ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

ESTUDO COMPARATIVO ENTRE TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA SISTEMAS DE DETECÇÃO DE INTRUSÃO (IDS)

Trabalho de Conclusão de Curso

Engenharia da Computação

Thyago Antonio Barbosa Vieira da Rocha
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, dezembro de 2005

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

ESTUDO COMPARATIVO ENTRE TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA SISTEMAS DE DETECÇÃO DE INTRUSÃO (IDS)

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Thyago Antonio Barbosa Vieira da Rocha
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, dezembro de 2005

Thyago Antonio Barbosa Vieira da Rocha

ESTUDO COMPARATIVO ENTRE TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA SISTEMAS DE DETECÇÃO DE INTRUSÃO (IDS)

Resumo

Detecção de Intrusão é um dos assuntos que ganha mais visibilidade dentro do atual cenário global. Esse fato pode ser explicado pelo aumento significativo da Internet em todo mundo, e com ele a exposição das empresas a esse mundo de interação e benefícios. Essa exposição não é só benéfica, pois invasões de pessoas não autorizadas estão sendo cada vez mais constantes. Os Sistemas de Detecção de Intrusão (IDS) aparecem como uma forma de identificar e realizar algum tipo de contramedida para tais invasões. Diversos métodos para reconhecimento de um ataque estão em uso ou sendo desenvolvidos. Um deles proposto envolve o uso de aprendizagem de máquina para identificação desses ataques. Este trabalho apresenta um estudo comparativo entre as técnicas Redes Neurais Artificiais (RNAs), do tipo MLP (*Multilayer perceptron*) e RBFN (*Radial Basis Functions Networks*) utilizando o algoritmo de treinamento DDA (*Dynamic Decay Adjustment*), Máquinas de Vetor Suporte (SVM) e técnicas que utilizam os vizinhos mais próximos (NN e kNN) aplicadas ao problema de detecção de intrusão. A ferramenta LIBSVM foi empregada para executar os treinamentos com SVM. O simulador WEKA foi utilizado para execução das técnicas NN, kNN e MLP, enquanto que e o SNNS foi utilizado para redes RBF-DDA. Ainda foi testada uma seleção de parâmetros com as redes RBF-DDA. Todos os treinamentos foram comparados empregando o método da validação cruzada. As técnicas foram analisadas com relação ao seu desempenho no erro de classificação, complexidade e tempo de processamento. As técnicas RBF-DDA e SVM obtiveram os melhores desempenhos.

Abstract

Intrusion Detection is one of the subjects that wins more visibility inside of the current global scene. This fact can be explained by the significant Internet increasement around the world, added to the exposition of the companies to this interaction and benefits world. This exposition is not only beneficial because there is very often invasion of non-authorized people. Intrusion Detection Systems (IDS) appear as a way to identify and prevent any kind of attack. Some attack recognition techniques are being used or being developed. One of them involves the use of machine learning in the identification of those attacks. This work presents a comparative study among the techniques: Artificial Neural Networks (ANN), MLP (Multilayer Perceptron) type and RBFN (Radial Basis Functions Networks), using the training algorithm DDA (Dynamic Decay Adjustment), Support Vector Machines (SVM) and other techniques that uses the neighbors (NN and kNN) applied to the problem of intrusion's detection. The tool LIBSVM was used to execute the trainings with SVM. The simulator WEKA was used to apply the techniques NN, kNN and MLP, while the SNNS was used for nets RBF-DDA. It was also verified a selection of parameters with the RBF-DDA nets. All the trainings had been compared using the crossing validation method. Those techniques were analyzed by its performance in the error classification, complexity and processing time. The techniques RBF-DDA and SVM had gotten the best performances.

Sumário

Índice de Figuras

Índice de Tabelas

Agradecimentos

Primeiramente, gostaria de agradecer a todos que participaram da elaboração desse trabalho:

A minha família, em especial a meus pais, Valdomiro Vieira da Rocha e Edenira Barbosa Vieira da Rocha, pelo apoio, pelo incentivo e pela confiança.

Outro agradecimento especial a minha namorada Adalgisa Maria Rodrigues da Silva, por toda atenção, pela ajuda e pela paciência.

A meus amigos que sempre estiveram do meu lado me incentivando e descontraindo o ambiente.

Ao meu orientador, Adriano Lorena Inácio de Oliveira, pela ajuda, pela orientação sempre pertinente e competente.

A todos os professores e colaboradores do curso de Engenharia da Computação da POLI pela minha formação e pelo meu engrandecimento profissional e social.

Um agradecimento maior a Deus, por me proporcionar todos esses momentos.

Introdução

O número constante e crescente de ataques a redes coorporativas se deve à exposição de informações importantes para intrusos que a Internet proporciona [28], por isso Sistemas de Detecção de Intrusão (IDS – *Intrusion Detection Systems*) estão tendo cada vez mais importância e visibilidade no mundo atual. Uma intrusão, em geral, é definida como um conjunto de ações que comprometem a integridade, a confiabilidade ou a disponibilidade de recursos [29]. Sistemas que detectam intrusões, basicamente, são conjuntos de hardware e software que trabalham juntos para identificar eventos inesperados, que podem indicar que um ataque irá acontecer, está acontecendo ou aconteceu [17]. Esses sistemas podem utilizar diversas técnicas para prever um ataque, entre elas Redes Neurais Artificiais (RNAs), caracterizadas por serem uma forma de computação não algorítmica e por lembrar, em algum nível, a estrutura do cérebro humano [1]. Outras técnicas também podem ser empregadas como Máquinas de Vetor Suporte (SVM) ou ainda técnicas que utilizam os vizinhos mais próximos, *Nearest Neighbor* (NN) e *k-Nearest Neighbor* (kNN).

O propósito de um IDS é distinguir entre intrusos e usuários normais. O sensor de um IDS é o mecanismo principal para fazer essa distinção, sua função é monitorar um *host* ou uma rede a fim de identificar intrusões, gerar *logs* localmente e gerar mensagens alertando a respeito da ocorrência de tais eventos. Dentro de um sensor existe uma parte nomeada analisador de eventos, cuja responsabilidade é identificar se um dado evento é um ataque ou não. É nessa parte que as técnicas de aprendizagem de máquina, através do seu poder de classificação, poderão se inserir em um IDS.

Um exemplo de IDS comercial é o *snort* que possui código aberto e é muito utilizado. Ele não utiliza técnicas de aprendizagem de máquina para analisar ataques. Suas características principais são análise de tráfego em tempo real e de registro de pacotes IP, além de gerar alertas em tempo real [36].

Neste trabalho, comparamos algumas técnicas de aprendizagem de máquina nesse problema utilizando a base de dados KDD *cup* 1999 [20]. Essa base foi concebida através da simulação de um ambiente de uma rede militar dos EUA, mais precisamente da força aérea dos Estados Unidos. Basicamente ela é composta de conexões “ruins”, denominadas ataques, e de conexões “boas” chamadas normais. Esse ambiente foi alimentado com conexões TCP (*Transmission Control Protocol*), cada uma dessas com um tamanho de 100 bytes, durante nove semanas. Em meio a essas conexões foram inseridos múltiplos ataques. Essa base de dados possui um total de 41 atributos de entrada, que incluem, por exemplo, duração da conexão, serviço de rede de destino, dados básicos sobre a conexão TCP, dentre outros (explicitados no Capítulo 2).

A utilização de RNA nesse problema pode ser justificada pela não linearidade da mesma e pela capacidade de adaptação que elas possuem. Isso é importante, pois como estão sendo identificados, os ataques atuais são uma combinação de ataques já existentes, por isso o analisador de eventos tem que se adaptar a cada uma dessas variantes. Outra vantagem é a facilidade de colher eventos para servir de exemplos no treinamento das RNAs, isto é feito simulando uma rede de computadores real e adicionando ataques a essa, armazenando os *logs* de cada conexão realizada, seja ela um ataque ou não.

A construção de um IDS que utiliza redes neurais pode ser dividida em três etapas:

1. Coleta dos dados para treinamento: nessa fase, é onde se obtêm os dados para o treinamento, sejam eles fornecidos por uma rede real ou através de uma simulação de uma rede;
2. O treinamento em si da rede: nessa etapa é onde vamos treinar as redes com a mesma base de dados, submetendo essa a várias técnicas de redes neurais distintas;
3. A rede neural treinada: nessa fase, a rede neural está pronta para distinguir ataques verdadeiros de conexões normais.

A utilização de Máquinas de Vetor Suporte é justificada pelo fato desta técnica recente, que utiliza apredizagem estatística, ter obtido melhor desempenho de classificação que outra técnicas, tais como RNAs em uma série de problemas importantes, como categorização de texto e reconhecimento óptico de caracteres (OCR) [11,34,35]. Uma vantagem adicional de SVM é que a técnica normalmente produz classificadores com poucas unidades escondidas.

As técnicas baseadas no vizinho mais próximo, NN e kNN, foram utilizadas, pois são simples e precisas. Além disso, essas técnicas possuem um parâmetro a ser modificado para o treinamento.

O trabalho foi dividido em quatro Capítulos. No primeiro foram introduzidos alguns conceitos inerentes ao tema, como as técnicas de aprendizagem de máquina e conceitos dos sistemas de detecção de intrusão.

O segundo Capítulo apresenta a base de dados, relatando como ela foi concebida, seus atributos, tipos de ataque que fazem partes dessa base, além de informar ao leitor algumas adequações que foram realizadas para formatar essa base para os simuladores utilizados nesse trabalho.

O terceiro Capítulo mostra os resultados obtidos com a utilização dessas quatro técnicas. Apresentamos a descrição de cada treinamento efetuado, bem como sua análise levando-se em consideração alguns fatores como: tempo de processamento, erro de validação cruzada e complexidade da rede. Por fim, foi feito um comparativo entre as técnicas abordadas.

O quarto Capítulo se refere às conclusões e trabalhos futuros.

Sistemas de Detecção de intrusão são primordiais para uma empresa que possua informações confidenciais e que esteja ligada à Internet, pois o risco de sofrer invasões é muito alto. Técnicas de aprendizagem de máquina surgem como um mecanismo interessante para detectar, classificar e identificar esses ataques.

Capítulo 1

Conceitos Básicos

Este Capítulo visa explicar conceitos básicos sobre o tema. Abordaremos as técnicas de aprendizagem de máquina utilizadas e os sistemas de detecção de intrusão (IDS).

1.1 Técnicas de aprendizagem de máquina

As técnicas empregadas neste trabalho são: redes neurais artificiais, representada por funções de bases radiais (*Radial Basis Function* - RBF) e perceptrons de multicamadas (*multilayer perceptron* - MLP), máquinas de vetor de suporte (SVM), técnica do vizinho mais próximo (NN) e técnica dos k-vizinhos mais próximos (kNN).

1.1.1 Redes Neurais Artificiais (RNAs)

Redes Neurais artificiais (RNAs), também chamadas de conexionismo ou redes de sistemas distribuídos [1], ressurgiram no final da década de 80, e hoje têm se tornando um amplo campo de pesquisa. RNAs nos permitem projetar sistemas não-lineares, podendo esses possuir um grande número de entradas, com o projeto baseado em relacionamentos do tipo entrada-saída [2]. Seus atrativos pricipais são a capacidade de *aprender* por exemplos e de generalizar as informação aprendidas. A generalização está associada à capacidade da rede de aprender através de um conjunto reduzido de exemplos e posteriormente dar respostas coerentes para dados não-conhecidos [1].

Outra característica importante é o fato das redes neurais artificiais não serem baseadas em regras ou programas, se constituindo assim em uma alternativa para a computação, visto que a utilização de algoritmos se restringe para algumas partes da execução de um rede neural artificial. Mas, o principal atrativo das RNAs é, sem dúvida, sua capacidade de aprender através de exemplos e de generalizar a informação aprendida. RNAs são inspiradas na biologia, particularmente na pesquisa do cérebro humano.

1.1.1.1 Inspiração na biologia

As RNAs tentam reproduzir as funções da rede neural biológica, buscando implementar seu comportamento básico e sua dinâmica. Como características comuns entre os dois sistemas temos que eles são baseados em unidades de computação paralela e distribuída que se comunicam por meio de conexões sinápticas, possuem detectores de características, redundância e modularização das conexões. A célula fundamental do cérebro é o neurônio, cada um desses neurônios se liga a milhares de outros continuamente e em paralelo. Os neurônios são divididos em três partes: corpo da célula, dendritos e axônios (Figura 1).

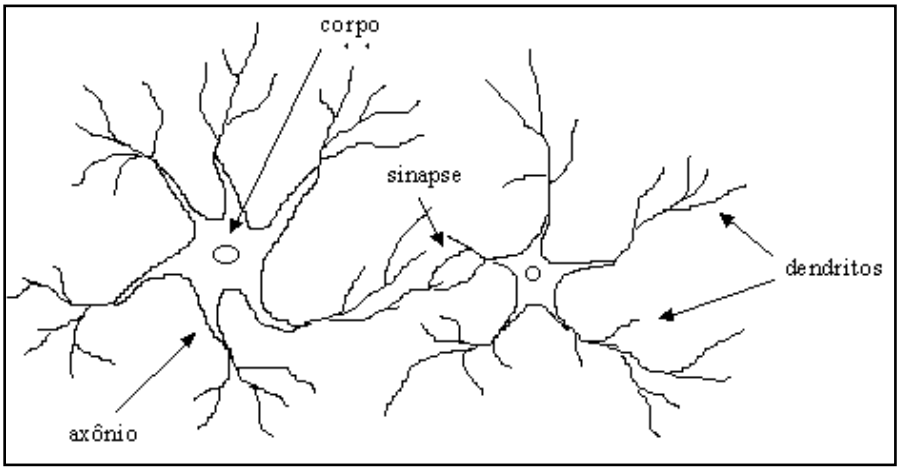


Figura 1. Partes de um neurônio biológico [31].

O corpo celular é a parte mais volumosa da célula; nela se localizam o núcleo e a maioria das estruturas citoplasmáticas. Os dendritos são prolongamentos finos e, geralmente, ramificados que conduzem os estímulos captados do ambiente ou de outras células em direção ao corpo celular. O axônio é um prolongamento estreito**,** geralmente mais longo que os dendritos, cuja função é transmitir para outras células os impulsos nervosos provenientes do corpo celular. Sinapse é uma região de contato muito próximo entre a extremidade do axônio de um neurônio e a superfície de outra célula. O impulso é transmitido de uma célula a outra através dessas sinapses.

1.1.1.2 Conceitos sobre redes neurais artificiais

Redes neurais artificiais possuem uma estrutura similar a um neurônio biológico. Essa estrutura foi desenvolvida por McCulloch e Pitts [1]. Eles modelaram uma estrutura com n terminais de entrada $x_1, x_2, ..., x_n$ (no neurônio biológico, poderíamos dizer que essas entradas representam os dendritos) e um terminal de saída (seria o axônio) para emular as sinapses. Os terminais de entrada têm associados a ele pesos $w_1, w_2, ..., w_n$. Em um neurônio biológico, um disparo acontece quando a soma dos impulsos que ele recebe ultrapassa seu limiar de excitação (*threshold*), já em um neurônio MCP (modelo de neurônio artificial de McCulloch e Pitts), a ativação de um neurônio é obtida através da aplicação de uma função de ativação, que ativa ou não a saída, dependendo do valor ponderado das suas entradas.

No modelo, a função de ativação é dada por uma função linear, cuja saída pode assumir os valores 0 ou 1. Por conseguinte, o nodo MCP ativa ou não sua saída, seguindo a equação abaixo:

$$\sum_{i=0}^{n} x_i w_i \geq \theta \qquad (1.1)$$

onde *n* é o número de entradas do neurônio, w_i é o peso associado à entrada x_i , e θ é o limiar (*threshold*) do neurônio. Uma simplificação realizada por McCulloch e Pitts no seu modelo diz respeito ao disparo de cada camada. Isso é feito sincronamente, ou seja, todos os neurônios são avaliados ao mesmo tempo. Já no sistema biológico sabe-se que não existe um mecanismo para realizar esse sincronismo [1]. Esse modelo possui algumas limitações e dentre elas podemos destacar as seguintes:

1. Esse modelo com uma camada só se adequava a problemas linearmente separáveis;
2. O modelo foi proposto com pesos fixos, não podendo estes ser ajustados.

A Figura 2 representa o modelo de McCulloch e Pitts:

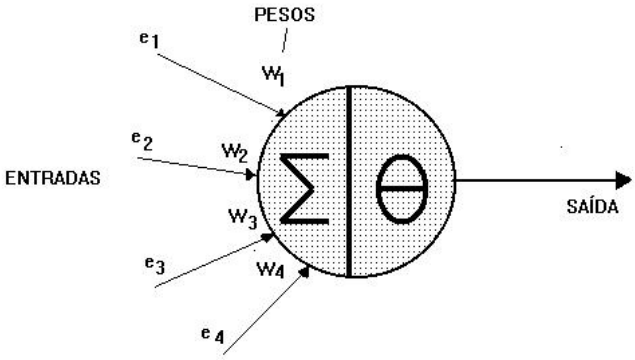


Figura 2. Modelo de um neurônio MCP.

Após o modelo de McCulloch e Pitts, foram propostos vários outros modelos que permitem a produção de saídas que não sejam necessariamente 0 ou 1 e com diferentes funções de ativação. A função de ativação linear é mostrada na equação abaixo e será exemplificada logo em seguida, em conjunto com outro tipos de funções:

$$y = ax \qquad (1.2)$$

onde a é um número real que define a saída linear para os valores de entrada, y é a saída e x a entrada. Em seguida, temos alguns exemplos de funções de ativação para redes neurais artificiais:

1. Função degrau: esssa função tem como valores de saída 0 ou 1 e é definida como:

$$f(x) = \begin{cases} 1 & se \quad x \geq 0 \\ 0 & se \quad x < 0 \end{cases} \qquad (1.3)$$

2. Função rampa: abaixo está um exemplo desse tipo de função:

$$f(x) = \begin{cases} 1 \quad se \quad x \geq \frac{1}{2} \\ x, \; -\frac{1}{2} < x < \frac{1}{2} \\ 0 \quad se \quad x < \frac{1}{2} \end{cases} \tag{1.4}$$

onde, 0 e 1 são os limites da função e (-½ e ½) é o intervalo que define a saída linear.

3. Função sigmóide: função cujos valores pertencem a intervalo contínuo, por exemplo, entre 0 e 1.

$$f(x) = \frac{1}{1+\exp(-\alpha x)} \tag{1.5}$$

onde, α determina a inclinação da função.

Além da função de ativação, as RNAs possuem uma arquitetura (topologia), cuja configuração é importante, pois restringe o tipo de problema que pode ser tratado pela rede [1]. Por exemplo, as redes descritas anteriormente, MCP, possuem apenas uma camada e só conseguem resolver problemas linearmente separáveis. Na Figura 3, estão ilustrados alguns exemplos de arquitetura de RNAs.

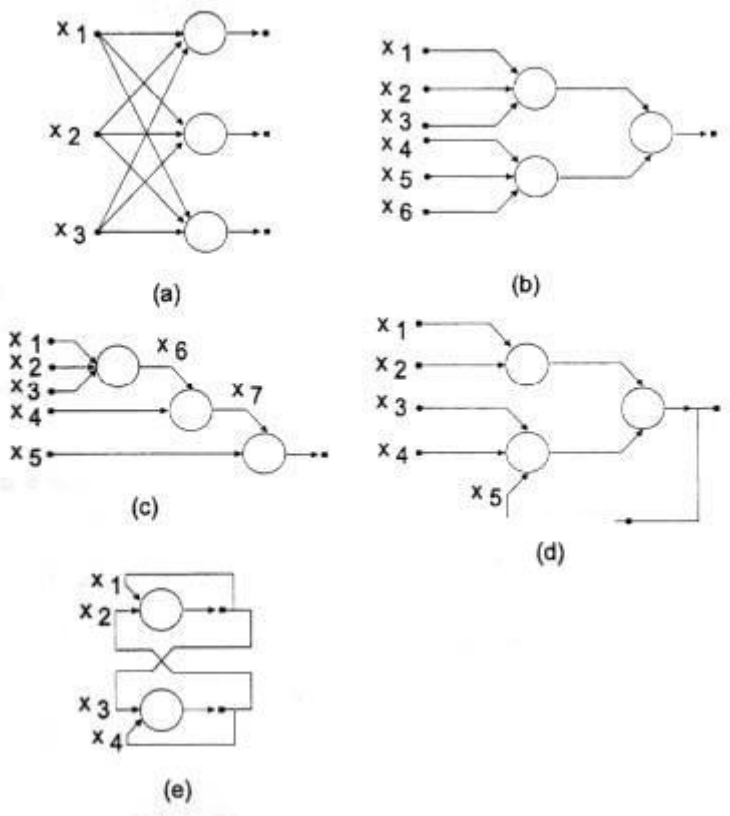


Figura 3. Arquiteturas de RNAs [1].

Essas arquiteturas referem-se a modelos:

1. Com única camada: só existe um nó entre a entrada e a saída (Figura 3. a, e);
2. Múltiplas camadas: existe mais de um neurônio entre a entrada e a saída (Figura 3. b, c, d).

Quanto às conexões entre os nodos podemos ter dois tipos:

1. *Feedforward*, ou acíclica: a saída de um neurônio em um a camada não pode ser utilizada como entrada em nenhuma camada anterior a esta (Figura 3. a, b, c);
2. *Feedback*, ou cíclica: a saída de algum neurônio de uma certa camada é utilizada como entrada para uma camada anterior a esta (Figura 3. d, e).

Nós já vimos até aqui como a rede neural artificial calcula suas saídas, como é sua arquitetura, agora vamos falar um pouco sobre como acontece a aprendizagem de uma RNA, visto que essas redes possuem a capacidade de aprender a partir de exemplos e fazer interpolações e extrapolações do que aprendem [1]. A aprendizagem se dá através de um algoritmo de aprendizagem, a definição de um algoritmo de aprendizagem pode ser: um conjunto de procedimentos bem-definidos para adaptar os parâmetros de uma RNA para que ela possa aprender uma determinada função [1]. A utilização de uma RNA se inicia por uma fase denominada fase de aprendizagem, é nela que, através de um processo iterativo, a rede vai ajustando os seus parâmetros que são os pesos das conexões entre as unidades de processamento. Essas, por sua vez, armazenam o conhecimento que a rede adquiriu do ambiente em que está operando. Diversos métodos de aprendizado foram desenvolvidos sendo que os dois principais são: aprendizado supervisionado e aprendizado não-supervisionado.

Aprendizado supervisionado é o método de aprendizado mais comum. Nesse método, a entrada e saída desejadas para a rede são fornecidas por um supervisor (professor) externo. Posteriormente, a saída dada é comparada com a saída obtida pela rede, tendo como objetivo ajustar os parâmetros da rede de forma a encontrar uma representação interna a partir dos pares de entrada e saída fornecidos. Nesse método, a soma dos erros quadráticos de todas as saídas é normalmente utilizada como medida de desempenho da rede. Existe uma desvantagem da utilização desse método: na ausência do professor a rede não conseguirá aprender novas estratégias para situações que não pertençam ao escopo de exemplos conhecidos. A Figura 4 ilustra o aprendizado supervisionado:

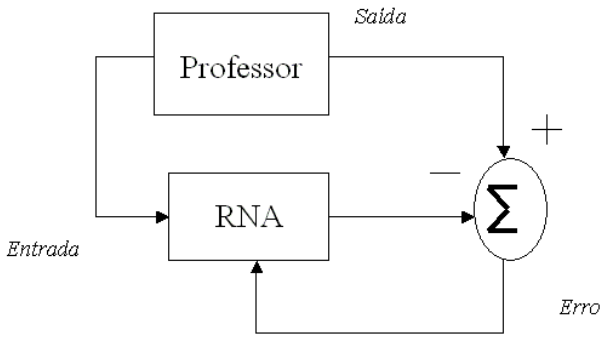


Figura 4. Treinamento supervisionado [1].

Aprendizado não-supervisionado: nesse método não existe o professor ou supervisor para acompanhar o processo de aprendizado, apenas padrões de entrada são fornecidos para a rede, e

através de regularidades estatísticas das entradas são estabelecidas algumas representações internas. Desse modo, esse método de aprendizado é possível apenas quando existe redundância nos dados de entrada. A Figura 5 demonstra esse tipo de aprendizado.

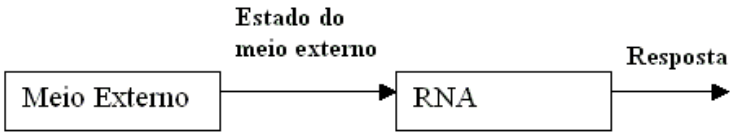


Figura 5. Treinamento não-supervisionado [1].

Uma forma de utilizar redes neurais em sistemas de detecção de intrusão, é criar um sistema que aprenda a predizer um próximo comando baseado numa seqüência prévia de comandos pertencentes a um usuário específico [3]. Em uma empresa, determinados funcionários são encarregados de realizar algumas tarefas, e essas exigem certas rotinas que envolvem programas nos computadores, a execução desses programas exigem certos comandos nos computadores e esses serão aprendidos pela técnica de aprendizagem de máquina. A construção de uma rede neural para um IDS consiste em três fases:

1. Coleta dos dados para treinamento, esses dados podem ser obtidos por *logs* de auditoria para cada usuário por um determinado período. Um vetor é formado para cada dia e cada usuário, assim ele pode mostrar quais comandos um usuário frequentemente executa;
2. Treinar a rede neural para reconhecer um usuário através dos vetores de distribuição de comandos;
3. A rede neural identifica um usuário baseado nos vetores de distribuição de comandos; se a sugestão da rede for diferente do usuário real, uma anomalia será sinalizada.

1.1.1.3 Redes MLP (*MultiLayer Perceptron*)

Para resolver problemas não-linearmente separáveis foram criadas as redes MLP, pois esse tipo de rede possui pelo menos duas camadas permitindo a aproximação de qualquer função contínua. As redes MLP derivam de um modelo denominado *perceptron* proposto por Frank Rosenblatt em 1958 [4]. Com esse modelo, apenas problemas linearmente separáveis poderiam ser solucionados. A utilização de uma camada escondida aumentou o poder computacional das MLP. A precisão a ser obtida e a implementação da função objetivo dependem do número de nodos utilizados nas camadas intermediárias.

Como já vimos, um dos principais aspectos das redes neurais artificiais é a utilização de uma função de ativação. Para redes MLP a mais empregada é a sigmoidal logística. Essa função é representada no plano cartesiano mostrado na Figura 6.

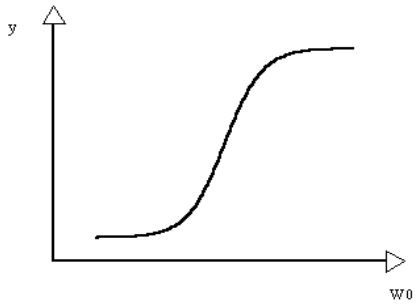


Figura 6. Função sigmoidal logística.

Em uma rede multicamada, o processamento realizado por cada nodo é definido pela combinação dos processamentos realizados pelos nodos da camada anterior, que estão conectados a ele. Para uma rede com duas camadas intermediárias pode-se dizer que o seguinte processamento ocorre em cada camada:

- Primeira camada intermediária: cada nodo traça retas (essas retas são criadas de acordo com a função de ativação da camada, sua orientação é dada pelo vetor de pesos) no espaço de padrões de treinamento;
- Segunda camada intermediária: cada nodo combina as retas traçadas pelos nodos da camada anterior (primeira camada intermediária) conectados a ele, formando regiões convexas, onde o número de lados é definido pelo número de unidades conectadas a ele. Abaixo, a Figura 7, mostra um exemplo de uma região convexa:

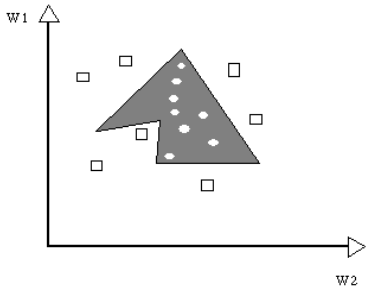


Figura 7. Regiões definidas pelo processamento da segunda camada intermediária.

- Camada de saída: cada nodo forma regiões que são combinações das regiões convexas definida pelos nodos conectados a ele da camada anterior (segunda camada intermediária). Na Figura 8, demonstramos um exemplo das combinações de regiões convexas.

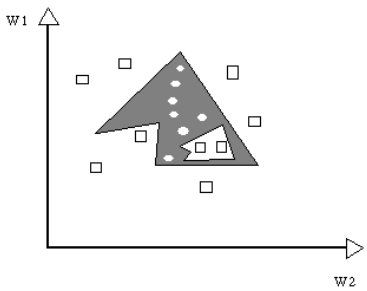


Figura 8. Regiões definidas pelo processamento da camada de saída.

Pode-se dizer que as camadas intermediárias de uma MLP funcionam como detectores de características. Eles geram uma representação interna dos padrões de entrada, que é utilizada para definição da saída da rede. A camada de saída de uma MLP emprega uma técnica denominada *winner-takes-all* [4], com isso a classe fornecida pela rede corresponderá à maior saída da rede.

O número de nodos contido em cada camada é definido empiricamente, esse número depende muito da distribuição dos padrões para treinamento e validação da rede. O número ideal de neurônios é influenciado por vários fatores, como:

- Número de exemplos de treinamento;
- Quantidade de ruído presente nos exemplos;
- Complexidade da função a ser aprendida;
- Distribuição estática dos dados de treinamento.

A alocação de unidades intermediárias (neurônios) deve ser suficiente para solucionar o sistema em questão. É preciso ter cuidado para não utilizar unidades demais, pois pode levar a rede a memorizar os padrões de treinamento, ao invés de extrair as características gerais que permitirão a generalização ou o reconhecimento de padrões que não fizeram parte do treinamento. Esse problema é denominado *overfitting*. Por outro lado, se utilizarmos poucos neurônios na camada intermediária, podemos fazer com que a rede gaste muito tempo para encontrar uma representação ótima.

Uma das formas empregadas para evitar o *overfitting* é estimar o erro de generalização durante o processo de treinamento. Para isso, a massa de dados é dividida em dois conjuntos: o de treinamento e o de validação. O conjunto de treinamento continua sendo utilizado na atualização dos pesos, enquanto que o conjunto de validação é empregado para estimar a capacidade de generalização da rede durante o processo de aprendizagem. O treinamento deve ser interrompido quando o erro de validação chegar ao seu mínimo global. Isto pode ser verificado através de técnicas como o critério de parada GL5, onde o treinamento é interrompido quando o erro de validação subir por cinco iterações consecutivas.

Neste trabalho, a forma utilizada para finalizar o treinamento foi o número de épocas (quanitdade de vezes que a rede é treinada por inteiro, ou seja, apresentação de todos os padrões de treinamento a rede), que ficou definido em 500, visto que em artigos anteriores [2,16], o número máximo de épocas de treinamento não chegou a esse número.

Algoritmo de treinamento: *back-propagation*

O algoritmo de treinamento escolhido para este trabalho foi o *back-propagation* [1,5]. Esse algoritmo foi um dos principais responsáveis pelo ressurgimento do interesse em RNAs, visto que desde a criação do *perceptron* as redes neurais haviam entrado numa fase de decadência, devido essa técnica só ser capaz de resolver problemas linearmente separáveis. O *back-propagation* é um algoritmo supervisionado que emprega pares (entrada, saída) e por meio de um mecanismo de correção de erros, ajusta os pesos da rede. Ele é dividido em duas fases: *forward* e *backward*, a primeira calcula a saída da rede para um dado padrão de entrada, já a segunda utiliza a saída desejada e a saída fornecida pela rede para ajustar os pesos das conexões. A Figura 9 demonstra essas fases.

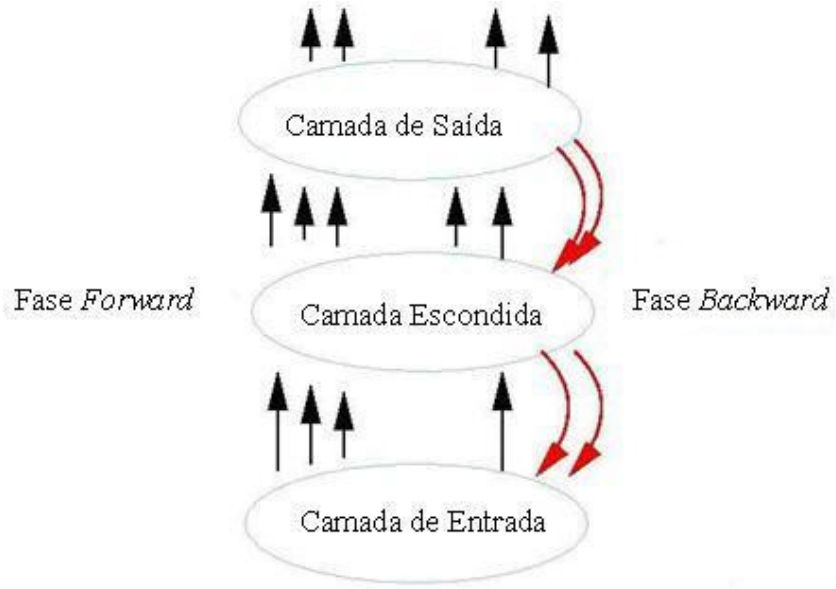


Figura 9. Fases do algoritmo *back-propagation*.

A fase *forward* se inicia com a apresentação do padrão primeira camada (camada de entrada), após os nodos calcularem suas saídas, essas são passadas para a camada posterior (camada escondida). Essa camada realiza o mesmo processo e as saídas produzidas pela última camada (camada de saída) são comparadas às saídas desejadas, tendo fim a fase *forward*.

A fase *backward* se inicia a partir dessa comparação feita no último passo da fase *forward*. Os pesos da camada atual são ajustados; os erros das camadas anteriores são calculados utilizando os erros dos nodos das camadas seguintes conectados a ele, ponderados pelos pesos das conexões entre eles.

1.1.1.4 Redes RBF (*Radial Basis Function*)

RBF são redes que empregam funções de base radiais. Esse nome se deve ao uso dessas funções nas camadas intermediárias dessas redes. RBF se diferencia da maioria das redes multicamadas por utilizar como argumento da função de ativação a distância entre seus vetores de entrada e de pesos, enquanto que MLPs utilizam o produto escalar do vetor de entrada e do vetor de peso para um nodo. Com a utilização dessa função na camada intermediária RBF é capaz de separar os padrões de classes distintas através de hiperelipsóides. As funções de bases radiais são representantes de uma classe de funções, cujo valor diminui ou aumenta em relação a um ponto central [1]. As mais comuns para serem empregadas em redes RBF são:

- Função Gaussiana: $f(u) = \exp(\frac{-v^2}{\sigma_i^2})$ (1.6)
- Função multiquadrática: $f(u) = \sqrt{(v^2 + \sigma^2)}$ (1.7)
- Função *thin-plate-spline* $f(u) = v^2 \log(v)$ (1.8)

Onde $v = \| \vec{x} - \vec{\mu} \|$ é, geralmente, dado pela distância Euclidiana $\vec{x}$ é o vetor de entrada, e μ e σ representam respectivamente o centro e a largura da função radial.

A arquitetura de uma rede RBF costuma ter apenas uma camada intermediária. Como podemos conferir na Figura 10. A camada intermediária utiliza funções de base radiais, agrupando os dados de entrada em *clusters*. Com isso, essa camada transforma um conjunto de

padrões não-linearmente separáveis, ou seja, que não podem ser separados apenas traçando um plano ou uma reta, em um conjunto de padrões linearmente separáveis. A camada de saída classifica os padrões recebidos da classe anterior. Podem ser utilizadas redes do tipo *perceptron* ou *adaline* nessa camada, uma vez que seus padrões são linearmente separáveis.

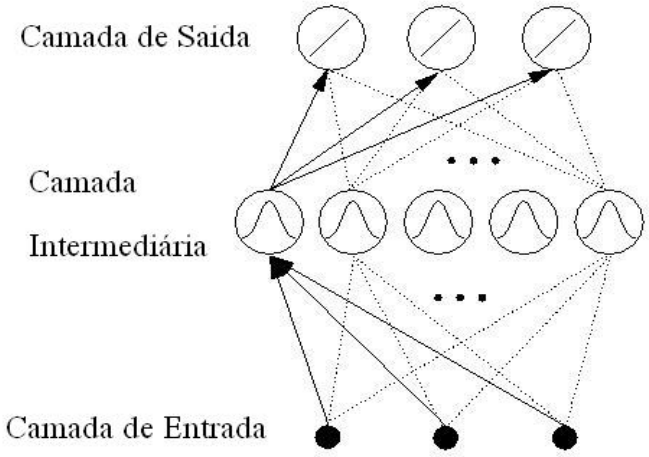


Figura 10. Arquitetura de uma rede RBF [32].

Quanto ao particionamento do espaço de entrada, as redes RBF, ao contrário das redes MLP que empregam hiperplanos para fazer o particionamento, utilizam hiperelipsóides, onde cada um desses agrupa padrões de mesma classe formando regiões específicos para cada classe. Esse particionamento realizado pela rede RBF implica que a rede só poderá classificar novos padrões se esses forem das mesmas classes utilizadas para o seu treinamento. Ou seja, se um determinado padrão pertencente à outra classe, que não tenha sido utilizada durante o treinamento, for apresentado à rede para ser classificado, a rede não saberá como classificar esse novo padrão [1].

Uma vantangem deste tipo de abordagem de RNA´s, é o fato de a rede não classificar um padrão discrepante, classificando-o como desconhecido. A desvantagem desse tipo de técnica é o fato dela ter um bom desempenho para problemas bem definidos [1].

Algoritmo de treinamento: DDA (*Dynamic Decay Adjustment*)

Existem vários métodos de treinamento para redes RBF. Neste trabalho, vamos utilizar o algoritmo DDA (*Dynamic Decay Adjustment*) [6,7,8,9,30]. Esse algoritmo se baseia no algoritmo contrutivo, utilizado para redes RBF, RCE (*Restricted Coulomb Energy*) [10]. Ele corrige um problema do RCE que é o de se confundir em áreas de conflito como o ilustrado na Figura 11. Quando um padrão é apresentado à rede, essa cria uma unidade RBF para classificá-la. Se um padrão de uma outra classe for inserido no treinamento, irá também criar uma unidade RBF e isso pode gerar uma região comum às duas gaussianas, denominado área de conflito. O algoritmo RCE não trata desse caso e, por conseguinte, pode não conseguir classificar corretamente esse padrão, podendo este ser atribuído a classe errada.

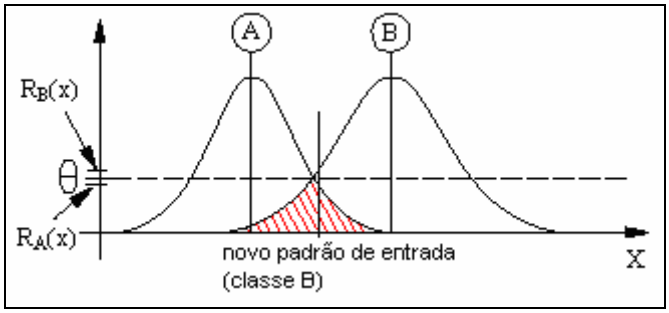


Figura 11. Exemplo de conflito de padrões [6].

O algoritmo DDA tem algumas peculiaridades que o diferenciam tanto na arquitetura usada por essas redes, como também no tempo utilizado para realização do treinamento. Quando aplicado a uma rede RBF, os nós da camada escondida utilizam funções gaussianas para processarem os valores de entrada [1].

O algoritmo DDA é um algoritmo construtivo, pois, inicialmente, é criada uma camada escondida sem neurônios. Cada gaussiana, apresentada na Figura 11, representa uma unidade na camada escondida de uma RBF, ou seja, um nodo. A partir do momento que o treinamento ocorre, novas unidades vão sendo adicionadas dentro dessa camada escondida. Se houver a necessidade da inclusão de um novo neurônio, isso será determinado dinamicamente durante o treinamento.

O DDA utiliza dois parâmetros específicos para decidir se um novo neurônio deverá ser introduzido na camada intermediária da rede RBF [2]. O limiar positivo θ^+ é utilizado para verificar se, para um novo padrão usado para o treinamento da rede, existe algum protótipo (uma gaussiana) da mesma classe com ativação acima do θ^+. Caso exista tal protótipo, não será adicionado novo protótipo à rede; ao invés disso o peso de um protótipo já existente será incrementado. O limiar negativo θ^- é utilizado para ajudar a solucionar o problema de conflitos de padrões que podem vir a existir durante o treinamento [6, 8].

O algoritmo 1, mostra um pseudocódigo para o DDA durante uma época de treinamento.

Algoritmo 1. Algoritmo DDA para treinamento de RBFs (uma época de treinamento)

```
//inicializa pesos com 0,0:
FORALL protótipos p^k_i DO
    A^k_i = 0,0
ENDFOR
//treina para uma época completa
FOR ALL padrão de treinamento (x, c) DO
   IF ∃p^c_i: R^c_i(x) >= θ+ THEN
      A^c_i + = 1,0
   ELSE
      // introduz um novo protótipo
      Adiciona um novo protótipo p^c_mc+1 com:
      r^c_mc+1 = x
      σ^c_mc+1 = max k≠c ∧ 1<=j<=mk {σ:R^c_mc+1( r^k_j ) < θ-}
      A^c_mc+1 = 1.0
      m_c + = 1
   ENDIF
    //ajusta protótipos conflitantes
   FORALL k ≠ c, 1 <= j <= m_k DO
      σ^k_j = max{σ : R^k_j( x ) < θ-}
   ENDFOR
ENDFOR
```

Podemos verificar no algoritmo 1 que, inicialmente, todos os pesos recebem valor 0 para cada protótipo presente na rede. Depois da inicialização dos pesos, para cada padrão de treinamento, verifica-se o valor de sua ativação e esse valor é comparado com o parâmetro θ^+. Se a ativação for maior ou igual ao θ^+, não é adicionado um novo protótipo à rede e o peso será incrementado de 1. Caso a ativação seja menor que o valor do parâmetro θ^+, um novo protótipo será adicionado à rede com o valor do seu centro igual ao vetor de entrada, o peso do novo protótipo será igual a 1 e o número de protótipos é incrementado de 1. O próximo passo é fazer os ajustes nos protótipos conflitantes da forma como descreve o algoritmo 1.

Temos, a seguir, a Figura 12 que demonstra um exemplo de rede utilizando o algoritmo DDA. Na Figura 12 (a), é inserido um padrão de treinamento pertencente à classe A do problema em questão, criando uma gaussiana. A seguir na Figura 12 (b), um padrão da classe B é inserido, portanto uma nova gaussiana terá que ser criada, e a gaussiana do padrão A terá que se ajustar para passar na intersecção do centro de B com o valor θ-. No terceiro passo da Figura 12 (c), um novo padrão B é inserido, e como a intersecção do seu centro com a gaussiana do padrão B anterior é maior que o θ+, a gaussiana é incrementada, tendo seu peso passado para dois. Na Figura 12 (d), um novo padrão A á inserido e a interseção do centro desse novo padrão com a gaussiana A já existente é menor que o θ+, sendo assim uma nova gaussiana A teve que ser criada.

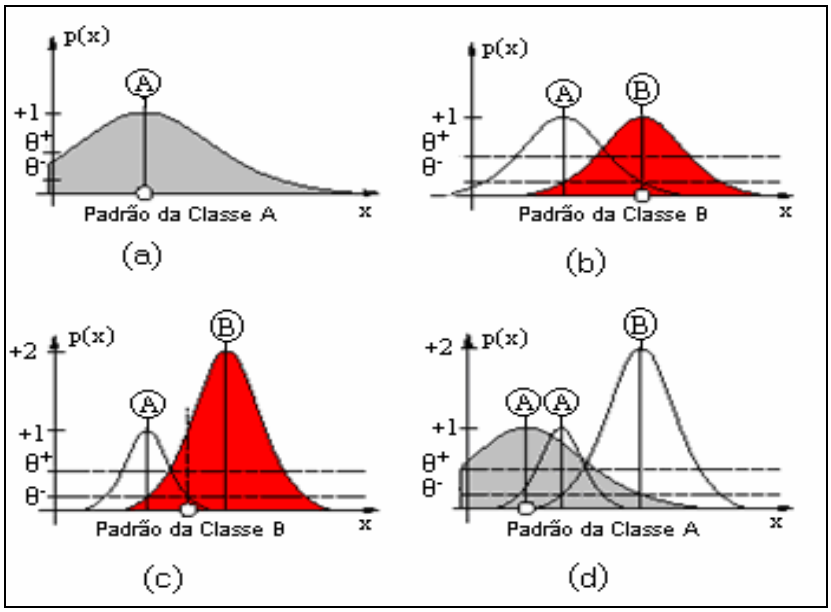


Figura 12. Exemplo da execução do algoritmo DDA [6].

Neste trabalho, também utilizamos uma técnica de seleção do parâmetro θ- [7, 8, 9]. O algoritmo DDA com seleção do parâmetro θ^-, propõe a utilização de valores menores que o *default* para o parâmetro θ^-, porém utiliza um método para selecionar um valor ótimo para esse parâmetro. O conjunto de dados contendo todos os padrões que serão utilizados pela rede neural RBF-DDA é dividido, inicialmente, em dois conjuntos: conjunto de treinamento e conjunto de teste. Realizada essa divisão inicial nos dados, o conjunto de treinamento é novamente dividido em duas novas partes: uma parte é um novo conjunto de treinamento, e outra parte é utilizada como um conjunto de validação. Esses dois novos conjuntos de dados serão usados para treinamento e validação da rede respectivamente, com o intuito de alcançar um valor para o parâmetro θ^- que seja ótimo, ou seja, um valor que proporcione uma taxa de generalização ideal

para a rede, e essa possa classificar novos valores sem a presença de *overfiting*. O método para seleção do parâmetro θ^- usando o algoritmo DDA é mostrado no algoritmo 2.

Algoritmo 2. Algoritmo DDA com seleção de θ^-.

$\theta^-_{opt} = \theta^- = 10^{-1}$
Treinar uma RBF-DDA com θ^- usando o conjunto de treinamento reduzido e testa com o conjunto de validação para obter *ValError* = *MinValError*
REPEAT
  $\theta^- = \theta^- \times 10^{-1}$
  Treinar uma RBF-DDA com o θ^- usando o conjunto de treinamento reduzido e testar com o conjunto de validação para obter o *ValError*
  IF *ValError* < *MinValError*
    MinValError = *ValError*
    $\theta^-_{opt} = \theta^-$
  ENDIF
UNTIL *ValError* > *MinValError* **OR** $\theta^- = 10^{-10}$
Treinar uma RBF-DDA com o θ^-_{opt} usando o conjunto de treinamento completo
Testar o RBF-DDA otimizado com o conjunto de teste

1.1.2 Máquinas de vetor de suporte (SVM)

Máquinas de vetor de suporte (SVM) [11, 12, 13] têm obtido sucesso em um grande número de aplicações, que variam desde identificação de partículas, identificação de face, categorização de texto, bioinformática e em banco de dados de marketing [11]. O modo como a aproximação é realizada tem base na teoria estatística [12]. SVM funciona construindo um hiperplano N-dimencional que otimamente separa os dados em duas categorias de forma ótima [13]. A Figura 13 demonstra um exemplo de como são separadas duas categorias.

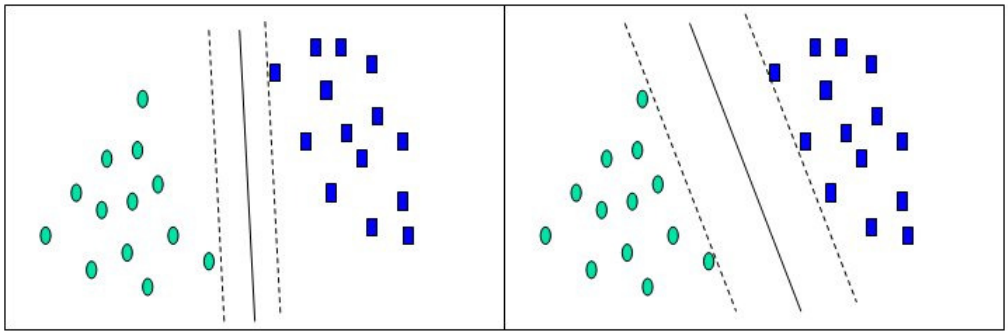


Figura 13. Exemplo em duas dimensões da utilização de SVM como classificador [13].

Na literatura de SVM, uma variável de predição é denominada atributo, e um atributo transformado, que é empregado na definição do hiperplano, é chamado característica. A tarefa de escolher a representação mais satisfatória é conhecida como seleção de característica. O conjunto de características que descrevem um caso é chamado de vetor. Assim, o objetivo de SVM é modelar um hiperplano ótimo que separa *clusters* do vetor. Os vetores perto do hiperplano são os *vetores de suporte*. A Figura 14 ilustra bem esses conceitos.

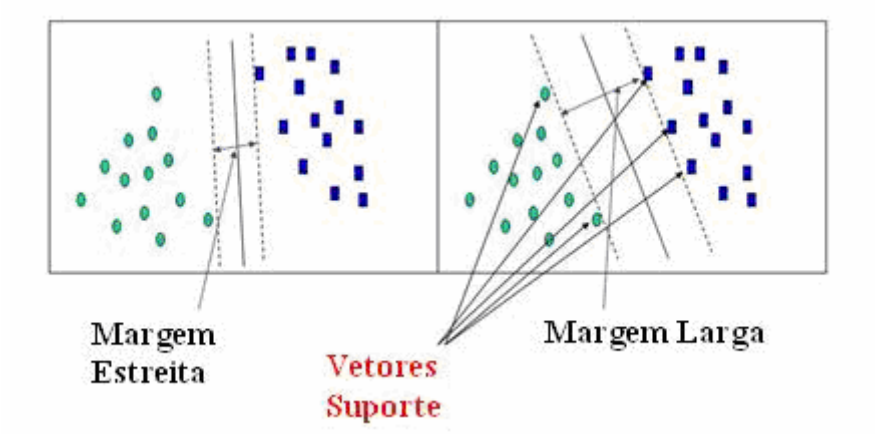


Figura 14. Exemplo em duas dimensões dos vetores de suporte [13].

Algumas das principais características das SVMs são:

- Boa capacidade de generalização – os classificadores gerados por uma SVM em geral alcançam bons resultados em termo de generalização. Essa capacidade é medida por sua eficiência na classificação de dados que não pertençam ao conjunto utilizado em seu treinamento, portanto, é evitado o *overfitting*.
- Robustez em grandes dimensões – as SVMs são robustas diante de grandes quantidades de dados.
- Teoria bem definida – as SVMs possuem uma base teórica bem estabelecida dentro da Matemática e Estatística.

O exemplo descrito nas Figuras 13 e 14 é simples, pois só possui duas dimensões. Nesse exemplo, assumimos que existem duas categorias, sendo uma categoria representada por retângulos e a outro por círculos. Nesse caso idealizado, uma categoria se localiza no canto inferior esquerdo e a outra categoria se localiza no canto superior direito. SVM tentar achar um hiperplano com uma dimensão que separe as duas categorias. Existe uma infinidade de hiperplanos que podem separar essas categorias. Na Figura 14, temos dois exemplos. O hiperplano do lado esquerdo da Figura 14 tem uma margem de separação pequena, enquanto que o hiperplano do lado direito da mesma figura tem uma margem de separação maior. As linhas pontilhadas paralelas à linha divisória marcam a distância entre essa e os vetores mais próximos da linha. A distância entre as linhas pontilhadas é chamada de margem. O vetores mais próximos as linha pontilhadas são chamados de vetores de suporte, como descrito no parágrafo anterior e ilustrados na Figura 14.

Uma SVM tenta achar um hiperplano de forma que o tamanho da margem seja máximo para que haja um maior limite de decisão para padrões não-lineares dentro do espaço de entrada. Outra decorrência disso é que a rede tende a generalizar melhor. Com a margem máxima há uma separação melhor entre as classes que fazem parte do problema.

SVM se utiliza de funções denominadas *kernel*. Essas funções são capazes de mapear o conjunto de dados em diferentes espaços, fazendo com que um hiperplano possa ser usado para fazer a separação. Os principais tipos de funções k*ernel* são:

- Linear: $K(x_i, x_j) = x_i^T x_j$

- Polinomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$
- Sigmóide: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$
- Função de base radial (RBF): $K(x_i, x_j) = \exp(-\gamma \| x_i - x_j \|^2), \gamma > 0$

A função *kernel* que vamos utilizar será a função de base radial. Esse tipo de *kernel* utiliza alguns parâmetros para o treinamento, nós utilizamos dois deles que são o C e o V. O parâmetro C é o parâmetro de penalidade do termo de erro (C>0) e V é a validação cruzada será explicada no Capítulo 3.

1.1.3 Técnica baseadas nos vizinhos mais próximos (NN e kNN)

O método do vizinho mais próximo para classificação é simples e preciso [14,15]. Nesse método, um nodo padrão é nomeado para a classe do seu vizinho mais próximo de um conjunto de treinamento rotulado e armazenado. A Figura 15 demonstra como é feita a classificação do método NN. Nela o novo padrão, marcado com uma cruz, será classificado como floco de neve (asterisco), devido a esse ser o rótulo do seu vizinho mais próximo.

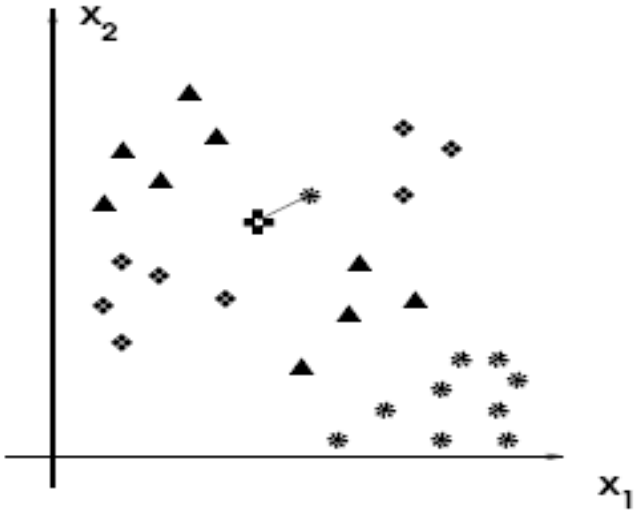


Figura 15. Exemplo de classificação no método NN [33].

Todos os dados rotulados são armazenados e usados no processo, isso faz com que NN precise de um tempo significativo para executar e também ocupe bastatnte memória.

O método kNN utiliza também a influência dos vizinhos. Entretanto, classifica seus padrões de acordo com uma matriz de votos dados por seus k vizinhos mais próximos. O funcionamento desse método é explicado dessa forma [24,38]:

1. Calcula-se as distâncias das amostras;
2. Agrupa-se as amostras por proximidade;
3. Os vizinhos mais próximos dão seus votos para sua classe;
4. A classe que possuir mais votos fica com o padrão.

Uma das grandes vantagens do kNN é que ele não depende da distribuição dos dados, sendo mais indicado para classificação de dados assimétricos.

O ideal quando se escolhe o número k de vizinhos, é escolher um número ímpar para evitar que duas classes possuam o mesmo número de votos na escolha da classe de um determinado padrão.

1.2 Sistemas de Detecção de Intrusão (IDS)

1.2.1 O que é um IDS?

Ataques a computador lançados a partir da Internet, são capazes de causar danos enormes, devido ao aumento da importância dos serviços fornecidos pela rede. Esses ataques crescem cada dia, o que pode ser comprovado a partir de dados estatísticos de órgãos como CERT e NBSO [19]. É complicado prevenir ataques com políticas como *firewalls*, políticas de segurança ou outros mecanismos, pois os sistemas e aplicativos possuem fraquezas desconhecidas ou falhas (*bugs*). Além disso, freqüentemente, os atacantes exploram vulnerabilidades desses sistemas e/ou protocolos de rede. Sistemas de detecção de intrusão são projetados para descobrir ataques que inevitavelmente acontecem, mesmo com a aplicação das políticas de segurança [16].

O processo de detecção de intrusão se caracteriza por identificar e relatar atividade maliciosa agindo em computadores e/ou recursos da rede [17]. Tendo em mente essa definição de detecção, podemos definir, ainda, um sistema de detecção de intrusão como sendo: um sistema de hardware e software que trabalham unidos para identificar eventos inesperados que podem indicar se um ataque irá acontecer, está acontecendo ou aconteceu [17]. A função de um IDS pode ser também, além de detectar e identificar um ataque, responder ao ataque ativando medidas preventivas e alertando o administrador de rede. Um IDS coleta informações de uma variedade de sistemas e recursos da rede e assim analisa essas informações para verificar se há algum sinal fora da normalidade. As principais funções executadas por um IDS são [18]:

- Análise e monitoração do usuário e do sistema;
- Avaliação da integridade crítica do sistema e seus arquivo de dados;
- Reconhecimento de padrões de atividades que indiquem um ataque;
- Informação ao processo de detecção.

1.2.2 Conceitos e tipos de um IDS

Alguns conceitos são inerentes a sistemas de detecção de intrusão como [17]:

- Ataque: ação inteligente que põe em risco o funcionamento de um sistema, explora vulnerabilidades inerentes ao sistema ou inerentes ao protocolo de rede;
- Vulnerabilidade: é uma falha que pode ter origem no sistema operacional, no protocolo de rede ou em serviços de qualquer componente no sistema que permita acesso ou intervenção de pessoas não autorizadas.
- Sensor: principal parte de um IDS, cuja função é monitorar um *host* ou rede a fim de identificar intrusões gravar *logs* localmente e gerar mensagens alertando tais eventos;
- Estação de gerenciamento: é uma estação encarregada de gerenciar um ou mais sensores;

- Evento: é uma ocorrência, detectada pelo sensor, na base de dados;
- Respostas ou contramedidas: ações que podem ser programadas na ocorrência de um determinado evento.

O propósito de um IDS é distinguir entre intrusos e usuários. Devido à enorme complexidade das redes atuais, essa distinção se torna difícil. Isso pode acarretar perda de desempenho dos IDS. Desse problema descrito, podemos apresentar dois conceitos inerentes, o dos falsos positivos e falsos negativos. Os erros de falso positivo ocorrem quando o sensor do IDS interpreta mal uma conexão normal, classificando-a como um ataque. Esse erro pode degradar a produtividade do sistema pela ativação desnecessária de contramedidas. Os erros de falso negativo são fatais, pois acontecem quando uma conexão de ataque é classificada como uma conexão normal.

Várias classificações existem para definir o tipo de IDS. Uma delas é classificá-lo quanto à tecnologia do analisador de eventos, que é a parte do sensor responsável pela identificação dos ataques. O analisador de eventos pode ter como tecnologia [17]:

- Análise de assinaturas: seu funcionamento é similar a de um antivírus. É o método mais utilizado.
- Análise estatística: constrói modelos estatísticos do ambiente baseados em fatores como: duração média de uma sessão de telnet [37], por exemplo. Qualquer desvio comportamental do sistema pode ser classificado com suspeito.
- Sistemas adaptativos: inicia generalizando regras de aprendizado para o ambiente em que está inserido, e então determinar o comportamento dos usuários com o sistema. Passado esse período, o sistema estará apto para distinguir entre conexões normais e ataques.

1.2.3 IDS utilizando aprendizagem de máquina

O propósito deste trabalho, como já foi descrito, é identificar qual técnica de aprendizagem de máquina fornece melhores resultados, ou ainda, se adapta melhor para sistemas de detecção de intrusão. Para isso o tipo de IDS que trabalha com aprendizagem de máquina são os sistemas adaptativos descritos na Seção 1.2.2.

As técnicas de aprendizagem de máquina se inserem em um IDS no analisador de eventos, que por sua vez, em geral, parte do sensor, e é responsável pela classificação das conexões. Como foi descrito na Seção anterior, a implementação desses sistemas obedece a duas fases distintas. Na primeira fase, chamada de fase de treinamento, a técnica começa a aprender o funcionamento do sistema em questão e como os usuários interagem com o mesmo. Na segunda fase, o sistema começa a fazer interpolações e extrapolações daquilo que ele aprendeu na fase de treinamento e, desse modo, ele estará habilitado para realizar a distinção das conexões, ou seja, classificá-las em normais ou ataques.

1.2.4 Arquitetura de um IDS

Todo IDS possui alguns componentes em comum; cada um responsável por uma parte importante na detecção de alguma anormalidade danosa ao sistema. A seguir, temos um exemplo de arquitetura proposto pela IDWG (*Intrusion Detection Exchange Format Working Group*) [39].

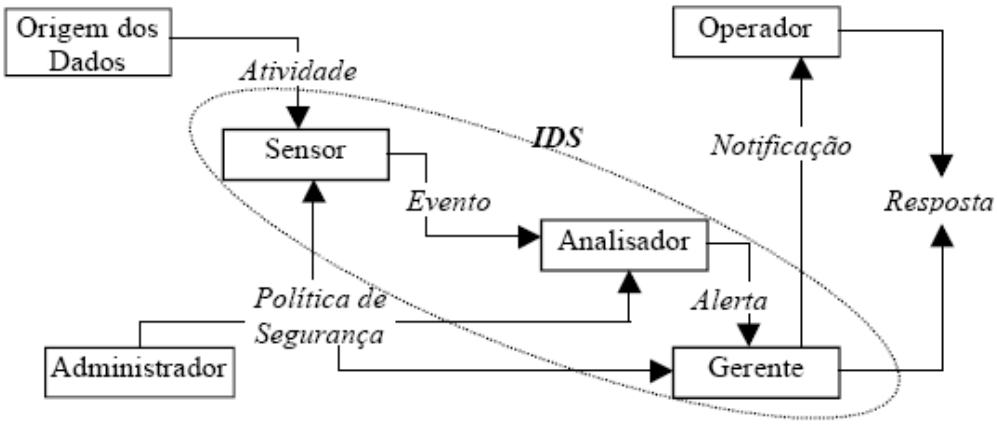


Figura 16. Componentes de um IDS segundo o IDWG [39].

Na Figura 16, podemos visualizar o papel de alguns componentes descritos na seção 1.1.2. A origem dos dados geralmente é representada pela Internet e apresenta um grande volume de informações. O sensor responsável pela obtenção dos eventos gerados fora do IDS, realizando um pré-processamento nos dados para adequá-los ao analisador de eventos que, geralmente, faz parte de um sensor, e sua função é analisar os dados pré-processados do sensor e classificar em conexão normal ou ataque. Essa estrutura básica também contempla o gerente que vai ser notificado sobre o ataque, o administrador da rede, responsável pelas políticas de segurança e, por sua vez, da configuração do IDS. A origem dos dados, na maioria dos casos, é a Internet. O volume de dados advindos da Internet é grande. Também temos a figura do operador que vai tomar as devidas providências para barrar o ataque eminente na rede.

A parte mais importante de um IDS é o analizador de eventos, pois é ele quem vai classificar as conexões contidas no tráfego como normal ou anormal. Como ele necessita processar um volume grande de informações em um curto espaço de tempo, poderia se pensar em uma implementação em *hardware*, para o mesmo.

1.2.5 Estudo de caso: *SNORT*

O *snort* é um IDS de código aberto, largamente utilizado em empresas. Desenvolvido pela Martin Roesch, executa análise de protocolo, busca/associa padrões de conteúdo e pode ser usado para detectar uma variedade de ataques e probes (ferramentas de varredura da rede), tais como *buffer overflows*, *stealth port scans*, ataques CGI, SMB probes, OS *fingerprinting*, entre outros. Uma característica relevante é a capacidade de gerar alertas em tempo real. Sua utilização é indicada para empresas de pequeno porte.

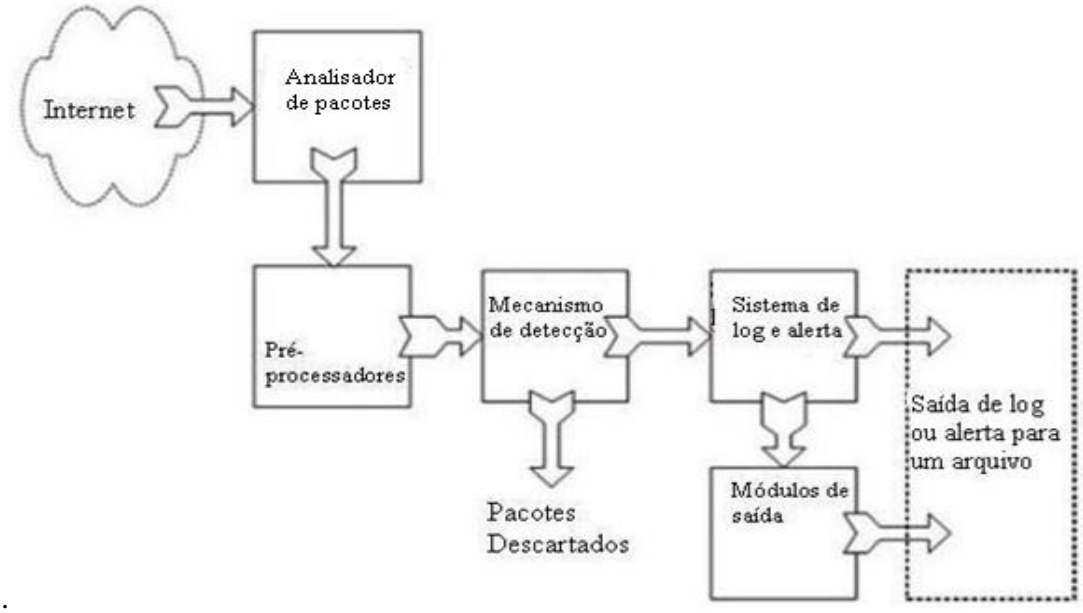
Arquitetura do *snort*

A implementação do *snort* segue uma arquitetura modular, cujo objetivo é melhorar o desempenho na coleta e análise de pacotes. Seus principais subsistemas são:

- **Pré-processamento**: Disposto entre o analisador de pacotes e o processamento do mecanismo de detecção, decodifica o pacote;

- **Detecção**: Ocorre durante o processamento do mecanismo de detecção;
- **Saída**: é executado após o processamento do mecanismo de detecção, para registrar e alertar.

Abaixo (Figura 17) segue a arquitetura do snort.



.

Figura 17. Arquitetura do *snort*.

Capítulo 2

Base de Dados

Neste Capítulo, iremos descrever a base de dados utilizada para o treinamento das técnicas de aprendizagem de máquina a serem comparadas, bem como os ataques contidos nela.

2.1 Descrição

A base de dados utilizada neste trabalho é a KKD *cup* 1999 data [20]. Esta base foi concebida através da simulação de um ambiente de uma rede militar da força aérea dos Estados Unidos. O objetivo da sua concepção era inspecionar e avaliar o estudo de detecção de intrusão, através de pesquisas. A rede foi operada em um ambiente real, sendo alimentada por conexões TCP *dump*, mas foi sendo bombardeada por uma seqüência de múltiplos ataques. Para cada conexão foram extraídas 41 diferentes características, tanto qualitativas quanto quantitativas, formando um banco de dados com aproximadamente cinco milhões de conexões [2, 21, 22].

A conexão é uma seqüência de pacotes TCP, começando e terminando em tempos bem definidos, com fluxos de dados entre um IP de origem e um IP de destino, funcionando em cima de um protocolo bem definido. Cada uma das conexões é rotulada como uma conexão normal ou como um tipo de ataque. Cada conexão gravada tem aproximadamente 100 bytes. Na Tabela 1, temos algumas características inerentes a esta base.

2.2 Tipos de Ataques

A base de dados possui quatro tipos de ataques principais. A seguir vamos descrever cada um deles e mencionar quais são os ataques pertencentes a cada tipo [2, 20, 21].

Tabela 1. Características da base kdd *cup* 1999

Característica	Descrição	Tipo
Duration	Tamanho da conexão (em segundos)	Contínuo
Protocol_Type	Tipo de protocolo	discreto (tcp, udp, ...)
Service	Serviço de rede no destino	discreto (http, telnet ...)
Flag	Status da conexão (normal ou erro)	discreto (normal ou erro)
Src_Bytes	número de bytes dos dados da origem para o destino	Contínuo
Dst_Bytes	número de bytes dos dados do destino para a origem	Contínuo
Land	1 se a conexão é do/para o mesmo host/porta; 0 caso contrário	discreto (0,1)
Wrong_Fragment	número de fragmentos errados	Contínuo
Urgent	número de pacotes urgentes	Contínuo
Hot	números de indicadores chave "hot"	Contínuo
Num_Failed_Logins	número de tentativas com login falhando	Continuo
Logged_In	1 se login com sucesso; 0 caso contrário	discreto (0,1)
Num_Compromised	Números de condições "comprometidas"	Continuo
Root_Shell	1 se obtive root shell; 0 caso contrário	Contínuo
Su_Attempted	1 se superusuário foi tentado; 0 caso contrário	Contínuo
Num_Root	número de acessos como root	Contínuo
Num_File_Creations	número de operações de criação de arquivos	Contínuo
Num_Shells	número de prompts shell	Contínuo
Num_Access_Files	número de operações de controle de acesso em arquivos	Contínuo
Num_Outbound_Cmds	Número limite de comandos em uma sessão ftp	Continuo
Is_Host_Login	1 se o login pertence a uma lista "hot"; 0 caso contrário	discreto (0,1)
Is_Guest_Login	1 se o login é um convidado; 0 caso contrário	discreto (0,1)
Count	número de conexões para um mesmo host como a conexão corrente nos últimos 2 segundo	Continuo
Srv_Count	número de conexões para um mesmo serviço como conexão correntes nos últimos 2 segundos para este serviõ	Contínuo
Serror_Rate	% de conexões que possuem erros "SYN" (bit do cabeçalho TCP utilizado para estabelecer e derrubar uma conexão)	Continuo
Srv_Serror_Rate	% de conexões que possuem erros "SYN" para este serviço	Continuo
Rerror_Rate	% de conexões que possuem erros "REJ" (bit do cabeçalho TCP utilizado para informar que um pacote não chegou)	Contínuo
Srv_Rerror_Rate	% de conexões que possuem erros "REJ" para este serviço	Continuo
Same_Srv_Rate	% de conexões para o mesmo serviço	Continuo
Diff_Srv_Rate	% de conexões para diferentes serviços	Continuo
Srv_Diff_Host_Rate	% de conexões deste mesmo serviço para hosts diferentes.	Contínuo

- DoS (*Denial of Service*): Também chamado de ataque de negação-de-serviço, se caracteriza por deixar um serviço ou rede parada ou muito lento. Há diferentes formas de se lançar um ataque do tipo DoS: abusando de características inerentes aos computadores (por exemplo, respostas ao *ping* do ICMP), identificando falhas de implementações e explorando configurações erradas dos sistemas. Eles podem ser classificados de acordo com os serviços que eles deixam indisponíveis, a Tabela 2 a seguir mostra alguns exemplos de ataques do tipo DoS.

Tabela 2. Ataques DoS

Ataque	Serviço	Mecanismo	Efeito do ataque
Apache2	http	Abuso	Colisões http
Back	http	Abuso/Falha de Implementação	Resposta do servidor fica mais lenta
Land	http	Falha de Implementação	Trava a máquina
Mail Bomb	N/A	Abuso	Aborrecimentos
SYN flood	TCP	Abuso	Negação de serviço para uma ou mais portas
Ping of death	Icmp	Falha de Implementação	Nenhum
Process table	TCP	Abuso	Negação de novos processos
Smurf	Icmp	Abuso	Rede lenta
Syslog	Syslog	Falha de Implementação	Para o *Syslog*
Teardrop	N/A	Falha de Implementação	Reinicia a máquina
Udpstrom	Echo/Chargen	Abuso	Rede lenta

- *Probing*: Nessa classe, os ataques se caracterizam por varrer a rede automaticamente a procura de vulnerabilidades para serem exploradas. Esse tipo de ataque é bastante útil para um intruso que pretenda atacar futuramente, pois através dele é possível criar um mapa da rede contento máquinas e serviços. Geralmente, abusam de alguma característica inerente ao computador .Alguns tipos de *Probing* podem ser vistos na Tabela 3.

Tabela 3. Ataques *Probing*

Ataque	Serviço	Mecanismo	Efeito do ataque
Ipsweep	Icmp	Abuso de característica	Identifica máquinas ativas
Mscan	Many	Abuso de característica	Procura por vulnerabilidades conhecidas
Nmap	Many	Abuso de característica	Identifica portas ativas na máquina
Saint	Many	Abuso de característica	Procura por vulnerabilidades conhecidas
Satan	Many	Abuso de característica	Procura por vulnerabilidades conhecidas

- R2L (*Remote to user attacks*): Chamado de ataque de um usuário remoto (R2L), essa classe se caracteriza pelo envio de pacotes a uma máquina de uma rede, a partir daí são exploradas vulnerabilidades da máquina para ganhar acesso ilegal de usuário local. Alguns ataques R2L estão descritos na Tabela 4.

Tabela 4. Ataques R2L

Ataque	Serviço	Mecanismo	Efeito do ataque
Dictionary	Telnet, rlogin, pop, ftp, imap	Abuso de característica	Ganha acesso de usuário
Ftp-write	Ftp	Configuração errada	Ganha acesso de usuário
Guest	Telnet, rlogin	Configuração errada	Ganha acesso de usuário
Imap	Imap	Falha de implementação	Ganha acesso de usuário
Named	Dns	Falha de implementação	Ganha acesso de usuário
Phf	http	Falha de implementação	Executa comandos como usuário de http
Sendmail	Smtp	Falha de implementação	Executa comandos como administrador
Xlock	Smtp	Configuração errada	Usa Spoof para obter a senha
Xnsoop	Smtp	Configuração errada	Monitora chaves remotamente

- U2Su (*User to root attacks*): Essa classe de ataques se caracteriza por iniciar o ataque como um usuário normal no sistema e explorar vulnerabilidades para ganhar acesso de usuário *root* do sistema. A maioria das explorações dessa classe se dá através de estouro de pilha (*buffers overflows*) que ocorre quando um programa copia muitos dados para um *buffer* estático, sem ter a certeza que os dados se ajustarão. A Tabela 5 mostra alguns tipos desse ataque:

Tabela 5. Ataques U2Su

Ataque	Serviço	Mecanismo	Efeito do ataque
Eject	Sessão do usuário	Estouro de pilha	Ganha acesso de administrador
Ffbconfig	Sessão do usuário	Estouro de pilha	Ganha acesso de administrador
Fdformat	Sessão do usuário	Estouro de pilha	Ganha acesso de administrador
Loadmodule	Sessão do usuário	Falha no carregamento de programas que limpam o ambiente	Ganha acesso de administrador
Perl	Sessão do usuário	Falha no carregamento de programas que limpam o ambiente	Ganha acesso de administrador
Os	Sessão do usuário	Falha no gerenciamento de arquivos temporários	Ganha acesso de administrador
Xterm	Sessão do usuário	Estouro de pilha	Ganha acesso de administrador

O gráfico da Figura 18, mostra a quantidade de cada ataque na base de dados kdd *cup* 1999. A maior parte desses ataques são da classe DOS.

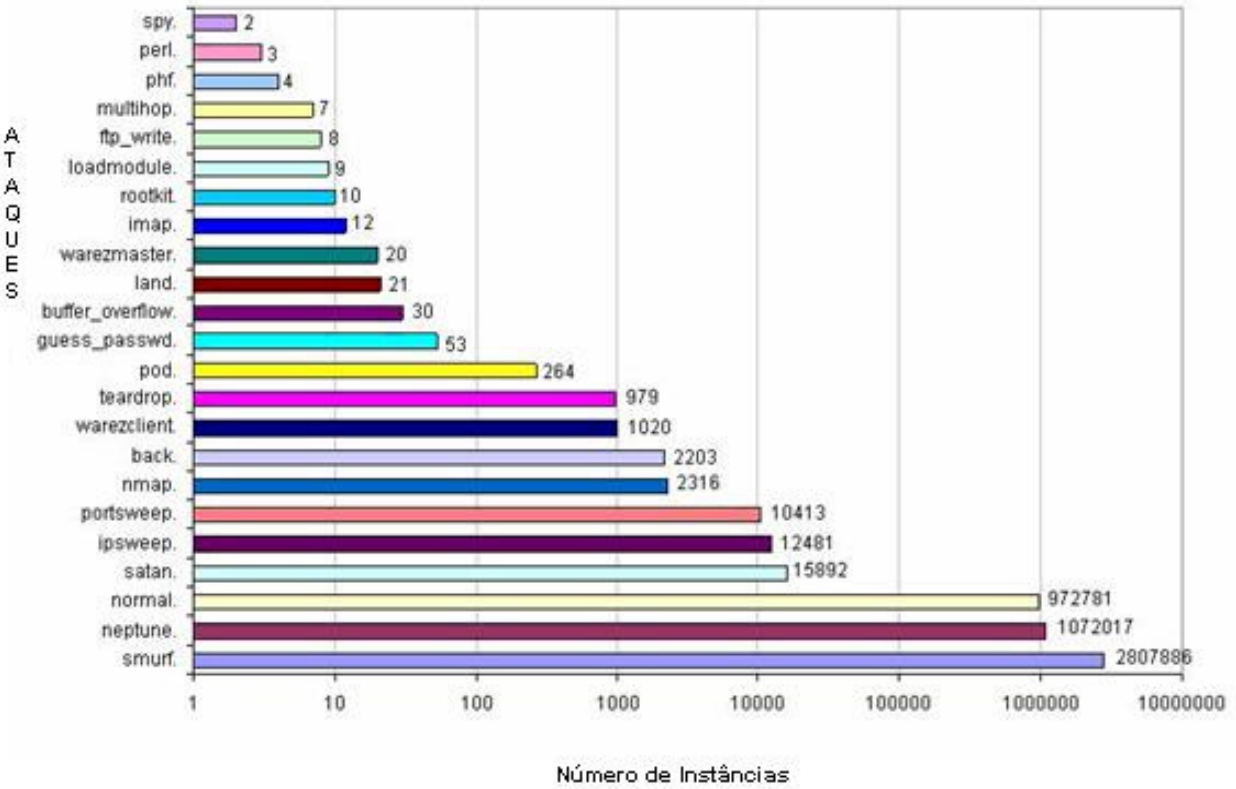


Figura 18. Quantidade de cada ataque na base kdd *cup* 1999 [2].

2.3 Formatação

A base de dados original, como já foi mencionado, possuía 5 milhões de conexões. Existe no mesmo repositório da base original uma base que representa 10% de todas essas conexões e esta base possui 494.021 padrões, com essa quantidade o trabalho ficaria inviável, devido a grande necessidade de processamento e tempo para realizar os experimentos. Alguns trabalhos com essa base de dados [2,21] utilizaram bases de tamanho inferior ao da base 10% da original. Decidimos reduzir a base de dados para 15000 padrões, escolhidos aleatoriamente, e cada tipo de ataque teve sua quantidade baseada na proporcionalidade com essa base que representa 10% da original, excetuando os ataques do tipo U2Su no qual foram inseridos alguns padrões a mais. Ao final dessa divisão, cada classe da base ostentava a seguinte quantidade:

- Normal: 2953 padrões;
- DoS: 11874 padrões;
- Probe: 124 padrões;
- R2L: 38 padrões;
- U2Su: 11 padrões.

Também decidimos subdividir essa base em cinco outras de forma que sempre contivessem os padrões normais, essa separação foi realizada pelo fato de que no mundo real termos conexões normais e ataques fazendo parte do tráfego de entrada numa empresa. Por conseguinte, as bases criadas foram:

- Normal + Ataques: essa base possui os 15000 padrões, porém só duas classes de saída que são a classe normal e a ataques (contém os quatro tipos de ataques);
- Normal + DoS: essa base possui 14827 padrões;
- Normal + Probe: essa base possui 3077 padrões;
- Normal + R2L: essa base possui 2991 padrões;
- Normal + U2Su: essa base possui 2964 padrões;

Após a separação das bases, foram feitas as normalizações necessárias para torná-las utilizáveis pelos respectivos simuladores (ver Capítulo 3). A normalização foi realizada seguindo os seguintes princípios:

- Há na base atributos representados por nomes; para cada atributo desse, foi realizado uma normalização, da seguinte forma: para cada nome contido em um atributo foi criada uma nova entrada. Assim, a base passou a ter 109 entradas e não mais as 41 originais; o valor 1 foi atribuído, se o nome de um determinado atributo existir para aquele padrão, e 0 caso contrário. Para exemplificar esse processo, tínhamos na base de dados um atributo que representava três protocolos (TCP, UDP e ICMP). Para cada protocolo foi criada uma nova entrada, e, se para um determinado padrão, o TCP fosse utilizado, a sua coluna teria o valor 1 e as demais, UDP e ICMP, seriam 0. A representação para o TCP seria então 1 0 0, já o UDP seria 0 1 0 e por sua vez o ICMP 0 0 1;
- Para cada tipo de ataque foi criada uma coluna. Assim a base de dados passou a ter uma saída para cada ataque. As bases explicadas anteriormente passaram a possuir 2 saídas cada;

- Os dados foram normalizados, antes da realização do treinamento, deixando seus valores entre 0 e 1 obedecendo à equação abaixo, lembrando que essa equação é para cada célula da tabela:

$$Xnovo = (Xreal - Xmin) / (Xmax - Xmin)$$

  Onde, Xmax é o valor máximo de um determinado atributo e Xmin é o valor mínimo de um determinado atributo.
- Para cada simulador foi inserido seu respectivo cabeçalho, bem como algumas modificações que foram necessárias;
- Os arquivos foram salvos seguindo a extensão de cada simulador.

Ao todo foram gerados 20 arquivos diferentes, sendo: 5 para MLP, 5 para RBF-DDA, 5 para SVM e 5 para NN e kNN.

Capítulo 3

Experimentos e Resultados

A finalidade deste Capítulo é descrever como foram realizados os experimentos, bem como analisar os resultados obtidos e traçar um estudo comparativo entre as técnicas empregadas de aprendizagem de máquina empregadas, levando em consideração alguns fatores como: tempo de processamento, erro de validação cruzada e complexidade do classificador gerado.

3.1 Ferramentas utilizadas

3.1.1 SNNS (Stuttgart Neural Network Simulator)

O SNNS, simulador utilizado no treinamento de vários tipos de redes neurais, foi desenvolvido em 1989 pelo Instituto Para Sistemas Paralelos e Distribuídos de Alta Desempenho (*Institute For Parallel And Distributed High Performance Systems)* (IPVR) da Universidade de *Stuttgart* na Alemanha [23]. O objetivo da sua criação era prover uma ambiente de simulação eficiente para pesquisa e aplicação de redes neurais artificiais. A versão do SNNS utilizada neste trabalho foi a 4.2 para o Windows.

Basicamente o SNNS é dividido em quatro partes principais [25]:

- Um simulador de *kernel* escrito em C: o *kernel* opera sobre uma representação interna das redes neurais e é responsável por todas as operações sobre as estruturas de dados que a compõe;
- Uma interface gráfica para interação com o usuário: trabalha sobre *kernel*, fornecendo uma representação gráfica para as redes neurais e controlando o *kernel* durante a execução do programa;
- Uma interface para execução em *batch* (*batchman*);
- Um compilador de redes, o snns2c.

A seguir, na Figura 19, temos a interface inicial do SNNS.

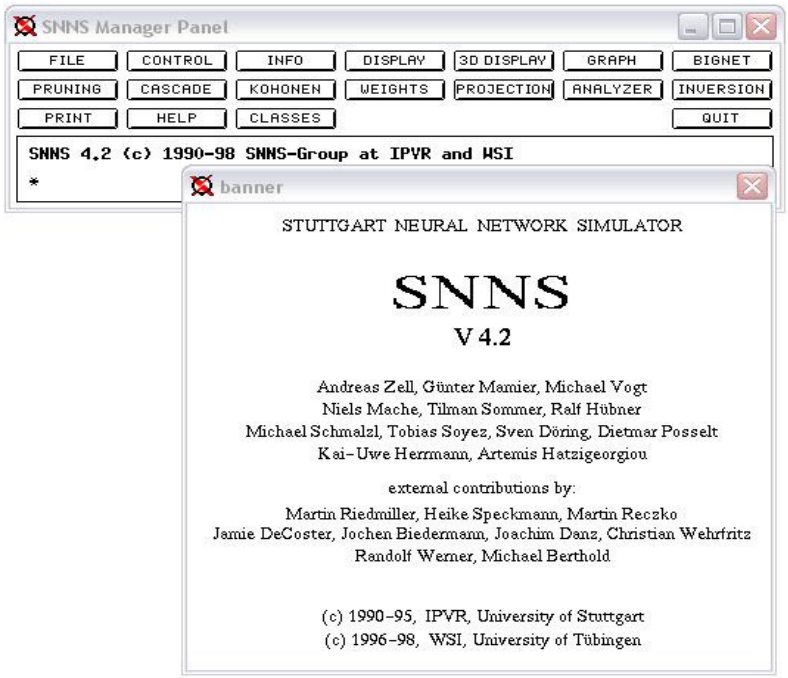


Figura 19. Tela inicial do SNNS.

Utilizando o SNNS para criação de uma rede

O SNNS foi utilizado neste trabalho para treinamento com redes neurais RBF, como já havia sido mencionado anteriormente. O primeiro passo para criação dessas redes no SNNS é ir ao painel inicial (ver Figura 19), pressionar o botão *BIGNET* e escolher a opção *general*. Neste local escolhemos o tipo da rede. Existem outros tipos, porém o utilizado para redes RBF é o geral (*general*). Na Figura 20, visualizamos o formulário para criação da rede. Nele definiremos a quantidade de nodos em cada camada e como as camadas estarão conectadas. A definição do número de nodos é feita no local indicado na Figura 20 pela topologia da rede, para o nosso exemplo particular foram colocados 109 nodos na camada de entrada, nenhum nodo na camada intermediária e 2 nodos na camada de saída. A camada intermediária ficará vazia, pois o algoritmo de treinamento que utilizamos para RBF foi o DDA (ver seção 1.1.1.4) e nele a camada escondida é construída à medida que o treinamento ocorre. Esse algoritmo também define a forma com que os nodos vão se ligar aos nodos da camada seguinte, não sendo necessário fazer a opção por uma rede totalmente conectada (isso seria feito utilizando a opção *full conection*, na Figura). A seguir, após definir tudo isso pressionamos o botão *create net* para criar nossa rede.

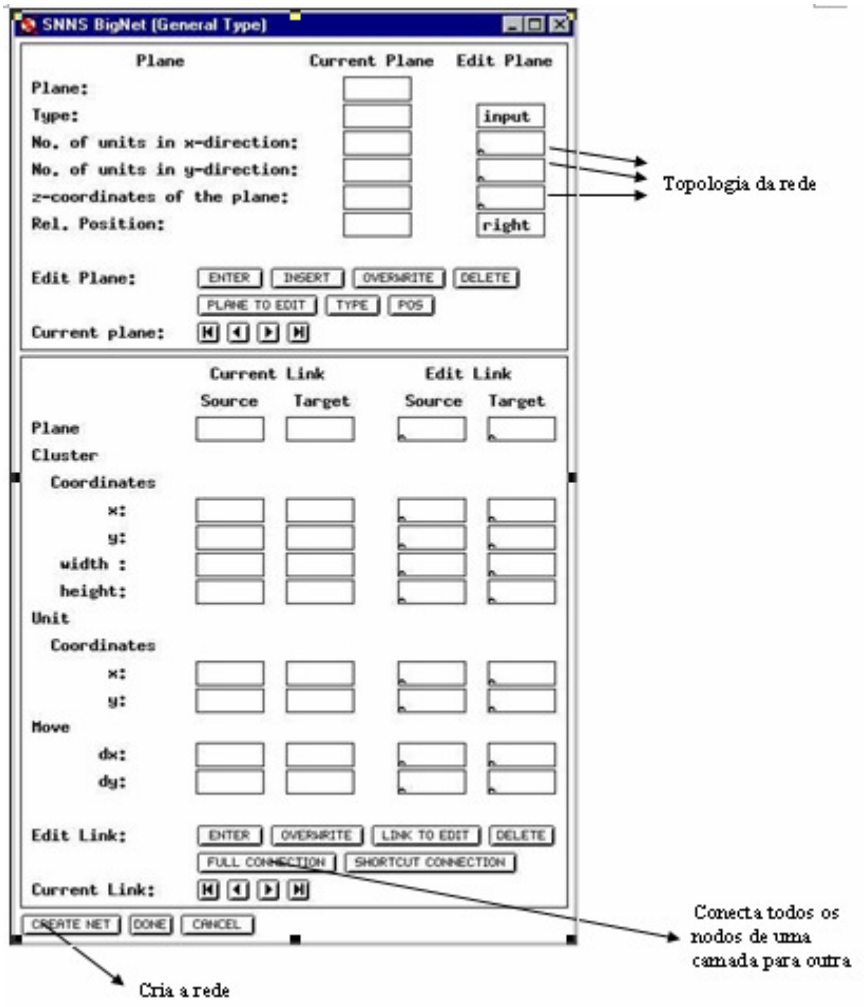


Figura 20. Definição da topologia da rede no SNNS.

Com a rede criada e, conseqüentemente, a topologia definida o último passo a ser realizado é salvar a rede. Para salvar vamos ao painel principal do SNNS e escolhemos a opção *FILE*, a Figura 21 é a tela que será aberta. Como podemos visualizar, no lado direito da tela existem os tipos de arquivos que o SNNS trabalha, os principais são:

- .NET: arquivo usado para definir a rede, contém todas as informações necessárias para a rede neural;
- .PAT: essa extensão é utilizada para definir o arquivo que conterá os padrões, esses arquivos, para funcionarem corretamente no SNNS, requerem um cabeçalho;
- .RES: extensão para os arquivos de resultados que são gerados após o treinamento e teste da rede neural, eles servem para fazer avaliação dos resultados obtidos. Esses arquivos são analizados um de cada vez.

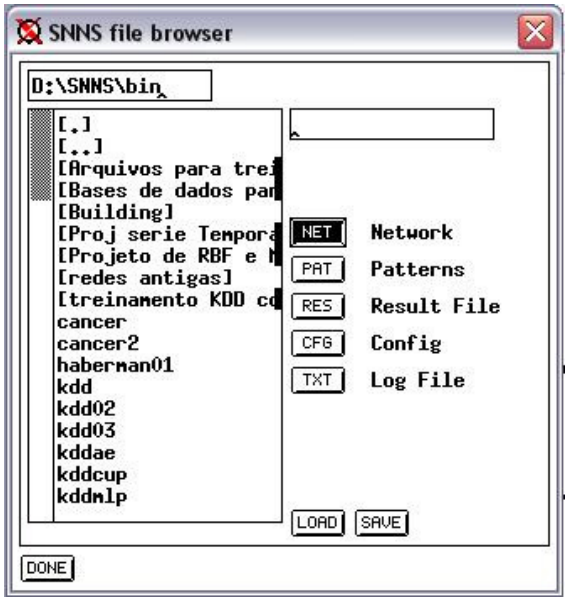


Figura 21. Tela salvar e carregar arquivos do SNNS.

Além de criar e treinar a rede, o SNNS também possui uma ferramenta para análise dos resultados, o *analyze*. Essa ferramenta atua sobre os arquivos de resultados gerados pelo treinamento, informando qual foi o percentual de acerto para o teste de determinada base, quantos padrões foram classificados corretamente, incorretamente e não conhecidos, dentre outros. A Figura 22 ilustra o uso do *analyze.*

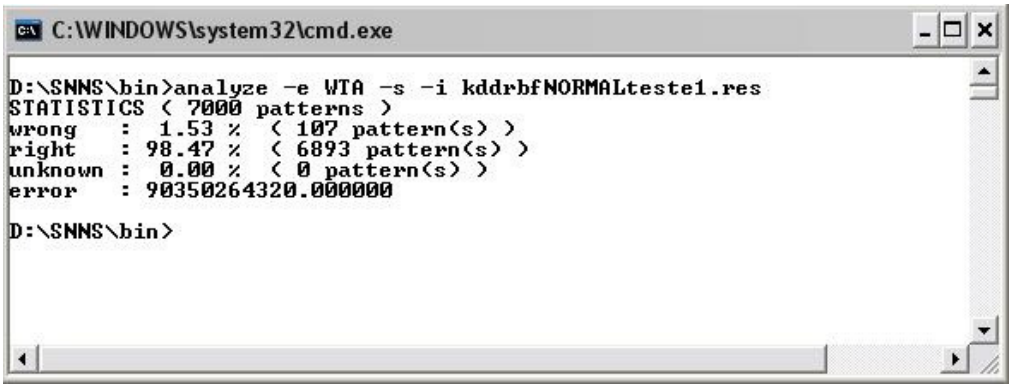


Figura 22. Tela da ferramenta *analyze*.

3.1.2 LIBSVM

LIBSVM [26, 2, 21] é um simulador para classificação, regressão e estimação de distribuição de máquinas de vetor de suporte (SVM). Ele possui uma interface com o usuário simples e de fácil manipulação. Ele contém alguns *softwares* auxiliares como, por exemplo, o *svmscale*, que normaliza base de dados. Na próxima Seção desse Capítulo veremos o conceito de validação cruzada, que foi utilizado no treinamento de todos os métodos presentes neste trabalho. Ao contrário do SNNS, que não implementa a validação cruzada como opção para treinamento, o LIBSVM o faz. Portanto é só inserir um comando e temos a validação cruzada para esse simulador.

O LIBSVM disponibiliza um *applet* para visualizarmos exemplos de classificação e regressão, porém esse *applet* serve apenas para demonstração. A Figura 23 demonstra os resultados da sua execução.

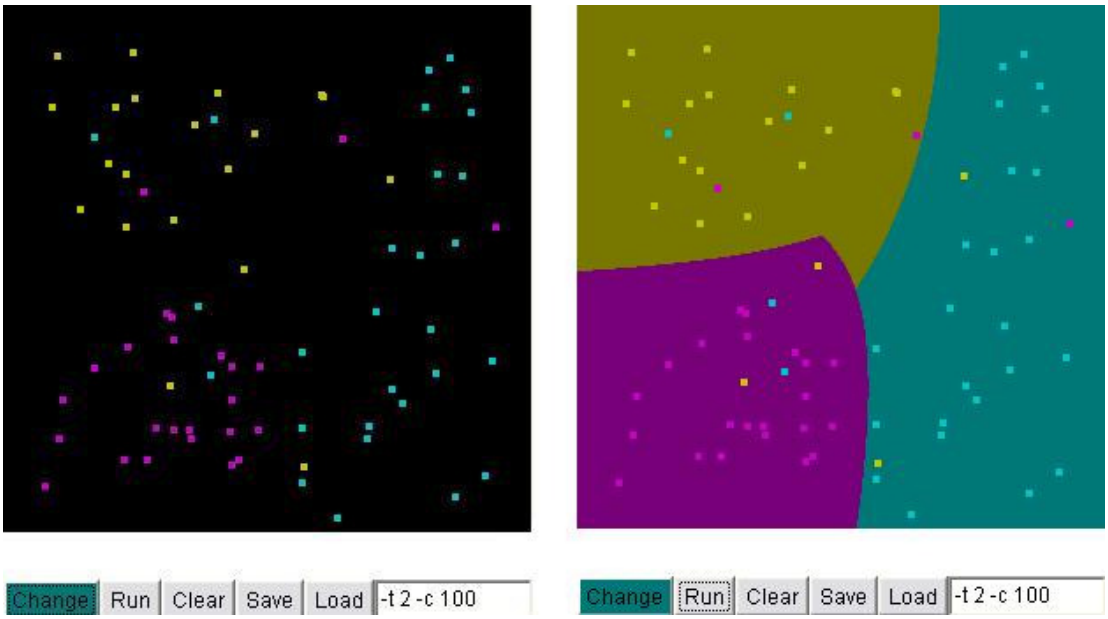


Figura 23. *Applet* do LIBSVM [13].

Como podemos visualizar na Figura 23, temos 3 classes diferenciadas pela cor. Quando executamos o *applet* são traçadas regiões que representam as três classes de forma que a região deva possuir o maior número possível de padrões.

Para treinarmos SVMs em problemas reais, utilizamos o *prompt* de comando. O LIBSVM disponibiliza uma série de aplicativos para treinamento e teste de base de dados; um deles é o *svmtrain* utilizado para realizar treinamentos. A Figura 24 demonstra a utilização desse aplicativo para uma das bases de dados do trabalho.

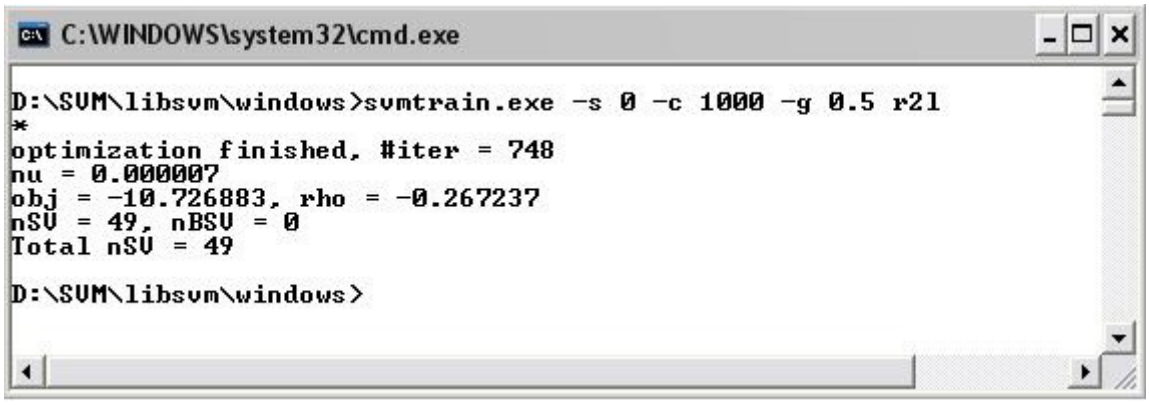


Figura 24. svmtrain[13].

Como podemos visualizar existem alguns parâmetros que são usados para o treinamento. Os parâmetros utilizados neste trabalho foram:

- -s: tipo de SVM, a opção 0 indica que vamos realizar classificação com a base de dados;
- -c : custo, define o parâmetro C do SVM;
- -g : gama, define o parâmetro gama na função do *kernel* RBF.

Neste trabalho, utilizamos um outro parâmetro, denominado número de *folds* de validação cruzada que vai ser explicado na próxima Seção. A Figura 25 apresenta a utilização do treinamento do LIBSVM com validação cruzada. Nela podemos visualizar a inserção de um parâmetro para o treinamento (-v), ele vai indicar que será utilizada a validação cruzada e o número seguinte a ele indica a quantidade de conjuntos que será utilizada.

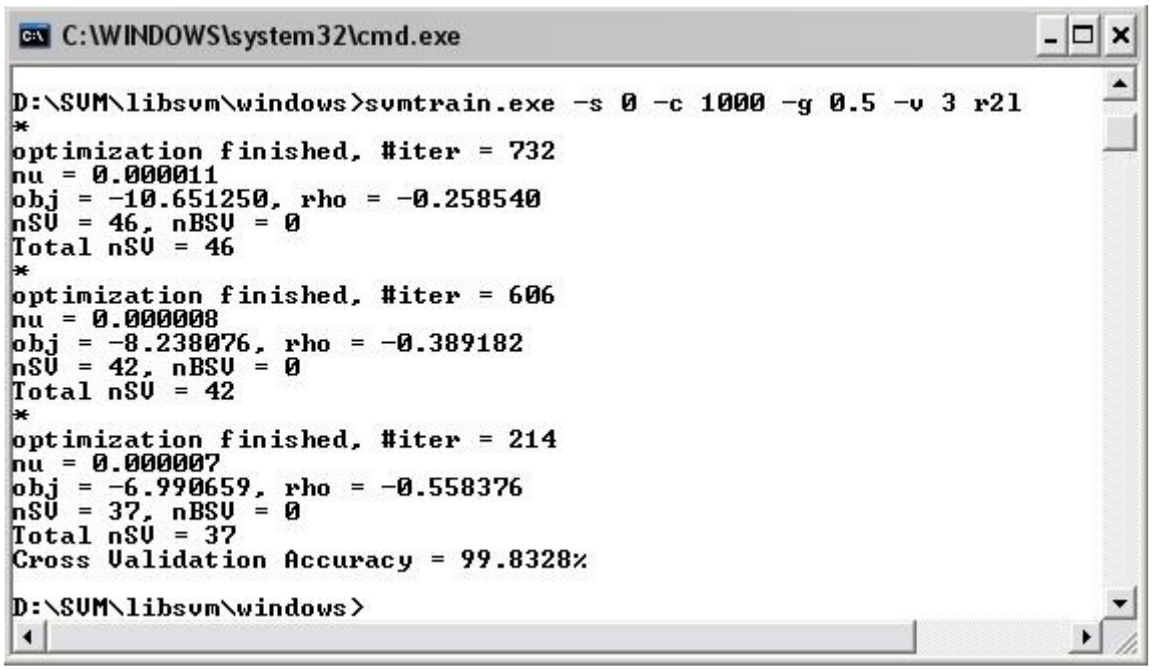


Figura 25. svmtrain com validação cruzada.

Um outro aplicativo importante é o *svmpredict*. Esse aplicativo é utilizado para classificação, porém com uso da validação cruzada, temos os resultados fornecidos pelo *svmpredict* sem precisar executá-lo . A Figura 26 mostra a execução desse aplicativo.

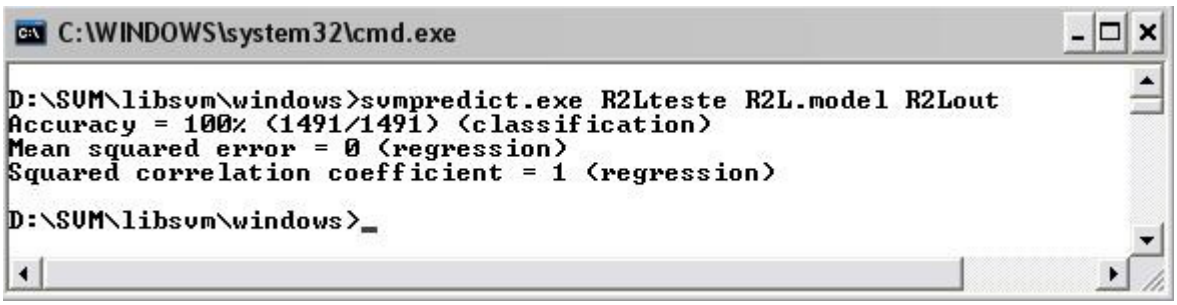


Figura 26. Tela do svmpredict.

3.1.3 WEKA (*Waikato Environment for Knowledge Analysis*)

Simulador desenvolvido na Universidade de Waikato na Nova Zelândia, implementado em Java, possui uma interface gráfica amigável, bem como um formato de arquivo peculiar (.arff) para as bases de dados. O WEKA possui implementações de várias técnicas de aprendizagem de máquina. No nosso trabalho, este simulador foi utilizado para os treinamentos com MLP, NN e kNN. A tela inicial do WEKA pode ser visualizada na Figura 27. A versão desse software utilizada no trabalho foi a 3.4.5 [27].

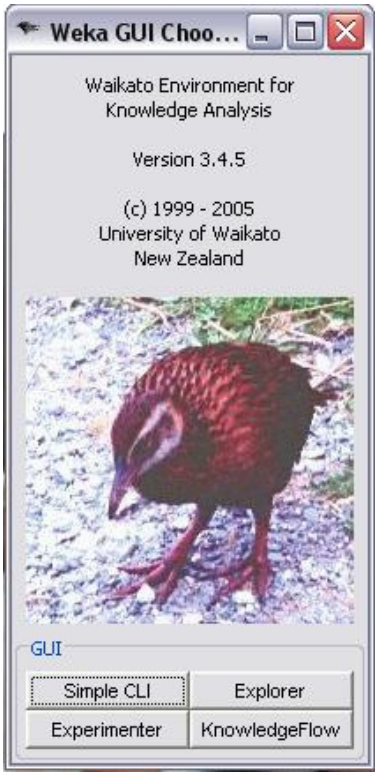


Figura 27. Tela inicial do WEKA [27].

Como podemos observar na Figura 27, a interface gráfica nos oferece 4 opções:

- *Simple CLI* – nesse modulo, utilizamos os comandos de linha (*prompt*) para realizarmos nossos experimentos;
- *Explorer* – serve para pré-processar a base de dados e realizar os experimentos através de uma interface gráfica;
- *Experimenter* – compara diferentes técnicas de aprendizagem de máquina tanto na classificação como na regressão;
- *KnowledgeFlow* – uma nova interface gráfica para o WEKA.

Dessas opções oferecidas, a única utilizada foi a *Explorer*. Primeiramente, tivemos que realizar um pré-processamento na base de dados para adequá-la ao padrão do WEKA. Um exemplo desse arquivo será mostrado em detalhes no apêndice A. A base de dados possui um cabeçalho peculiar e será gravada com a extensão .arff. A Figura 28 apresenta a página do WEKA onde carregamos a base de dados e temos a possibilidade de fazer as normalizações.

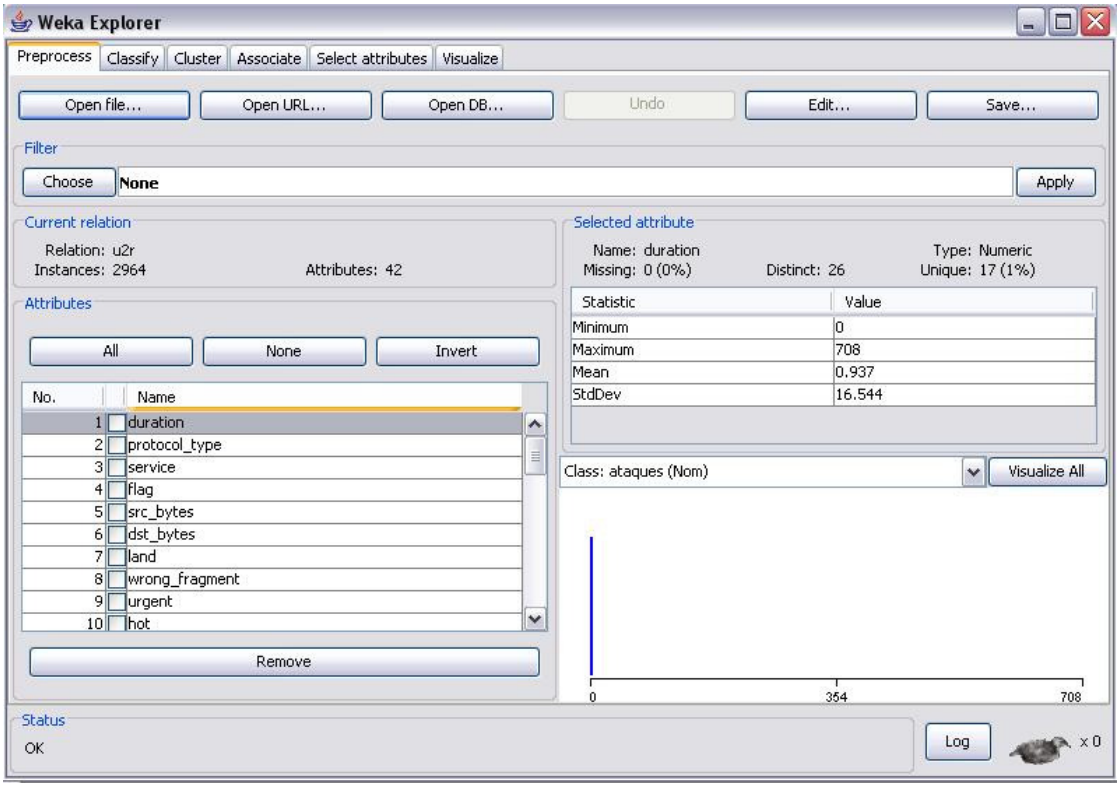


Figura 28. Abrir arquivo para treinamento no WEKA.

A opção *filter*, como mostrado na Figura 28, é onde será realizada a escolha da forma como queremos normalizar a base, bem como realizar também outros ajustes.

Para efeito de classificação temos que escolher a aba *classify*. Nela, podemos optar pela técnica que utilizaremos bem como inserir os parâmetros referentes a cada uma delas antes do treinamento.

Para escolher o algoritmo de treinamento vamos até *classifier*, após pressionarmos o botão *choose* podemos visualizar na Figura 29, as opções de técnicas disponíveis no WEKA. Para inserir os parâmetros necessários para o treinamento, clicamos com o botão direito do *mouse* no nome da técnica de treinamento. Em *test options* escolhemos a técnica de validação cruzada, que vai ser explicada na próxima Seção.

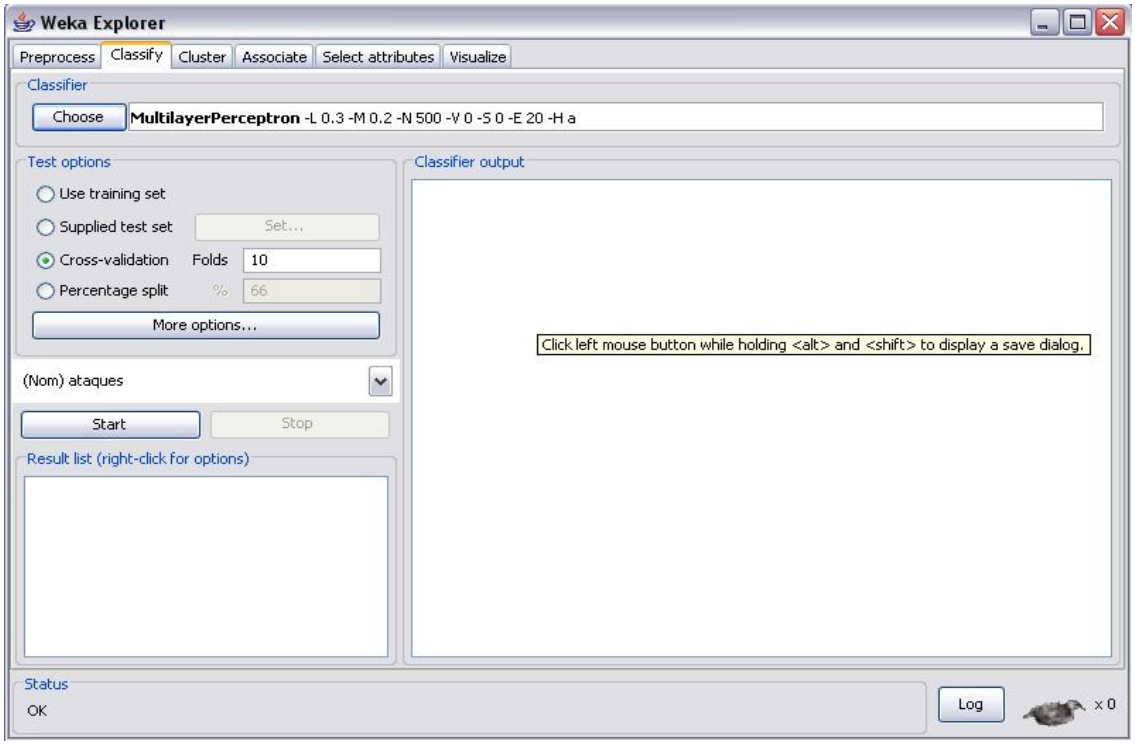


Figura 29. Escolha da técnica de aprendizagem de máquina no WEKA.

Como podemos notar na Figura 30. A WEKA nos oferece uma variedade de técnicas de aprendizagem de máquina para utilizarmos. Neste trabalho, utilizamos as técnicas M*ultilayerPerceptron* presentes nos conjunto de técnicas *functions* e IB1 (NN) e IBK (kNN) presentes no conjunto *lazy*.

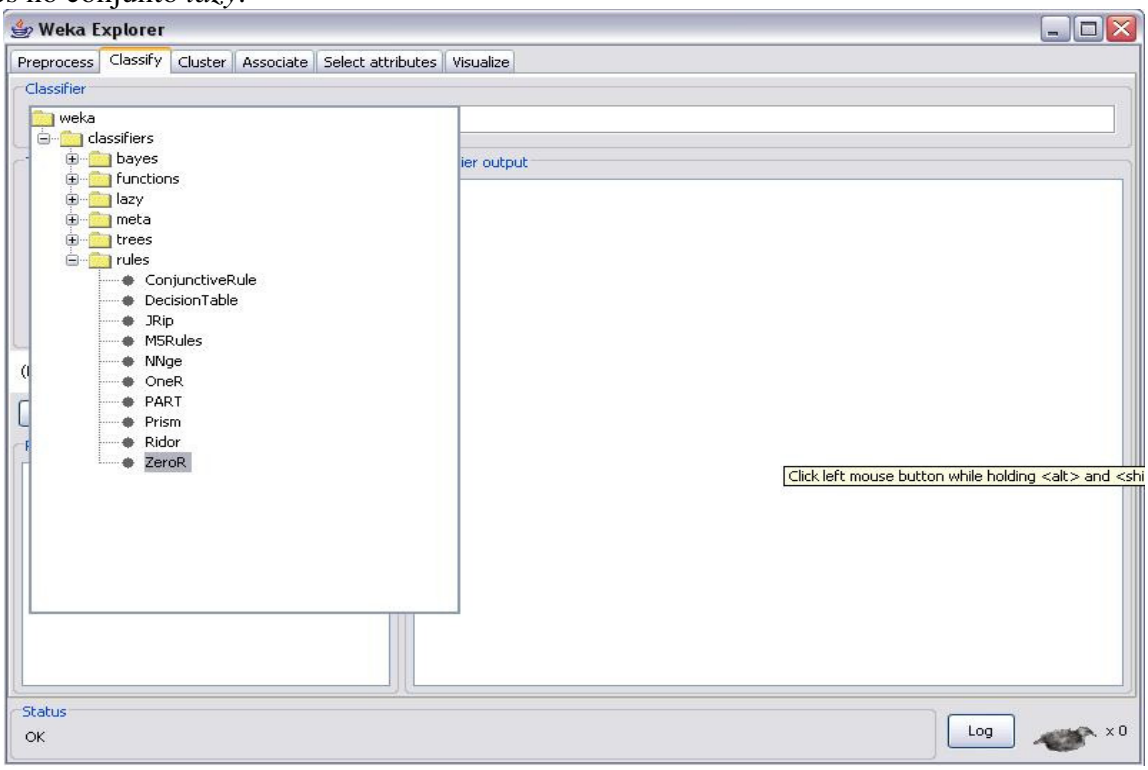


Figura 30. Visualização das técnicas de aprendizagem do WEKA.

3.2 Validação Cruzada

Validação cruzada [40] é uma técnica que propicia estimar a capacidade de generalização de um classificador. Essa técnica consiste em dividir o conjunto de treinamento em K-partes aproximadamente iguais. Uma dessas partes será o subconjunto a ser utilizado para teste. A cada execução do experimento esse conjunto vai mudando.

Para poder exemplificar a técnica da validação cruzada utilizada em todas as técnicas de aprendizagem de máquina, imagine uma base de dados com 3.000 padrões; vamos desenvolver o treinamento dessa base utilizando validação cruzada com 3 subconjuntos. Na primeira execução, foram divididos os conjuntos, ficando o primeiro com os padrões de 1 a 1000, o segundo com os padrões de 1001 a 2000 e o terceiro de 2001 a 3000. Também foi definido pela validação cruzada que o conjunto de 1 a 1000 seria utilizado para os testes na primeira execução e a concatenação dos outros dois seria utilizada para treinamento. Na segunda rodada, foi utilizado para teste o segundo conjunto e para treinamento a concatenação do primeiro e terceiro conjunto. Na terceira e última execução, foi utilizado para teste o terceiro conjunto e para treinamento a concatenação do primeiro e segundo conjunto.

Uma vantagem da utilização desse método é que ele utiliza a base de dados em sua totalidade, gerando um resultado mais confiável.

O erro médio da validação cruzada é calculado realizando a média aritmética dos erros fornecidos por cada conjunto de testes.

3.3 Experimentos utilizando Redes Neurais Artificiais

3.3.1 *MultiLayer Perceptron* (MLP)

Para os treinamentos utilizando redes MLP, foram utilizadas redes com duas camadas escondidas. Essa decisão foi baseada em alguns trabalhos anteriores [2, 21] e na complexidade do problema. Essas camadas utilizaram, como função da ativação, a sigmóide logística. A conexão entre as camadas foi total, ou seja, um nodo da camada anterior se liga a todos da camada posterior. O treinamento ocorreu com uma taxa de aprendizagem fixa em 0,01. Foi estabelecido um limite de épocas para treinamento em 500 (esse valor foi pensado de acordo com outros trabalhos relacionados) e utilizando um conjunto de validação que equivale a 25% da base de dados. O número de neurônios nas camadas escondidas, no primeiro treinamento, foi de 20; para o segundo treinamento, foram utilizados 40 neurônios em cada camada.

A base de dados foi adequada ao padrão dos arquivos .arff, do WEKA. Para isso, foi criado um arquivo contendo todos os padrões de cada uma das cinco bases de dados. Inicialmente, havia 41 atributos de entrada, porém com a normalização efetuada pelo WEKA a base de dados passou a ter 109 entradas. Em cada base o número de saídas é igual a 2.

O treinamento transcorreu com a utilização do algoritmo *backpropagation*, a cada execução, a base foi treinada e testada 10 vezes, devido ao uso da validação cruzada. A Tabela 6 apresenta um resumo do que foi utilizado para o treinamento com MLP.

Tabela 6. Parâmetros MLP.

Camadas Escondidas	2
Número de Neurônios Escondidos (em cada camada)	20 (1ºexperimento) 40 (2º experimento)
Taxa de aprendizagem	0,01
Conjunto de validação	25%
Épocas máxima de Treinamento	500
Algoritmo de treinamento	*Backpropagation*

3.3.2 *Radial Basis Function* (RBF)

O treinamento com redes RBF foi realizado com o SNNS. Essas redes utilizam como função de ativação da camada intermediária a função de base radial Gaussiana. A base de dados foi dividida em três partes: treinamento (50%), validação (25%) e teste (25%).

No treinamento com RBF foram criadas redes que só possuíam nodos na camada de entrada e saída, para ser mais preciso, 109 nodos na entrada e 2 na saída para cada base. Isso se deve a utilização do algoritmo de treinamento DDA. Esse algoritmo é construtivo, ou seja, à medida que o treinamento ocorre, a camada intermediária vai sendo criada de acordo com a necessidade. O DDA possui dois parâmetros que são o limiar positivo θ+ e o limiar negativo θ-.

Neste trabalho, optamos por utilizar a técnica de seleção do limiar negativo θ-, pois ela tem apresentado bons resultados em vários tipos de problemas, tendo o erro de teste, sem o θ- padrão, diminuído para a maioria dos problemas [7, 8, 9]. Tanto o θ+ quanto o θ- possuem um valor padrão, 0,4 e 0,1, respectivamente. Através da técnica de seleção do θ-, podemos obter melhores resultados, variando o valor do θ-; isso foi realizado neste trabalho. Os valores utilizados foram: 0,2, 0,1, 1e-2, 1e-3,1e-4, 1e-5, 1e-6, 1e-7 e 1e-8, esses valores foram extraídos de outra pesquisas a respeito dessa técnica [7, 8, 9]. A diminuição do θ- acarreta um aumento da complexidade da rede, pois com a diminuição desse do valor, esse parâmetro faz com que mais protótipos (nodos) sejam necessários. A Tabela 7 descreve os parâmetros gerais para os treinamento das redes RBF-DDA.

Tabela 7. Parâmetros RBF.

Camadas Escondidas	1
θ+	0.4
θ-	0,2; 0,1; 1e-2; 1e-3; 1e-4; 1e-5; 1e-6; 1e-7 e 1e-8
Conjunto de validação	25%
Conjunto de teste	25%
Conjunto de treinamento	50%
Algoritmo de treinamento	DDA

3.3.3 Resultados obtidos pelas redes neurais

Os resultados obtidos com MLP podem ser visualizados na Tabela 8. Nela podemos visualizar que o tempo de processamento com a validação cruzada, ou seja, o treinamento completo, para esse tipo de rede neural é alto, por exemplo, a base Normal + Ataques tem um tempo de processamento superior à sete horas.

Tabela 8. Resultados do treinamento com MLP.

Treinamento com MLP utilizando validação cruzada (10-fold)				
	Número de Neurônios na Camada Escondida		Tempo de Processamento	Erro de Validação Cruzada (%)
	1ª Camada	2ª Camada		
Normal + Ataques	20	20	07:36:00	0,1667
Padrões: 15000	40	40	25:25:00	0,1667
Normal + Probe	20	20	01:51:00	0,1625
Padrões: 3077	40	40	02:56:00	0,1625
Normal + DOS	20	20	06:26:00	0,0067
Padrões: 14827	40	40	14:04:00	0,0067
Normal + U2Su	20	20	00:53:00	0,3711
Padrões: 2964	40	40	01:30:00	0,3711
Normal + R2L	20	20	01:39:00	1,2705
Padrões: 2991	40	40	03:10:00	1,2705

Além dos resultados na forma de tabela, com os dados colhidos durante o treinamento e teste dessas redes, como utilizamos o simulador WEKA para realização dessa execução, ele nos fornece um dado interessante que são as matrizes de confusão. Através da matriz de confusão poderemos ter a idéia de quantos falsos negativos e quantos falsos positivos existem. Abaixo segue as matrizes de confusão para o treinamento com MLP. Tomando como exemplo a matriz da letra a), essa matriz está nos informando que 14 padrões normais foram classificados como ataque e 11 padrões de ataques foram classificados como normal.

a)
```
    a      b   <--   classificado como
 2939     14    |    a = normal
   11  12036    |    b = ataque
```

b)
```
    a      b   <--   classificado como
 2953      0    |    a = normal
    3    121    |    b = probe
```

c)
```
    a      b   <--   classificado como
 2953      0    |    a = normal
    1  11872    |    b = dos
```

d)
```
    a      b   <--   classificado como
 2953      0    |    a = normal
   11      0    |    b = u2su
```

e)
```
    a      b   <--   classificado como
 2952      1    |    a = normal
   38      0    |    b = r2l
```

Na Tabela 9, podemos acompanhar os resultados obtidos com o uso de RBF-DDA. Nela estão sendo mostrados apenas os resultados com o θ- padrão e com o melhor resultado obtido com a seleção desse parâmetro. Podemos visualizar que com a seleção do θ- temos um aumento do número de nodos na camada intermediária e, conseqüentemente, um aumento no tempo de processamento. Em contra partida, os melhores resultados são com o θ- diferente do padrão. A tabela completa, com todos os experimentos com RBF-DDA, pode ser vista no apêndice B.

Tabela 9. Resultados do treinamento com RBF.

Treinamento com RBF-DDA utilizando validação cruzada (10-fold)	Theta-	Nº Neurônios na Camada Escondida	Tempo de Processamento	Erro de Validação Cruzada	Ciclos
Normal + Ataques	0,1	113	00:16:03	1,57%	4
Padrões: 15000	1,00E-05	287	01:03:22	0,05%	3
Normal + Probe	0,1	39	00:00:34	0,75%	3
Padrões: 3077	1,00E-06	87	00:01:26	0,16%	4
Normal + DOS	0,1	76	00:04:20	1,56%	4
Padrões: 14827	1,00E-06	245	00:51:03	0,03%	3
Normal + U2Su	0,1	24	00:00:27	0,37%	4
Padrões: 2964	1,00E-04	49	00:00:43	0,14%	4
Normal + R2L	0,1	27	00:00:30	1,27%	3
Padrões: 2991	1,00E-04	50	00:00:46	0,10%	3

Após visualizarmos as tabelas contendo os melhores resultados de cada uma das técnicas de redes neurais, podemos destacar que os resultados obtidos pelas redes RBF-DDA, são superiores aos obtidos pela redes MLP. A Tabela 10 apresenta a comparação entre as duas técnicas, com suas respectivas redes, que obtiveram os melhores resultados. Nessa comparação, podemos notar que RBF-DDA apresenta os melhores resultados, visto que os tempos de processamento são equivalentes apenas para as rede de maior porte (Normal e DOS). Nos demais casos, RBF apresenta um tempo menor de processamento bem como um erro para a validação cruzada menor. Por exemplo, na base de dados Normal + Ataques, o erro de validação cruzada da rede RBF-DDA é de apenas 0,05% enquanto que o da MLP é de 0,1667%. Nas bases Normal + Probe, Normal + R2L e Normal + U2Su o tempo de processamento da MLP é superior ao da RBF-DDA, enquanto que na base Normal + R2L o tempo de processamento é igual a 1 hora e 39 minutos para MLP, com RBF-DDA esse tempo é de apenas 46 segundos.

Tabela 10. Comparação dos treinamentos de RBF e MLP.

RBF-DDA X MLP	Técnica	Theta-	Nº Neurônios na Camada Escodida	Tempo de Processamento	Erro de Validação Cruzada (%)	Ciclos
Normal + Ataques	MLP	-	20	07:36:00	0,1667	-
Padrões: 15000	RBF-DDA	1,00E-05	287	01:03:22	0,05	3
Normal + Probe	MLP	-	20	01:51:00	0,1625	-
Padrões: 3077	RBF-DDA	1,00E-06	87	00:01:26	0,16	4
Normal + DOS	MLP	-	20	06:26:00	0,0067	-
Padrões: 14827	RBF-DDA	1,00E-06	245	00:51:03	0,03	3
Normal + U2Su	MLP	-	20	00:53:00	0,3711	-
Padrões: 2964	RBF-DDA	1,00E-04	49	00:00:43	0,14	4
Normal + R2L	MLP	-	20	01:39:00	1,2705	-
Padrões: 2991	RBF-DDA	1,00E-04	50	00:00:46	0,10	3

3.4 Experimentos utilizando Máquinas de Vetor Suporte

Os treinamentos utilizando SVM foram realizados de três formas distintas. Na primeira a base de dados foi treinada sem validação cruzada, na segunda a rede foi treinada com uma validação cruzada de 5 *folds*, a terceira utilizou a validação cruzada de 10 *folds* e para efeito deste trabalho só vamos considerar essa terceira forma, pois todas as outras técnicas se utilizaram da validação cruzada de 10 *folds*. Vamos demonstrar também os resultados alcançados pelas outras duas formas no apêndice C. O simulador utilizado para o treinamento foi o LIBSVM [26]. SVM utiliza alguns parâmetros para o seu treinamento. Neste trabalho, utilizamos os seguintes parâmetros: c, γ, s e v. Ficou decidido que esse valores seriam s = 0, c = 1000, γ = 0,5 e v = 10, seguindo trabalhos anteriores [21]. A seguir, a Tabela 11 mostra o resumo dos parâmetros utilizados para SVM.

Tabela 11. Parâmetros SVM.

s	0
c	1000
γ	0,5
v	10

3.4.1 Resultados obtidos por máquinas de vetor suporte

A Tabela 12 demonstra os resultados obtidos com os treinamentos, utilizando SVM com validação cruzada de 10 *folds*. Para visualizar os outros treinamentos consulte o apêndice C.

Tabela 12. Resultados do treinamento com SVM 10-*fold*.

Treinamento com SVM	Vetores Suporte	Tempo de Processamento	Erro de Validação Cruzada (%)	Ciclos
Normal + Ataques	166	00:00:30	0,02	8347
Normal + Probe	71	00:00:03	0,0975	457
Normal + DOS	123	00:00:30	0	7525
Normal + U2Su	37	00:00:02	0,135	286
Normal + R2L	46	00:00:02	0,1672	440

Como podemos acompanhar, o tempo de processamento que SVM requer é baixo, bem como seu erro de validação cruzada também apresenta baixas taxas de erro, se comparadas com as outras técnicas. Apesar de tudo isso, o número de iterações que tem que ser realizadas aumenta um pouco, em relação ao das outras técnicas. Podemos conferir na tabela do apêndice C que se utilizarmos a validação cruzada, aumentaremos conseqüentemente o número de vetores de suporte da camada intermediária.

No apêndice C, podemos conferir que a utilização da validação cruzada de 10-*folds* obtém melhores resultados que os outros dois métodos citados anteriormente. Isso se deve ao fracionamento dos padrões, podendo esses terem uma maior abrangência, pois teremos 10 conjuntos de treinamento diferentes e 10 conjuntos para teste, fazendo um uso mais racional da base.

3.5 Experimentos utilizando técnica dos vizinhos mais próximo (NN e kNN)

O treinamento utilizando a técnica do vizinho mais próximo foi realizado através do simulador WEKA [27]. No treinamento utilizando NN, não existe nenhuma seleção de parâmetro. Para o treinamento do kNN foram realizados experimentos contendo o número de vizinhos igual a 1, 3 e 5. A validação cruzada 10-*fold* também foi utilizada nesses experimentos.

3.5.1 Resultados obtidos pelas técnicas do vizinho mais próximo

Os resultados obtidos com o treinamento da técnica do vizinho mais próximo podem ser visualizados na Tabela 13, para o NN e na Tabela 14, para o kNN. Nela, podemos inferir que, para o kNN, sempre o melhor resultado foi com o K=1, com exceção da base R2L, onde K=1 e K=3 obtiveram o mesmo erro de validação cruzada, 0,0669. Porém com K=3 o tempo de processamento foi menor. O problema do treinamento com este tipo de rede é o fato de que no treinamento todos os padrões são armazenados, isso faz com que essa técnica consuma muitos recursos do computador, mais especificamente recursos de armazenamento. Porém, a sua vantagem são os resultados obtidos. Os erros de validação cruzada são baixos, entretando não tão baixos quanto os de RBF-DDA e SVM. Outra vantagem é o fato de possuir nenhum parâmetro, no caso do NN, e poucos parâmetros, no caso do kNN.

Tabela 13. Resultados do treinamento com NN 10-*folds*.

Treinamento com NN	Tempo de Processamento	Erro de Validação Cruzada (%)	Padrões Treinamento
Normal + Ataques	00:28:52	0,0867	15000
Normal + Probe	00:01:24	0,0975	3077
Normal + DOS	00:26:22	0,0000	14827
Normal + U2Su	00:01:15	0,0337	2964
Normal + R2L	00:01:10	0,0669	2991

Na Tabela 14 podemos notar que o aumento do número de vizinhos, para estas bases, implica um erro de validação cruzada maior.

Tabela 14. Resultados do treinamento com kNN 10-*fold*.

Treinamento com kNN	Tempo de Processamento	Erro de Validação Cruzada (%)
Normal + Ataques		
Padrões: 15000		
k=1	01:24:00	0,0867
k=3	01:12:00	0,1067
k=5	01:08:00	0,1533
Normal + Probe		
Padrões: 3077		
k=1	00:01:28	0,0975
k=3	00:01:26	0,2275
k=5	00:01:22	0,1625
Normal + DOS		
Padrões: 14827		
k=1	01:16:00	0,0000
k=3	01:17:00	0,0135
k=5	01:18:00	0,0135
Normal + U2Su		
Padrões: 2964		
k=1	00:01:15	0,0337
k=3	00:01:23	0,1350
k=5	00:01:25	0,1687
Normal + R2L		
Padrões: 2991		
k=1	00:01:26	0,0669
k=3	00:01:21	0,0669
k=5	00:01:23	0,1003

3.6 Comparação dos resultados RNAs x SVM x NN x kNN

Na Tabela 15, podemos visualizar os melhores resultados para cada técnica de aprendizagem de máquina utilizada neste trabalho. Observando a tabela podemos tirar algumas conclusões. A técnica que necessita de menor tempo de processamento é SVM, além disso, máquinas de vetor suporte podem trabalhar, sem muitos problemas, com bases de maior tamanho enquanto que redes neurais demorariam muito no treinamento dessas bases [2]. NN e kNN por sua vez iriam exigir um espaço para armazenamento grande, além de precisar também de um tempo de processamento mais extenso. Em termos de armazenamento de unidades de processamento, MLP só preciou de 40 unidades (20 + 20) para realizar seus treinamentos, por exemplo, na base de dados Normal + Ataques. Para a mesma base, SVM precisou armazenar 166 unidades enquanto que NN e kNN precisam de todos os padrões de treinamento, 13500 unidades, RBF-DDA precisou de 287 unidades de processamento.

Máquinas de vetor de suporte obtiveram os melhores resultados quanto ao erro de validação cruzada, em uma das bases de dados, Normal + Ataques (0,02%). Já RBF-DDA obteve os melhores resultados para 3 base de dados que foram: Normal + Probe (0,0016%), Normal + U2Su (0,0014) e Normal + R2L (0,001). Esse fato pode ser explicado, pois essas três bases possuem muitos padrões normais e poucos padrões dos seus respectivos ataques. Redes RBF são muito boas em rejeitar padrões discrepantes, ou seja, ela possui, por exemplo, uma capacidade de classificação maior do que MLP para esses padrões, além dessa característica, o que também contribuiu foi a seleção do parâmetro θ-, pois todos esses bons resultados obtidos com RBF tiveram o θ- diferente do padrão (0,1). Outra vantagem de RBF-DDA foi o número de ciclos, épocas de treinamento, reduzido em média para 3 ou 4, enquanto que SVM chegou a ter 8347 ciclos na base de dados Normal + Ataques. Analisamos 4 das 5 bases de dados; a única restante foi a Normal + DOS, onde os melhores resultados foram obtido através das técnicas SVM, NN e kNN (0%). DOS é a classe de ataques com o maior número, como foi descrito no Capítulo 2. Essas técnicas conseguiram não ter falsos positivos e falsos negativos para esse tipo de ataque. Mas isso não garante que, se os outros tipos de ataques tivessem uma quantidade grande de padrões, seus resultados também seriam da ordem de 0%, pois cada ataque tem caracteríscas diferentes, o que facilitaria ou não sua identificação utilizando essas técnicas. O que pesa contra NN e kNN e o fato de que para conseguir esses resultados todos os padrões tiveram que ser armazenados, aproximadamente 13500 padrões, enquanto que SVM precisou de apenas 123 unidades de processamento para obter o mesmo resultado.

O maior tempo de processamento foi exigido pelas redes MLP, já SVM foi a técnica que obteve seus resultados com o menor tempo, porém não foram utilizados muitos parâmetros para o treinamento com SVM; isso faria com seus tempo aumenta-se.

Apesar da utilização de ferramentas de simulação distintas, os resultados obtidos por este trabalho podem ser garantidos, pois essese simuladores são ferramentas já consagradas no meio acadêmico.

Tabela 15. Comparação dos resultados

Comparação dos melhores Resultados (RBF X MLP X SVM X NN X kNN)				
Base de Dados	Nº Unidades de processamento	Tempo de Processamento	Erro de Validação Cruzada (%)	Ciclos
Normal + Ataques				
Padrões: 15000				
RBF	287	01:03:22	0,05 (theta- = 1,00E-5)	3
MLP	-	01:03:00	0,1003	500
SVM	166	00:00:01	0,0200	8347
NN	Todos os padrões do treinamento	00:28:52	0,0867	-
KNN	Todos os padrões do treinamento	01:24:00	0,0867 (K=1)	-
Normal + Probe				
Padrões: 3077				
RBF	87	00:01:26	0,0016 (theta- = 1,00E-6)	4
MLP	-	00:13:00	0,0975	500
SVM	71	00:00:01	0,0975	457
NN	Todos os padrões do treinamento	00:26:22	0,0975	-
KNN	Todos os padrões do treinamento	00:01:28	0,0975 (K=1)	-
Normal + DOS				
Padrões: 14827				
RBF	245	00:51:03	0,03 (theta- = 1,00E-6)	3
MLP	-	00:50:00	0,4587	500
SVM	123	00:00:01	0,0000	7525
NN	Todos os padrões do treinamento	00:26:22	0,0000	-
KNN	Todos os padrões do treinamento	01:16:00	0,0000 (k=1)	-
Normal + U2Su				
Padrões: 2964				
RBF	49	00:00:43	0,0014	4
MLP	-	00:11:00	0,3711 (theta- = 1,00E-4)	500
SVM	37	00:00:01	0,1350	286
NN	Todos os padrões do treinamento	00:01:15	0,0337	-
KNN	Todos os padrões do treinamento	00:01:15	0,0337 (k=1)	-
Normal + R2L				
Padrões: 2991				
RBF	50	00:00:46	0,001 (theta- = 1,00E-4)	3
MLP	-	00:12:00	0,1003	500
SVM	46	00:00:01	0,1672	440
NN	Todos os padrões do treinamento	00:01:10	0,0669	-
KNN	Todos os padrões do treinamento	00:01:21	0,0669 (k=3)	-

Conclusões e Trabalhos Futuros

Este trabalho teve como foco principal realizar um estudo comparativo entre algumas técnicas de aprendizagem de máquina para sistemas de detecção de Intrusão. As técnicas utilizadas neste estudo foram: RNAs do tipo MLP e RBF-DDA, SVM, NN e kNN. Para algumas técnicas foram variados seus parâmetros para obtenção de um melhor desempenho.

Três simuladores foram utilizados para a execução dos treinamentos. O LIBSVM foi utilizado para o treinamento com SVM, o SNNS para RBF-DDA e para as demais técnicas foi usado o simulador WEKA. Em todos foi utilizada uma técnica de comparação denominada validação cruzada, para garantir uma melhor eficiência dos treinamentos.

Ficou evidente, a partir dos treinamentos, que a opção por utilizar aprendizagem de máquina para sistemas de detecção de intrusão tem resultados eficientes, pois na maioria absoluta dos treinamentos houve uma taxa de acerto superior a 99% utilizando qualquer tipo de técnica.

A opção por fazer seleção do parâmetro θ- em redes RBF-DDA foi acertada. Com essa seleção conseguimos ter um desempenho até superior ao de SVM para esse problema, visto que das cinco bases de dados utilizadas, três obtiveram melhores resultados com a técnica RBF-DDA com a seleção do θ-. Essas bases obtiveram esses resultados, pois redes RBF são boas em reconhecer, e rejeitar, padrões discrepantes, ou seja, padrões que fogem das características da maioria dos padrões da base de dados, as bases que RBF obteve os melhores resultados foram com as base de dados onde tivemos poucos exemplos de ataques, já nas que haviam muitos exemplos de ataques, SVM obteve melhores resultados. Inclusive, em uma delas, NN e kNN tiveram o resultado idêntico ao de SVM. kNN também incluiu a seleção do número de vizinhos, os valores foram: 1, 3 e 5. Na maioria das redes, obtiveram-se os melhores resultados com o K=1. kNN e NN possuem maior complexidade computacional, pois necessitam armazenar todos os padrões utilizados para o treinamento. A menor complexidade computacional foi observada em MLP, pois precisou de apenas 40 unidades de processamento para obter seus resultados. Para redes do tipo MLP, foi utilizada 2 camadas escondidas, devido a complexidade do problema, e ainda 2 topologias diferentes. A primeira utilizava 20 neurônios em cada camada, já a segunda utilizava 40. O tempo de processamento também foi um fator analisado. SVM consegue realizar seus treinamentos em um curto espaço de tempo. Já Redes Neurais, RBF-DDA e MLP, obtiveram tempo de processamento alto para as bases de dados com uma quantidade maior de padrões, o mesmo aconteceu com as técnicas NN e kNN. SVM tem a vantagem de poder trabalhar, sem muitos problemas, com bases de dados grandes [2], o mesmo não acontece com Redes Neurais, pois elas demorariam muito no treinamento e as técnicas que utilizam o vizinho mais próximo precisariam além de um tempo grande para o treinamento, uma área de armazenamento grande.

Como trabalho futuro, propomos utilizar outras técnicas de aprendizagem de máquina, não para demonstrar que elas podem ser utilizadas em sistemas de detecção de intrusão, pois isso já foi demonstrado neste trabalho, e sim observar qual obtém os melhores resultados.

Outro trabalho que poderá ser proposto é a implementação de um sistema de detecção de intrusão, utilizando uma técnica de aprendizagem de máquina, visto que os resultados obtidos

neste trabalho comprovam a eficiência dessas técnicas. Hoje, um IDS com essas características não é comum.

A utilização de um maior número de padrões da base de dados original para treinamento poderá ser uma opção para trabalhos futuros. Isso pode melhorar ainda mais a confiabilidade nos resultados, visto que na realidade, o número de conexões é muito alto.

Bibliografia

[1] BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B. Redes Neurais Artificiais Teoria e Aplicações. Livros Técnicos e Científicos Editora, Rio de Janeiro, 2000.

[2] MUKKAMALA, S.; SUNG, A. H.; ABRAHAM, A. *Intrusion detection using an ensemble of intelligent paradigms*. Journal of Network and Computer Applications, volume 28, p. 167-182, 2005.

[3] BIERMAN, E.; CLOETE, E.; VENTER, L.M. *A comparison of Intrusion Detection Systems*. Computers & Security, volume 20, p. 676-683, 2001.

[4] HAYKIN, S. Redes Neurais, Princípios e Prática. 2.edição. Porto Alegre: Bookman, 2001.

[5] WANG, X. G., TANG, Z., TAMURA, H., ISHII, M., e SUN, W. D.. *An improved backpropagation algorithm to avoid the local minima problem*. Neurocomputing, 56:455–460, 2004.

[6] BERTHOLD, M. R.; DIAMOND , J. *Boosting the performance of RBF networks with dynamic decay adjustment.* Advances in Neural Information Processing, volume 7. p. 512-528, 1995.

[7] OLIVEIRA, A. L. I.; MELO, B. J. M.; MEIRA, S. R. L. *Integrated method for constructive training of radial basis function networks*. Electronics Letters, volume 41, p. 429-430, 2005.

[8] OLIVEIRA, A. L. I.; NETO, F. B. L.; MEIRA, S. R. L. *Improving RBF-DDA Performance on Optical Character Recognition through Parameter Selection*. Proceedings of the 17th International Conference on volume 4, p. 625–628, 2004.

[9] OLIVEIRA A. L. I.; A. MEDEIROS E. A.; ROCHA, T. A. B. V.; BEZERRA, M. E. R.; VERAS R. C. *A Study on the Infuence of Parameter θ- on Performance of RBF Neural Networks Trained with the Dynamic Decay Adjustment Algorithm*. Fifth International Conference on Hybrid Intelligent Systems, 2005.

[10] HUDAK, M. J. *RCE classifiers: Theory and practice*. Cybernetics and Systems, 23:483–515, 1992.

[11] CAMPBEL, C. *Kernel methods:a survey of current techniques*. Neurocomputing, volume 48, p. 63-84, 2002

[12] VAPNIK, V., *Statistical Learning Theory*, Wiley, New York, 1998.

[13] SVM - Support Vector Machines. Disponível em: <http://www.dtreg.com/svm.htm>, acessado em 05/10/2005.

[14] KUNCHEVA, L. I. *Reducing the computational demand of the nearest neighbor*. School of Informatics Symposium on Computing 2001. p.61-64, Aberystwyth, 2001.

[15] MELLISH, C.; BRINGHTON, H. *Advances in Instance Selection for Instance-Based Learning Algorithms*. Data Mining and Knowledge Discovery, volume 6, p. 153–172, 2002.

[16] LIPPMAN, R. P.; CUNNINGHAM, R. K. *Improving intrusion detection performance using keyword selection and neural networks*. Computer Networks, volume 34, p. 597-603, 2000.

[17] BARBOSA, A. S.; MORAIS, L. F. M. Sistemas de Detecção de Intrusão. Seminários Ravel - CPS760, UFRJ.

[18] JOO, D.; HONG, T.; HAN I. *The neural network models for IDS based on the asymmetric costs of false negative errors and false positive errors*. Expert Systems with Applications, volume 25, p. 69–75, 2003.

[19] BOMBONATO, F.; COELHO, F. E S. *Beholder - Utilizando Redes Neurais MPL na Detecção de Intrusos*.

[20] KDD Cup 1999 Data.Disponível em: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, visitado em 01/09/2005.

[21] MUKKAMALA, S.; JANOSKI, G.; SUNG, A. *Intrusion detection using neural networks and support vector machines*. IEEE International Joint Conference on Neural Networks, p. 1702-1707, 2002.

[22] MUKKAMALA, S.; SUNG, A. *Identifying Important Features for Intrusion Detection Using Support Vector Machines and Neural Networks*. Symposium on Applications and the Internet, p. 209-216, 2003.

[23] ZELL, A.;et al. Stuttgart neural network simulator. *Software* disponível em <http://www-ra.informatik.uni-tuebingen.de/SNNS>, visitado em 20/09/2005.

[24] WEBB, A. *Statistical Pattern Recognition*. Wiley, second edition. 2002.

[25] SERRA, S. A. T. G. *Uma aplicação de Redes Neurais Artificiais*. Trabalho disponível em: <http://www.inf.ufrgs.br/procpar/disc/cmp135/trabs/sergio/trab2/geo.html>, visitado em 8/10/2005.

[26] HSU, C. W.; CHANG, C. C.; LIN, C. J. *A Practical Guide to Support Vector Classification*. Disponível em: <http://www.csie.ntu.edu.tw/~cjlin/libsvm>. 2004, visitado em 02/10/2005.

[27] WEKA. *Software* dispinível em <http://www.cs.waikato.ac.nz/ml/weka/>, visitado em 25/10/2005.

[28] ZHANG, C.; JIANG, J.; KAMEL, M. *Intrusion detection using hierarchical neural networks*, Pattern Recognition Letters, volume 26, p. 779-791, 2005.

[29] LIU, Y.; CHEN, K.; LIAO, X.; ZHANG, W.; *A genetic clustering method for intrusion detection, Pattern Recognition*, volume 37, p. 927-942, 2004.

[30] OLIVEIRA, A. L. I.; MELO, B. J. M.; MEIRA, S. R. L. Improving constructive training of RBFnetworks through selective pruning and model selection. Neurocomputing, p. 537-541, 2005.

[31] MUELLER A. *Uma aplicação de redes neurais artificiais na previsão do mercado acionário*. Dissertação de Pós-graduação. Universidade Federal de Santa Catarina. Julho, 2005.

[32] Figura diponível em: <http://www.cogs.susx.ac.uk/users/jonh/>, visitado em 01/11/2005.

[33] KUNCHEVA L. I. *Reducing the computational demand of the nearest neighbor classifier. Mathematics Publications List*. Universidade de Wales. 2001.

[34] KOTROPOULOS, C.; PITAS, I. *Segmentation of ultrasonic images using Support Vector Machines*. Pattern Recognition Letters, vol. 24, p. 715-727, 2003.

[35] SHIMA, K.; TODORIKI, M.; SUZUKI, A. SVM-based feature selection of latent semantic features. Pattern Recognition Letters, vol. 25, p. 1051-1057, 2004.

[36] VOSSEN, J. P. *Snort Technical Guide*. Disponível em: <http://searchsecurity.techtarget.com/general/0,295582,sid14_gci1083823,00.html>, visitado em 05/11/2005.

[37] KUROSE, J. F.; ROSS, K. W. Redes de Computadores e a Internet, uma nova abordagem. Addison Wesley, p. São Paulo, 2003.

[38] BRIGHTON, H.; MELLISH C. *Advances in Instance Selection for Instance-Based Learning Algorithms*. Data Mining and Knowledge Discovery, vol. 6, p. 153-172, 2002.

[39] CAMPELLO, R. S.; WEBER, R. F. Sistemas de Detecção de Intrusão. Workshop em Segurança de Sistemas Computacionais, Florianópolis-SC, 2001.

[40] PEÑA, J. M.; BJÖRKEGREN J.; TEGNÉR, J. *Learning dynamic Bayesian network models via cross-validation*. Pattern Recognition Letters, vol. 26, p. 2295-2308, 2005.

Apêndice A

Base de dados do WEKA

Apresentaremos um modelo de uma base de dados no formato aceito pelo simulador WEKA.

@relation normal #nome da base de dados

@attribute duration numeric #@attribute descreve os atributos da base de dados
@attribute protocol_type {tcp,icmp,udp}
@attribute service
{auth,bgp,courier,csnet_ns,ctf,daytime,discard,domain,domain_u,eco_i,echo,ecr_i,efs,exec,finger
,ftp,ftp_data,gopher,hostnames,http,http_443,imap4,iso_tsap,klogin,kshell,ldap,link,login,mtp,na
me,netbios_dgm,netbios_ns,netbios_ssn,netstat,nnsp,nntp,ntp_u,other,pm_dump,pop_2,pop_3,pri
nter,private,remote_job,rje,shell,smtp,sql_net,ssh,sunrpc,supdup,systat,telnet,time,uucp,uucp_pat
h,vmnet,whois,Z39_50}
@attribute flag {SF,S1,S2,S3,SH,S0,REJ,RSTO,RSTR}
@attribute src_bytes numeric
@attribute dst_bytes numeric
@attribute land numeric
@attribute wrong_fragment numeric
@attribute urgent numeric
@attribute hot numeric
@attribute num_failed_logins numeric
@attribute logged_in numeric
@attribute num_compromised numeric
@attribute root_shell numeric
@attribute su_attempted numeric
@attribute num_root numeric

@attribute num_file numeric

@attribute num_shells numeric

@attribute num_access_files numeric

@attribute num_outbound_cmds numeric

@attribute is_host_login numeric

@attribute is_guest_login numeric

@attribute count numeric

@attribute srv_count numeric

@attribute serror_rate numeric

@attribute srv_serror_rate numeric

@attribute rerror_rate numeric

@attribute srv_rerror_rate numeric

@attribute same_srv_rate numeric

@attribute diff_srv_rate numeric

@attribute srv_diff_host_rate numeric

@attribute dst_host_count numeric

@attribute dst_host_srv_count numeric

@attribute dst_host_same_srv numeric

@attribute dst_host_diff_srv_rate numeric

@attribute dst_host_same_src_port_rate numeric

@attribute dst_host_srv_diff_host_rate numeric

@attribute dst_host_serror_rate numeric

@attribute dst_host_srv_serror_rate numeric

@attribute dst_host_rerror_rate numeric

@attribute dst_host_srv_rerror_rate numeric

@attribute ataques {normal,ataque}

@data #@data padrões de treinamento da base de dados

0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,215,8,1.00,1.00,0.00,0.00,0.04,0.07,0.00,255,8,0.03,0.08,0.00,0.00,1.00,1.00,0.00,0.00,ataque

0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,218,5,1.00,1.00,0.00,0.00,0.02,0.06,0.00,255,5,0.02,0.07,0.00,0.00,1.00,1.00,0.00,0.00,ataque

0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,normal ...

Apêndice B

Resultados do treinamento das redes RBF-DDA com seleção do parâmetro θ-

Neste apêndice, são mostrado todos os resultados advindos do treinamento das redes RBF-DDA com seleção do parâmetro θ-.

Treinamento com RBF					
Bases de Dados	Theta-	Nº Neurônios na CE	Tempo de Processamento	Erro de Validação Cruzada	Ciclos
Normal + Ataques	0,2	94	00:05:26	1,61%	4
Padrões: 15000	0,1	113	00:16:03	1,57%	4
	1,00E-02	214	00:46:46	0,63%	4
	1,00E-03	245	01:01:40	0,17%	4
	1,00E-04	273	01:02:37	0,09%	4
	1,00E-05	287	01:03:22	0,05%	3
	1,00E-06	300	00:59:00	0,07%	3
	1,00E-07	318	01:04:27	0,09%	3
	1,00E-08	332	01:09:00	0,12%	3
Normal + Probe	0,2	30	00:00:32	2,76%	4
Padrões: 3077	0,1	39	00:00:34	0,75%	3
	1,00E-02	63	00:00:54	0,29%	4
	1,00E-03	70	00:01:02	0,23%	4
	1,00E-04	76	00:01:01	0,20%	3
	1,00E-05	81	00:01:11	0,20%	3
	1,00E-06	87	00:01:26	0,16%	4
	1,00E-07	92	00:01:04	0,16%	4
	1,00E-08	95	00:01:08	0,16%	4
Normal + DOS	0,2	63	00:03:04	1,98%	3
Padrões: 14827	0,1	76	00:04:20	1,56%	4

	1,00E-02	163	00:34:21	0,57%	6
	1,00E-03	189	00:39:32	0,15%	4
	1,00E-04	216	00:43:30	0,05%	3
	1,00E-05	233	00:50:15	0,05%	4
	1,00E-06	245	00:51:03	0,03%	3
	1,00E-07	263	00:52:00	0,06%	3
	1,00E-08	274	00:58:00	0,06%	3
Normal + U2Su	0,2	21	00:00:25	0,37%	4
Padrões: 2964	0,1	24	00:00:27	0,37%	4
	1,00E-02	38	00:00:36	0,37%	4
	1,00E-03	45	00:00:37	0,24%	3
	1,00E-04	49	00:00:43	0,14%	4
	1,00E-05	55	00:00:47	0,14%	4
	1,00E-06	58	00:00:48	0,14%	4
	1,00E-07	64	00:00:52	0,17%	4
	1,00E-08	68	00:00:58	0,20%	4
Normal + R2L	0,2	19	00:00:28	1,27%	4
Padrões: 2991	0,1	27	00:00:30	1,27%	3
	1,00E-02	40	00:00:37	0,43%	3
	1,00E-03	46	00:00:41	0,13%	3
	1,00E-04	50	00:00:46	0,10%	3
	1,00E-05	54	00:00:47	0,10%	3
	1,00E-06	59	00:00:54	0,10%	3
	1,00E-07	67	00:00:58	0,10%	3
	1,00E-08	73	00:01:03	0,13%	4

Apêndice C

Resultados do treinamento de máquinas de vetor suporte (SVM)

Neste apêndice, apresentamos os resultados dos treinamentos executados utilizando SVM.

Treinamento com SVM				
Bases de Dados	Vetores Suporte	Tempo de Processamento	Erro de Validação Cruzada (%)	Ciclos
Padrões totais: 15000				
Normal + Ataques				
S/ Validação Cruzada	137	00:00:02	0,0143	5513
C/ Validação Cruzada (5)	126	00:00:08	0,0625	3531
C/ Validação Cruzada (10)	166	00:00:30	0,02	8347
Padrões totais: 3077				
Normal + Probe				
S/ Validação Cruzada	53	00:00:01	0,1354	335
C/ Validação Cruzada (5)	49	00:00:01	0,3127	272
C/ Validação Cruzada (10)	71	00:00:03	0,0975	457
Padrões totais: 14827				
Normal + DOS				
S/ Validação Cruzada	60	00:00:01	0,0439	4421
C/ Validação Cruzada (5)	92	00:00:04	0	2413
C/ Validação Cruzada (10)	123	00:00:30	0	7525
Padrões totais: 2964				
Normal + U2Su				
S/ Validação Cruzada	35	00:00:01	0,2049	242
C/ Validação Cruzada (5)	29	00:00:01	0,2001	123
C/ Validação Cruzada (10)	37	00:00:02	0,135	286

Padrões totais: 2991				
Normal + R2L				
S/ Validação Cruzada	38	00:00:01	0	478
C/ Validação Cruzada (5)	34	00:00:01	0,3336	400
C/ Validação Cruzada (10)	46	00:00:02	0,1672	440