiros++;

			if(ptTeste.isNovelty() && ptTeste.isClassificacaoNovelty() != ptTeste.isNovelty())
				falsosPositivos++;

		}
		taxaFP = (double) falsosPositivos/nPontosNovidades;
		taxaTP = (double) posVerdadeiros/nPontosNormais;

		System.out.println(nPontosNovidades+" "+nPontosNormais);
		System.out.println("pos verdadeiros -->> "+posVerdadeiros+" taxaTP => "+taxaTP);
		System.out.println("falsos positivos -->> "+falsosPositivos+" taxaFP => "+taxaFP);

		System.out.println("*******************************************************");
		strFile = taxaFP+ "\t"+taxaTP+"\n";

}

/**
 * String contendo os valores para concatenacao e geracao do arquivo de testes
 * @return strFile
 */
public String getStrFile() {
		return strFile;
}
```

```
}
```

Classe PontoKMeans

```
package kmeans;

import java.util.ArrayList;

/**
 * PontoKMeans.java
 *
 * @author Daniel
 */
public class PontoKMeans {

	/** Valor na dimensao x */
	private ArrayList dim;

	/** Cluster atribuido */
	private int numeroCluster;

	/** Classificacao Obtida para a amostra */
	private boolean classificacaoNovelty = true;

	/** Classificacao desejada */
	private boolean isNovelty = true;

	/**
	 * Cria uma nova instancia de ponto de dados
	 *
	 * @param _x
	 *            valor na dimensao x
	 * @param _y
	 *            valor na dimensao y
	 */
	public PontoKMeans(ArrayList _x) {

		this.dim = _x;
		this.numeroCluster = 0;
	}

	/**
	 * Atribui um ponto de dado a um cluster
	 *
	 * @param _numeroCluster
	 *            o cluster que esse ponto de dados deve ser atribuido
	 */
	public void assignToCluster(int _numeroCluster) {

		this.numeroCluster = _numeroCluster;

	}

	/**
	 * Retorna o cluster que o ponto pertence
	 *
```

```
 * @return o numero do cluster que o ponto pertence
 */
public int getNumeroCluster() {

	return this.numeroCluster;

}

/**
 * Retorna o valor do ponto na dimensao X
 *
 * @return o valor na dimensao x
 */
public ArrayList getDim() {

	return this.dim;

}

/**
 * Retorna o valor que indica se o ponto é ou nao novidade.
 *
 * @return valor que indica se o ponto é ou nao novidade
 */
public boolean isNovelty() {
	return isNovelty;
}

/**
 * Seta o valor que indica se o ponto é ou nao novidade.
 *
 * @param isNovelty
 */
public void setNovelty(boolean isNovelty) {
	this.isNovelty = isNovelty;
}

public float getAtributteDim(int pos) {
	return Float.parseFloat(this.dim.get(pos).toString());
}

/**
 * Retorna a distancia Euclidiana entre dois pontos
 *
 * @param dp1
 *          o primeiro ponto (origem)
 * @param dp2
 *          o segundo ponto (destino)
 * @return a distancia entre dois pontos de dados
 */
public static double distancia(PontoKMeans dp1, PontoKMeans dp2) {

	double resultado = 0;
	for (int j = 0; j < dp1.getDim().size(); j++) {
		resultado += ((dp1.getAtributteDim(j) - dp2.getAtributteDim(j)) *
			(dp1.getAtributteDim(j) - dp2.getAtributteDim(j)));
	}
	resultado = Math.sqrt(resultado);
```

```
		return resultado;

	}

	/**
	 * Retorna a Classificacao Obtida para a amostra
	 *
	 * @return classificacaoNovelty
	 */
	public boolean isClassificacaoNovelty() {
		return classificacaoNovelty;
	}

	/**
	 * Seta a Classificacao Obtida para a amostra
	 * @param classificacaoNovelty
	 */
	public void setClassificacaoNovelty(boolean classificacaoNovelty) {
		this.classificacaoNovelty = classificacaoNovelty;
	}

	/**
	 * Retorna uma representacao em string do ponto kMeans
	 *
	 * @return uma representacao em string do ponto de dados
	 */
	public String toString() {
		String points = "";
		for (Object point : this.dim) {
			points = points + point + " ";
		}
		// imprimir o ponto e a que cluster ele pertence
		return "(" + points + ")[" + this.numeroCluster + "]";
	}

	/**
	 * Main -- para testar essa classe
	 *
	 * @param args
	 *            argumentos em linha de comando
	 */
	public static void main(String[] args) {

	}

}
```

Classe cluster

```
package kmeans;


/**
 * Representa um cluster
 * @author         Daniel
 */
 class cluster {
```

```
/** Numero do Cluster, que varia de 0 a k-1 */
private int numeroCluster;

/** Limiar do Cluster */
private double limiarCluster;

/** Ponto de dado médio desse cluster */
private PontoKMeans centro;


/**
 * Retorna uma nova instancia de um cluster
 *
 * @param          _numeroCluster   o numero desse cluster
 */
public cluster(int _numeroCluster) {

        this.numeroCluster = _numeroCluster;

}


/**
 * Seta o ponto de dados médio desse cluster
 *
 * @param          pontoDadoMedioo novo ponto de dados medio desse cluster
 */
public void setCentro(PontoKMeans pontoDadoMedio) {

        this.centro = pontoDadoMedio;

}


/**
 * Seta o ponto de dados médio desse cluster
 *
 * @return         o ponto de dados medio desse cluster
 */
public PontoKMeans getCentro() {

        return this.centro;

}


/**
 * Retorna o numero do cluster
 *
 * @return         o numero desse cluster
 */
public int getNumeroCluster() {

        return this.numeroCluster;

}
```

```
	/**
	 * Retorna o limiar do cluster
	 *
	 * @return		o limiar desse cluster
	 */
	public double getLimiarCluster() {
		return limiarCluster;
	}

	/**
	 * Seta o limiar desse cluster
	 *
	 * @return		o limiar desse cluster
	 */
	public void setLimiarCluster(double limiarCluster) {
		this.limiarCluster = limiarCluster;
	}


	/**
	 * Retorna o numero do cluster em formato de string, para impressao no console,
	 * no final do treinamento.
	 *
	 * @return texto com o numero do cluster (em string)
	 */
	public String toString(){
		return "Cluster [" + this.getNumeroCluster() + "]: " +
			this.getCentro().toString()+" limiar -> "+this.getLimiarCluster();
	}
	/**
	 * Metodo main -- para testar essa classe
	 *
	 * @param		args	argumento de linha de comando.
	 */
	public static void main(String[] args) {


	}
}
```