

TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA PREVISÃO DE SUCESSO EM IMPLANTES DENTÁRIOS

**Trabalho de Conclusão de Curso
Engenharia da Computação**

**Nome do Aluno: Carolina Baldisserotto
Orientador: Prof. Adriano Lorena Inácio de Oliveira**

Recife, julho de 2005

TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA PREVISÃO DE SUCESSO EM IMPLANTES DENTÁRIOS

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.

Nome do Aluno: Carolina Baldisserotto
Orientador: Prof. Adriano Lorena Inácio de Oliveira

Recife, julho de 2005

Carolina Baldisserotto

**TÉCNICAS DE APRENDIZAGEM DE
MÁQUINA PARA PREVISÃO DE
SUCESSO EM IMPLANTES
DENTÁRIOS**

Resumo

Diversas técnicas de aprendizagem de máquina têm sido aplicadas com sucesso em problemas de classificação de padrões. Este trabalho contém um estudo comparativo entre as técnicas Redes Neurais Artificiais, do tipo MLP e RBF-DDA, Máquina de Vetor Suporte (SVM), Regra dos Vizinhos mais Próximos (KNN) e uma nova técnica, proposta recentemente na revista *IEEE Transactions on Neural Networks*, que é baseada na regra dos vizinhos mais próximos, porém com seleção de padrões (NNSRM), aplicadas ao problema de previsão de sucesso em implantes dentários. Os classificadores foram comparados usando validação cruzada e holdout. Para uma melhoria nos resultados, foi feita uma seleção de parâmetros nas técnicas RBF-DDA e SVM. Foram utilizados neste trabalho dois simuladores: o SNNS, para treinamento de redes neurais, e o LIBSVM, para treinamento de SVMs. Os treinamentos das técnicas KNN e NNSVM foram realizados através de implementações em linguagem JAVA. As técnicas foram analisadas em relação ao desempenho de classificação, complexidade do classificador e tempo para classificar novos padrões. As técnicas RBF-DDA, SVM e KNN obtiveram desempenhos similares, porém a técnica NNSVM levou vantagem em relação à complexidade do classificador e tempo para classificar novos padrões.

Abstract

Several machine learning techniques have been applied successfully to pattern classification problems. This work presents a comparative study between artificial neural networks techniques: Multi-Layer Perceptron (MLP) and Radial Basis Function (RBF-DDA), Support Vector Machines (SVM), nearest neighbor rule (KNN) and a new technique launched recently in the *IEEE Transactions on Neural Networks* Journal. The last technique is based on nearest neighbor rule however with pattern selection (NNSRM), applied to problem of prediction of success of dental implants. The classifiers were compared using cross validation and holdout in order to improve the results. A selection of parameters is done in RBF-DDA and SVM. In this work, it was used two simulators: the SNNS, to train and test neural networks and LIBSVM, to train and test SVMs. The training and testing of KNN and NNSRM were carried out using implementations on Java programming language. The techniques are compared taking into account classification performance, complexity of the classifiers, and time to classify a new pattern. The techniques RBF-DDA, SVM and KNN obtained similar performance, however the technique NNSRM presents the advantage of less complexity and less time to classify a new pattern.

Sumário

Índice de Figuras	v
Índice de Tabelas	vii
Introdução	9
1 Técnicas de aprendizagem de máquina	11
1.1 Redes neurais artificiais	11
1.1.1 Redes biológicas	11
1.1.2 Redes neurais artificiais	12
1.1.3 Redes neurais MLP	15
1.1.4 Redes neurais RBF	19
1.2 Máquinas de vetor suporte	23
1.3 Regra do vizinho mais próximo	24
1.4 Regra do vizinho mais próximo com seleção de padrões	25
1.4.1 Construção do classificador	25
2 Comparação de classificadores	29
2.1 Holdout	29
2.2 Validação cruzada	29
2.3 Leave-one-out	30
3 Ferramentas de simulação de aprendizagem de máquina	31
3.1 SNNS	31
3.1.1 Criação da rede	32
3.1.2 Visualização da rede	33
3.1.3 Arquivos	35
3.2 LIBSVM	36
3.3 Implementações KNN	40
4 Experimentos	42
4.1 Base de dados	42
4.2 Experimentos com redes neurais MLP	43
4.2.1 Topologia das redes neurais	44
4.2.2 Divisão dos dados	44
4.2.3 Metodologia de treinamento	44
4.2.4 Resultados obtidos	44
4.3 Experimentos com redes neurais RBF	45
4.3.1 Divisão dos dados	45
4.3.2 Metodologia de treinamento	45
4.3.3 Resultados obtidos	45
4.4 Experimentos com SVM	47
4.4.1 Metodologia de treinamento	47
4.4.2 Resultados obtidos	47

	iv
4.4.3 SVMs X RNAs	48
4.5 Experimentos com KNN e NNSRM	49
4.5.1 Metodologia de treinamento	49
4.5.2 Resultados obtidos	49
4.5.3 RBF-DDA x SVM x KNN x NNSRM	53
Conclusões e Trabalhos Futuros	55
Bibliografia	57
Apêndice A	60

Índice de Figuras

Figura1. Organização do sistema nervoso	12
Figura2. Neurônio artificial no modelo MCP	12
Figura3. Gráfico da função degrau	13
Figura4. Gráfico da função linear	13
Figura5. Gráfico da função sigmóide	14
Figura6. Organização em camadas de uma MLP	14
Figura7. Exemplo de problemas linearmente separável e não linearmente separável	16
Figura8. Arquitetura de uma RNA MLP	16
Figura9. Fluxo de processamento do algoritmo back-propagation	18
Figura10. RNA RBF típica com uma camada intermediária	19
Figura11. Conflito encontrado no algoritmo P-RCE	20
Figura12. O algoritmo DDA classifica corretamente entre duas classes conflitantes, através de limiares diferentes	21
Figura13. Exemplo do algoritmo DDA	22
Figura14. (a) Hiperplano com uma margem de separação menor (b) Hiperplano com uma margem de separação maior	23
Figura15. Esquema de classificação pelo Método KNN	25
Figura16. Ilustração do classificador NNSRM para dois padrões	27
Figura17. Demonstração gráfica do procedimento de validação cruzada 10-fold	30
Figura18. Painel inicial do SNNS	32
Figura19. Menu para criação das redes neurais pelo SNNS	32
Figura20. Painel para criação das redes neurais pelo SNNS	33
Figura21. Opção de visualização da rede neural pelo SNNS	34
Figura22. Visualização da rede MLP criada no SNNS	34
Figura23. Visualização da rede RBF criada no SNNS	35
Figura24. Opção de arquivos do SNNS	36
Figura25. Janela de manipulação de arquivos do SNNS	36
Figura26. Applet do LIBSVM	37
Figura27. Representação gráfica da classificação pelo applet do LIBSVM	37
Figura28. Execução do aplicativo svmtrain do LIBSVM	38
Figura29. Treinamento dos dados sem normalização e resultado da classificação pelo LIBSVM	39
Figura30. Treinamento dos dados normalizados e resultado da classificação pelo LIBSVM	39

Figura31. Treinamento dos dados através de validação cruzada (5-fold) pelo LIBSVM	40
Figura32. Idades normalizadas entre 0 e 1	43
Figura33. Quantidade de padrões armazenados nas técnicas KNN e NNSRM para cada conjunto de dados	52
Figura34. Comparativo entre os erros de classificação obtidos por cada técnica via validação cruzada	53
Figura35. Comparativo entre a complexidade das técnicas RBF-DDA, SVM, KNN e NNSRM	54

Índice de Tabelas

Tabela 1. Características analisadas e possíveis valores	42
Tabela 2. Distribuição das classes na base de dados	42
Tabela 3. Erro de classificação de cada topologia de rede MLP	45
Tabela 4. Resultado da classificação da redes RBF variando o parâmetro θ	46
Tabela 5. RBF com parâmetro default e validação cruzada (10-fold)	46
Tabela 6. RBF-DDA com parâmetros selecionados	47
Tabela 7. SVM com parâmetros default	48
Tabela 8. SVM com seleção de parâmetros	48
Tabela 9. Comparação entre as técnicas RBF-DDA e SVM	49
Tabela 10. KNN com holdout	50
Tabela 11. KNN com validação cruzada (10-fold)	50
Tabela 12. NNSRM com holdout	51
Tabela 13. NNSRM com validação cruzada (10-fold)	51
Tabela 14. Comparativo entre os melhores resultados da técnica KNN	52
Tabela 15. Comparativo entre os erros de classificação obtidos pelas técnicas via validação cruzada	53
Tabela 16. Comparativo entre a quantidade de padrões armazenados em cada técnica	54

Agradecimentos

Gostaria de agradecer a todos que me apoiaram ao longo desse trabalho:

A Deus, que me deu saúde e força para o desenvolvimento deste trabalho;

Aos meus pais, Claudio Baldisserotto e Maria da Glória Baldisserotto, que nunca me deixaram faltar nada durante toda a minha vida;

Ao meu namorado, Otávio Francisco da Silva, que sempre me apóia em minhas decisões e me dá ânimo para continuar;

Ao meu tio, Júlio Baldisserotto, que me inspirou no tema deste trabalho e me forneceu material para realizá-lo;

Ao meu orientador, Adriano Lorena Inácio de Oliveira, que me ensinou a parte teórica e me orientou durante todo o trabalho;

Aos meus professores e amigos da POLI que caminharam junto comigo durante todo o curso;

Aos meus familiares e amigos.

Introdução

A demanda pelos pacientes de reabilitações orais pelo uso de implantes dentários está crescendo em um ritmo significativo [1, 3]. A técnica de implantes dentários tem sido utilizada para substituir a perda de peças dentárias. Essas perdas podem ser unitárias, parciais e até a perda total de dentes. Os implantes dentários vêm sendo utilizados de forma rotineira na recuperação de dentes perdidos e com taxas de sucesso bastante altas [3]. Porém, a reabilitação através do uso de implantes apresenta riscos de insucessos relacionadas a diferentes etapas do processo de cicatrização (osseointegração) do implante ao osso adjacente. Diversos fatores podem estar relacionados ao insucesso no tratamento com implantes como: as condições de saúde geral do paciente, complexidade e técnicas cirúrgicas, técnica protética adequada, fumo, tipo do implante, etc. Neste trabalho, o sucesso é considerado quando o implante apresenta características de osseointegração nas diferentes etapas do processo desde a implantação, colocação da prótese e à preservação sem a ocorrência de problemas. Admite-se que houve insucesso se, durante esse tempo, tiver ocorrido algum problema relacionado ao implante e o mesmo necessitou ser removido.

As características dos pacientes analisadas para cada implante foram cuidadosamente selecionadas para terem algum relacionamento com o resultado. Foram consideradas sete características: idade do paciente, sexo, localização do implante, tipo de implante, técnica cirúrgica, se o paciente é fumante e se é portador de alguma doença que influencie no resultado final (ex.: osteoporose, diabetes...). Foram coletados dados de cento e cinquenta e sete implantes dentários realizados entre os anos de 1998 e 2004 em uma clínica dentária na cidade de Porto Alegre.

Como, no período que foram coletados os dados, havia implantes realizados há menos de cinco anos, foi feita uma divisão temporal no resultado da previsão, ou seja, o resultado não terá apenas duas opções, sucesso ou insucesso, e sim, sete opções: sucesso confirmado de até um ano; sucesso confirmado de 1 a 2 anos; sucesso confirmado de 2 a 3 anos, sucesso confirmado de 3 a 4 anos, sucesso confirmado de 4 a 5 anos, sucesso confirmado por mais de 5 anos e insucesso. Quanto maior a quantidade de anos confirmada, maior a garantia de sucesso do implante.

Atualmente, a previsão de sucesso de um implante dentário é feita pelo Cirurgião Dentista, através de avaliação clínica e radiográfica. Este trabalho tem por objetivo prever o sucesso de um implante dentário através das características do paciente que realizou o implante.

Neste trabalho comparamos quatro técnicas de aprendizagem de máquina para essa previsão. Estas técnicas são Redes Neurais Artificiais (RNA), do tipo MLP [36, 42, 43, 6] e RBF [29, 4, 5], Máquinas de Vetor Suporte (SVM) [35, 34, 10], Regra dos vizinhos mais próximos (KNN) [39, 30] e uma nova técnica, lançada recentemente na revista *IEEE Transactions on Neural Networks* [23, 22], que é baseada na regra dos vizinhos mais próximos, porém com seleção de padrões de treinamento através da minimização estrutural do risco (*Structural Risk Minimization - SRM*) (NNSRM). Resultados de simulações com esse novo método mostraram que ele possibilita uma redução significativa no custo computacional em relação ao SVM. As técnicas RNA, SVM e KNN foram selecionadas para esse estudo porque estão sendo muito

utilizadas atualmente, com resultados satisfatórios, e a técnica NNSRM foi selecionada por ser uma técnica recente e inovadora que está propondo melhores resultados e vantagens em relação às técnicas já existentes. Todas as técnicas receberão como entrada as características dos pacientes e retornarão como saída a previsão, se o implante resultará em falha, ou sucesso de tempo determinado, após realizado.

Este trabalho está estruturado em 5 capítulos. O primeiro capítulo revisará os classificadores, ou técnicas de aprendizagem de máquina, utilizados neste estudo comparativo. Serão apresentadas as redes neurais artificiais, especificamente os dois tipos que foram utilizados neste trabalho, que são as redes MLP e RBF, assim como seu funcionamento, métodos de aprendizagem e algoritmos de treinamento. Também serão revisadas as máquinas de vetor suporte (SVM), a regra dos vizinhos mais próximos (KNN), e o método dos vizinhos mais próximos com seleção de padrões (NNSRM).

O segundo capítulo apresenta algumas técnicas utilizadas para comparar esses classificadores. As principais técnicas usadas neste trabalho foram validação cruzada [11, 13] e *holdout* [40, 32], porém este capítulo mostra também a técnica *leave-one-out* [12, 32], que é uma simplificação da validação cruzada.

O terceiro capítulo apresenta algumas ferramentas de simulação de técnicas de aprendizagem de máquina. Para as técnicas de redes neurais foi utilizada a ferramenta SNNS [37], pois é uma ferramenta simples e com funcionalidades que atendem ao objetivo deste trabalho. Pela mesma razão, foi escolhida a ferramenta LIBSVM [19] para a técnica de máquinas de vetor suporte. Com relação a KNN e NNSRM, essas técnicas foram implementadas na linguagem Java.

O quarto capítulo contém os experimentos realizados para a comparação dos classificadores considerando tanto o desempenho de classificação quanto a complexidade. Isso inclui a preparação da base de dados, o pré-processamento dos dados, as simulações e os resultados obtidos.

Finalmente, o quinto capítulo apresenta as conclusões e sugestões para trabalhos futuros.

Capítulo 1

Técnicas de aprendizagem de máquina

Neste capítulo são explicadas as técnicas de aprendizagem de máquina utilizadas neste trabalho: redes neurais artificiais (RNA) do tipo MLP e RBF-DDA, máquinas de vetor suporte (SVM), técnica dos vizinhos mais próximos (KNN) e a técnica dos vizinhos mais próximos com seleção de padrões (NNSRM).

1.1 Redes neurais artificiais

RNAs [7, 24, 6] constituem uma base de técnicas de aprendizagem de máquina muito utilizada atualmente, com um vasto campo de pesquisa, e que está em crescimento constante. Essa técnica é muito utilizada para reconhecimento de padrões, o que a torna adequada para este trabalho. A base de dados utilizada pode vir de diversos tipos de fontes, o que atrai muitos pesquisadores para a área. As RNAs se baseiam no sistema nervoso humano, nas estruturas do cérebro e na sua capacidade de aprendizagem [6, 32]. Portanto ela tenta “imitar” o comportamento do cérebro humano, esperando, com isso, conseguir aprender a aprender. Os métodos anteriores possuíam uma eficiência menor porque apenas “decoravam” dados, características e regras, tornando os sistemas menos flexíveis, ou seja, sem inteligência. Hoje em dia, com as RNAs, pode-se construir sistemas inteligentes, que obtêm resultados bastante eficientes, aprendendo de acordo com seu próprio funcionamento e com o ambiente.

1.1.1 Redes biológicas

O sistema nervoso é formado, basicamente, por células, que se interconectam de forma precisa, formando os chamados circuitos neurais. Através desses circuitos são definidos os comportamentos, fixos ou variáveis. Todo ser vivo dotado de um sistema nervoso é capaz de modificar o seu comportamento através de experiências passadas. Essa modificação de comportamento é chamada de aprendizado.

O principal componente do sistema nervoso é o neurônio. Os neurônios possuem, entre outros elementos, os dendritos, que são responsáveis pela comunicação entre eles, e o axônio, responsável por transmitir o sinal para o próximo neurônio. O processo de comunicação entre os neurônios é feito através da sinapse, que é onde o sinal que vem do axônio de um neurônio é recebido pelos dendritos do outro neurônio. A Fig. 1 ilustra a organização do sistema nervoso.

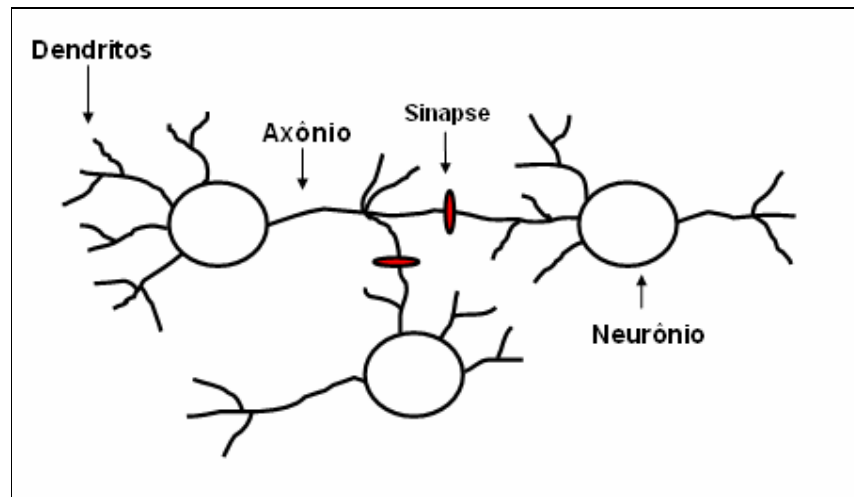


Figura 1. Organização do sistema nervoso

1.1.2 Redes neurais artificiais

Assim como o cérebro, as redes neurais artificiais (RNAs) possuem unidades simples (os neurônios), que se interconectam uns aos outros, formando redes capazes de armazenar e transmitir informação [32]. Os elementos básicos de um neurônio (modelo MCP – McCulloch e Pitts) numa RNA são: os pesos sinápticos, a função soma e a função de ativação, como mostra a Fig. 2.

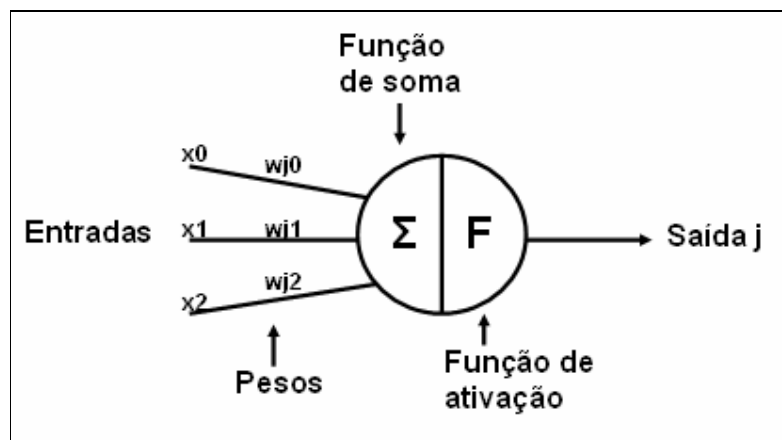


Figura 2. Neurônio artificial no modelo MCP

As conexões entre os neurônios, denominadas pesos sinápticos, são responsáveis pelo armazenamento das informações, e também definem o efeito que a saída de um neurônio exerce sobre a entrada do neurônio seguinte.

A função de soma processa os estímulos recebidos, com seus respectivos pesos, ou seja:

$$x_j = \sum_i w_{ij} y_i \quad (1.1)$$

A função de ativação, também chamada função de transferência, limita a amplitude do intervalo do sinal de saída do neurônio para algum valor finito, geralmente no intervalo normalizado $[0,1]$ ou $[-1,1]$.

$$y_j = f(x_j) \quad (1.2)$$

Existem diversas funções de ativação, porém as mais utilizadas são:

a) Função Degrau. É o tipo mais simples de função de ativação. Sua resposta pode assumir os valores 0 ou 1, como demonstrado a seguir:

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

A Fig. 3 ilustra graficamente a função degrau.

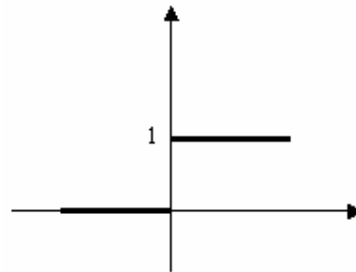


Figura 3. Gráfico da função degrau

b) Função Linear. Um exemplo de função linear pode ser descrito como:

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

A Fig. 4 ilustra graficamente a função linear.

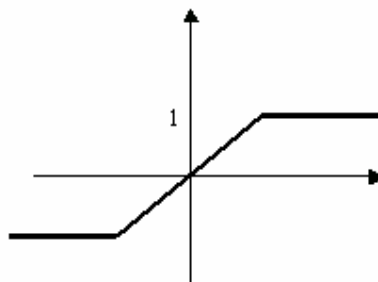


Figura 4. Gráfico da função linear

onde $(-\frac{1}{2}$ e $\frac{1}{2})$ é o intervalo que define a saída linear, e 0 e 1 são os limites mínimo e máximo da função.

c) Função Sigmóide. Esta função assume valores num intervalo contínuo entre 0 e 1. Sua fórmula é dada por:

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

A Fig. 5 ilustra graficamente a função sigmóide.

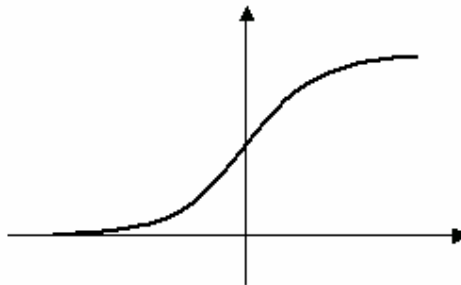


Figura 5. Gráfico da função sigmóide

na qual α determina a inclinação da função.

Além dos elementos já citados, o neurônio pode apresentar um *bias* (limiar) que tem o efeito de aumentar ou diminuir a entrada da função de ativação [18]. O termo *bias* age como um peso extra nas conexões das unidades cuja entrada é sempre 1 [15].

As RNAs do tipo MLP (*Multi Layer Perceptron*) possuem camadas bem definidas, que são: camada de entrada, camadas intermediárias e camada de saída. A organização em camadas de uma RNA está demonstrada na Fig 6.

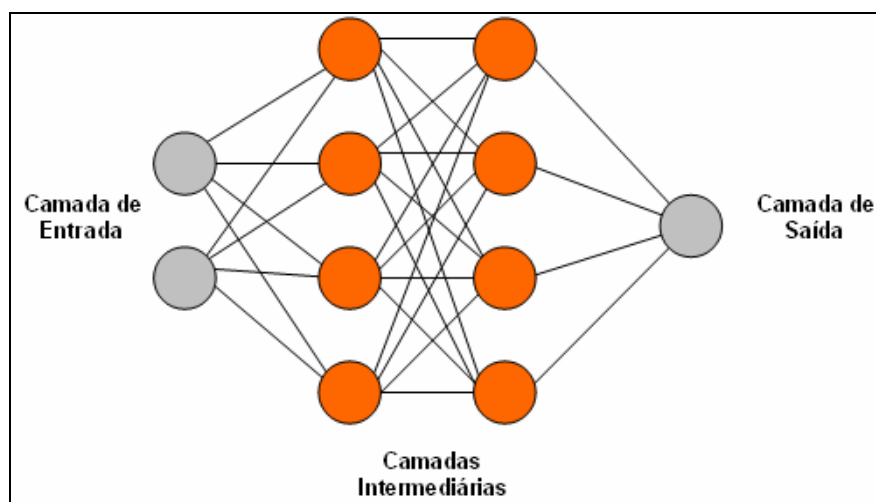


Figura 6. Organização em camadas de uma MLP

Vários aspectos diferenciam uma rede neural de outra, tais como, o tipo de conexão, número de camadas escondidas e o tipo de treinamento. Cada combinação de características é mais adequada para um determinado problema. O tipo de conexão utilizada pela rede neural

determina a topologia (ou arquitetura) da rede. Os principais tipos de conexão estão descritos a seguir:

- a) Redes alimentadas adiante (*feedforward*): nesse tipo de rede uma camada é ligada somente à próxima camada, não havendo conexões entre neurônios de uma mesma camada, ou conexões com uma camada anterior. Essa estrutura é totalmente conectada, uma vez que todas as saídas dos neurônios de uma camada são conectadas com as entradas de todos os neurônios da camada seguinte.
- b) Redes recorrentes: uma camada pode se conectar com qualquer outra camada. São permitidas redes com realimentação, onde os neurônios da camada de entrada recebem sinais diretamente da camada de saída.

Após definida a topologia da rede neural, ela é submetida a um processo de aprendizagem [18, 38], pois o objetivo de uma rede neural é aprender a generalizar os dados de entrada e não apenas decorá-los. Ela deve ser capaz de classificar corretamente não só os dados já apresentados, como também dados novos que venham a aparecer. As redes neurais, assim como o cérebro humano, aprendem a partir de experiência, portanto deve-se tomar muito cuidado na escolha da base de dados que será fornecida como entrada para a aprendizagem da rede neural. A aprendizagem é feita através de um processo iterativo de ajustes nos pesos, o treinamento. Esse ajuste é feito de acordo com os erros gerados pela rede. Ou seja, a rede neural é capaz de modificar-se em função da necessidade de aprender a informação que lhe foi apresentada [38]. O treinamento acaba quando é encontrada uma solução generalizada para o problema em questão.

Os três principais paradigmas de aprendizagem são apresentados a seguir [6]:

- a) Aprendizagem supervisionada (com “professor”): nesse aprendizado, é fornecido tanto o padrão de entrada, como a saída desejada pela rede para tal padrão. O erro é calculado comparando a saída obtida pela rede com a saída desejada, fornecida previamente.
- b) Aprendizagem não supervisionada (sem “professor”): nesse tipo de aprendizagem, não existe um agente externo indicando a saída desejada para os padrões de entrada. A rede neural utiliza os neurônios como classificadores e os dados de entrada como elementos de classificação. Esse tipo de rede trabalha essas entradas e se organiza de modo a classificá-las mediante algum critério de semelhança.
- c) Aprendizagem por reforço: nessa aprendizagem, a rede recebe a informação que a saída está certa ou errada, porém não é fornecida à rede a saída correta.

Em um processo de aprendizagem os pesos das conexões dos neurônios são ajustados através de um algoritmo de aprendizagem: um conjunto pré-estabelecido de regras bem definidas para a resolução de um problema de aprendizagem [18]. O algoritmo tem como objetivo encontrar os pesos adequados que façam a rede fornecer as saídas desejadas.

1.1.3 Redes neurais MLP

As RNAs do tipo MLP (MultiLayer Perceptron) [36, 42, 43, 6] são RNAs que apresentam pelo menos uma camada intermediária ou escondida. A RNA mais simples é a rede com apenas um neurônio, denominada perceptron, proposta por Frank Rosenblatt em 1958 [18]. Essas redes apenas conseguem resolver problemas linearmente separáveis, ou seja, que podem ser resolvidos com apenas uma única reta ou hiperplano (em espaços de dimensão maior que 2). As MLPs maiores, com camadas escondidas, possuem um melhor poder computacional se comparado às

RNAs sem camada escondida, pois podem tratar com dados que não são linearmente separáveis, o que ocorre na maioria das vezes na vida real [6]. Um exemplo de problemas linearmente separável e não-linearmente separável está na Fig. 7. Uma rede neural com uma camada intermediária pode implementar qualquer função contínua, e, com pelo menos duas camadas intermediárias, pode implementar qualquer função, linearmente separável ou não [6].

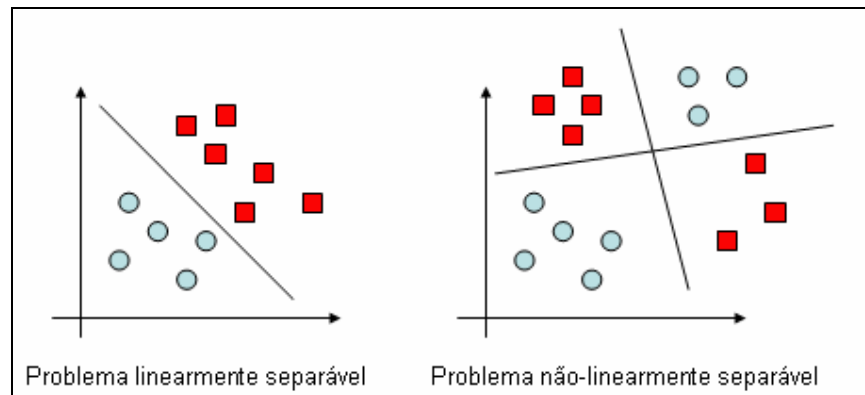


Figura 7. Exemplo de problemas linearmente separável e não linearmente separável

Uma demonstração da arquitetura de uma RNA MLP é apresentada na Fig. 8.

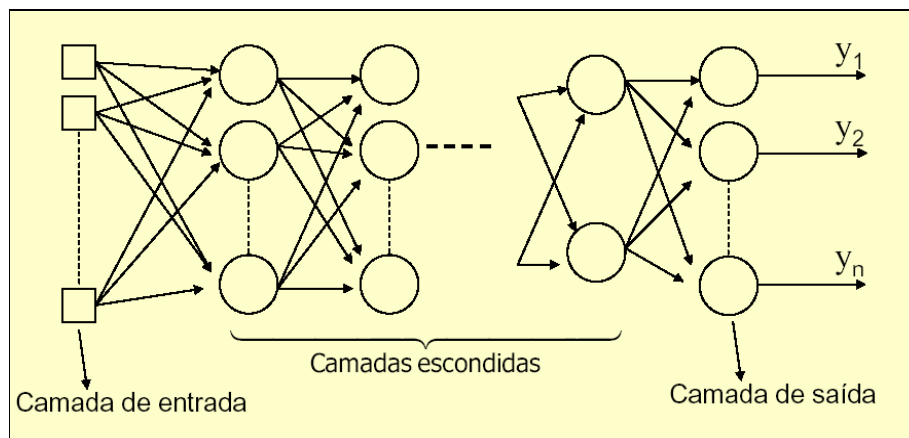


Figura 8. Arquitetura de uma RNA MLP

Diversos fatores influenciam no desempenho de uma RNA, e a escolha certa deles é de extrema importância, podendo levar a um resultado ótimo, se as escolhas forem corretas, ou levar a resultados insatisfatórios para uma mesma base de dados, se as escolhas forem erradas. Por isso, neste trabalho, bastante tempo foi gasto na análise e escolha desses fatores, para não comprometer o resultado final.

Um dos primeiros fatores a ser escolhido para a RNA é a função de ativação a ser utilizada. Existem diversas funções de ativação propostas para RNAs MLP, porém a que utilizaremos é a função sigmoideal logística [6], que é a mais usada atualmente. Outro fator a ser escolhido previamente é a quantidade de nodos da RNA, tanto da camada de entrada, quanto das camadas escondidas e de saída. Para nosso problema, são usadas redes MLP com dez nodos na camada de entrada e sete nodos na camada de saída. A camada intermediária varia entre 2, 5 e 10 nodos. A camada de saída usa a classificação *winner-takes-all* [17], onde a unidade de saída que possuir o maior peso é a “vencedora”, ou seja, o padrão de entrada pertence àquela classe.

O número de nodos na camada intermediária depende de vários fatores, tais como, número de exemplos de treinamento, quantidade de ruído nos dados de entrada, complexidade da função a ser aprendida, ou distribuição estatística dos dados de treinamento. Deve-se ter cuidado para não escolher um número muito grande de nodos na camada intermediária, pois assim a rede acabaria “decorando” os dados, perdendo, portanto, a generalização, ou seja, caso apareça um elemento novo para a rede identificar, é muito provável que ela irá errar, pois ela decorou apenas os dados que foram apresentados a ela anteriormente. Esse problema é chamado *overfitting* [29, 28, 27]. Deve-se também ter cuidado para não escolher um número muito pequeno de nodos, pois a rede pode perder muito tempo sem achar um resultado ótimo, ou seja, sem convergir para uma resposta.

Para se evitar um *overfitting*, uma opção é utilizar o método da parada antecipada. Um *overfitting* ocorre quando, durante o treinamento, ao invés da rede melhorar o desempenho, ela começa a piorar, então o método da parada antecipada propõe que se pare o treinamento antes que a rede comece a diminuir o desempenho e perder a generalização. Para isso os padrões de treinamento são divididos em dois conjuntos, um conjunto de treinamento e um conjunto de validação. O conjunto de validação serve para verificar o desempenho da rede durante o treinamento. Ele não será usado para ajuste de pesos, sendo usado periodicamente para verificar a generalização da rede enquanto a mesma está sendo treinada. No início do treinamento, o erro de validação tende a cair, porém em um determinado momento esse erro começa a aumentar, concluindo o treinamento. O critério de parada GL5 [31], utilizado neste projeto, trabalha calculando a diferença entre o erro de validação atual e o erro de validação mínimo encontrado até o momento, durante a fase de treinamento. Quando essa diferença for maior ou igual a cinco por cento, o treinamento é parado, indicando que o erro está aumentando. A fórmula para calcular a generalização pelo critério GL(t) está mostrada abaixo.

$$GL(t) = 100 \times \left(\frac{E_{val}(t)}{E_{opt}(t)} - 1 \right)$$

Onde $E_{opt}(t)$ é o erro de validação mínimo obtido durante o treinamento até a época t, e $E_{val}(t)$ é o erro de validação na época t.

Um outro fator fundamental na criação de uma RNA MLP é o algoritmo de aprendizado. Atualmente existem vários algoritmos, sendo o mais conhecido o *back-propagation* [45, 6]. O algoritmo *back-propagation* é um algoritmo supervisionado, que recebe como entrada pares (entrada, saída desejada), e utiliza um mecanismo de correção de erros que vai ajustando os pesos da rede. O treinamento possui duas fases: *forward*, que é realizada no sentido entrada-saída, onde são calculadas as saídas para um determinado padrão de entrada; e a fase *backward* que calcula o erro gerado pela rede, através da saída desejada e da saída conhecida, e ajusta os pesos das conexões. A Fig. 9 ilustra essas duas fases.

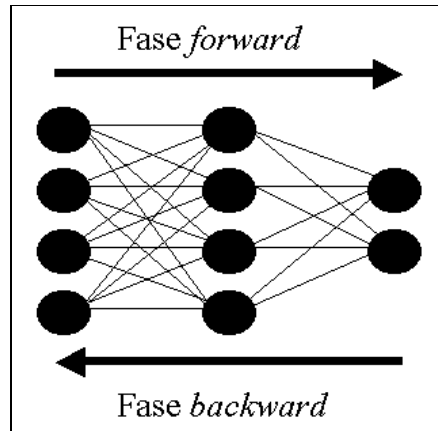


Figura 9. Fluxo de processamento do algoritmo back-propagation

Apesar do algoritmo back-propagation ser satisfatório em muitas aplicações, várias alterações têm sido propostas, tanto para diminuir o seu tempo de treinamento, quanto para melhorar o desempenho na classificação de padrões. Utilizamos o algoritmo Rprop (*resilient back-propagation algorithm*) [33, 25] que é uma dessas variações do algoritmo back-propagation.

Algoritmo Rprop

O Rprop é um algoritmo de aprendizagem local adaptativo que trabalha nas atualizações dos pesos. Ele elimina a influência prejudicial do tamanho da derivada parcial na atualização dos pesos $\Delta w_{ij}(t)$ [25]. Cada peso possui um valor de atualização $\Delta w_{ij}(t)$ individual, o qual indica o quanto esse peso vai ser ajustado. Em cada época do treinamento, todos os pesos são ajustados pela seguinte fórmula:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \quad (1.6)$$

onde $\Delta w_{ij}(t)$ é dado por:

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t), & \text{se } \frac{\partial E(t)}{\partial w_{ij}} > 0 \\ +\Delta_{ij}(t), & \text{se } \frac{\partial E(t)}{\partial w_{ij}} < 0 \\ 0, & \text{caso contrário} \end{cases} \quad (1.7)$$

Ou seja, se a derivada for positiva, o peso é decrementado do valor de atualização. Se a derivada for negativa o peso será incrementado do mesmo valor.

Esses valores de atualização adaptativos envolvem, durante o treinamento, de acordo com o comportamento local da função erro E. Isso é representado pela equação 1.8.

$$\Delta_{ij}(t) = \begin{cases} \eta^+ \times \Delta_{ij}(t-1), & \text{se } \frac{\partial E(t-1)}{\partial w_{ij}} \times \frac{\partial E(t)}{\partial w_{ij}} > 0 \\ \eta^- \times \Delta_{ij}(t-1), & \text{se } \frac{\partial E(t-1)}{\partial w_{ij}} \times \frac{\partial E(t)}{\partial w_{ij}} < 0 \\ \Delta_{ij}(t-1), & \text{caso contrário} \end{cases} \quad (1.8)$$

Assim, em cada época, se a derivada parcial do erro muda de sinal, o valor de atualização é decrementado do valor η^- . Mas se a derivada mantém o sinal, o $\Delta w_{ij}(t)$ é incrementado do valor η^+ .

Inicialmente, todos os $\Delta w_{ij}(t)$ são iniciados com um valor Δ_0 . Uma boa escolha é $\Delta_0 = 0,1$ [33]. O Rprop possui dois parâmetros usados para restringir os limites de $\Delta w_{ij}(t)$: Δ_{\max} e Δ_{\min} , que definem o limite superior e o limite inferior respectivamente. Seus valores padrão são $\Delta_{\max} = 50,0$ e $\Delta_{\min} = 10^{-6}$. Experimentos têm mostrado que os valores $\eta^+ = 1,2$ e $\eta^- = 0,5$ são uma boa escolha para os parâmetros de incremento e decremento [33].

1.1.4 Redes neurais RBF

Um outro tipo de RNA são as redes de Funções Base Radiais (RBF) [29, 27, 4, 5], que são utilizadas para aplicações em problemas de aprendizagem supervisionada (regressão, classificação, previsão de séries temporais). Ela possui este nome por causa da utilização, pelos nodos da camada intermediária, de funções de base radiais. A Fig. 10 ilustra uma rede neural do tipo RBF.

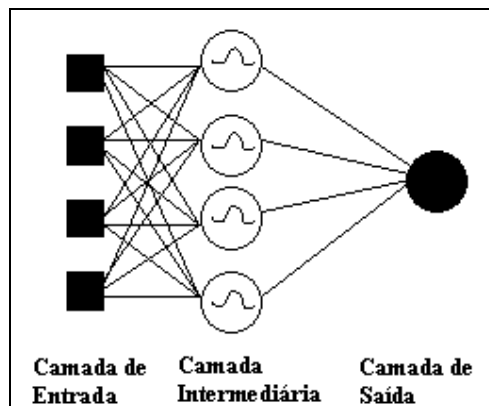


Figura 10. RNA RBF típica com uma camada intermediária

Essas redes, na maioria das vezes, possuem uma única camada escondida [7, 6] e seu treinamento é, geralmente, mais rápido que o treinamento das MLPs.

A função de ativação aplicada a um nodo de uma RNA RBF utiliza como medida a distância entre os vetores de entrada e de peso. Funções radiais representam uma classe especial

de funções cujo valor diminui ou aumenta em relação à distância de um ponto central [24]. Diferentes funções de base radial têm sido utilizadas em redes RBF. As mais comuns são:

- Função Gaussiana: $f(u) = \exp\left(\frac{-v^2}{\sigma_i^2}\right)$ (1.9)

- Função multiquadrática: $f(u) = \sqrt{(v^2 + \sigma^2)}$ (1.10)

- Função thin-plate-spline: $f(u) = v^2 \log(v)$ (1.11)

Onde v é a distância euclidiana, definida, por $v = \|x - \mu\|$, \bar{x} é o vetor de entrada, $\bar{\mu}$ e σ representam o centro e a largura da função radial, respectivamente. Usamos a função Gaussiana como função de ativação dos nodos da camada escondida.

O algoritmo de treinamento usado nas redes RBF é o DDA (*Dynamic Decay Adjustment*) [29, 26, 4, 5]. Com esse algoritmo, as redes não precisam de uma estrutura física totalmente pré-definida, como as RNAs do tipo MLP, elas constroem sua arquitetura dinamicamente durante o treinamento. Não é necessário definir a quantidade de camadas intermediárias, pois o algoritmo é construtivo, ou seja, ele mesmo calcula o melhor número de camadas escondidas, assim como o número de nodos em cada camada escondida.

Algoritmo DDA (*Dynamic Decay Adjustment*)

O algoritmo DDA é um algoritmo construtivo baseado no algoritmo RCE (*Restricted Coulomb Energy*) [20]. O algoritmo RCE se baseia no fato de que para cada padrão de entrada é criada uma unidade RBF, para classificá-la. Porém, este algoritmo pode ter problemas, como indicado na Fig. 11. Nela, são criados dois RBFs para identificar dois padrões de entrada, A e B. Porém, se um novo padrão de entrada do tipo B se encontrar entre os dois RBFs, o algoritmo vai se confundir para classificá-lo corretamente, sem saber se o padrão é do tipo A ou do tipo B.

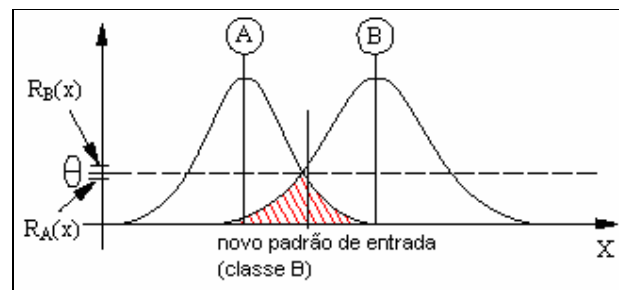


Figura 11. Conflito encontrado no algoritmo P-RCE

Para resolver esse problema, o algoritmo DDA introduz dois limiares: um limiar positivo (θ^+), que deve ser superado por uma ativação do protótipo de mesma classe para não ser adicionado um novo protótipo, e um limiar negativo (θ^-), que é o limite superior para a ativação de classes conflitantes. A Fig. 12 mostra um exemplo de um novo padrão que está acima do limiar positivo, para o RBF da classe B, e que está abaixo do limiar negativo do RBF da classe A, indicando, portanto, que esse padrão é da classe B.

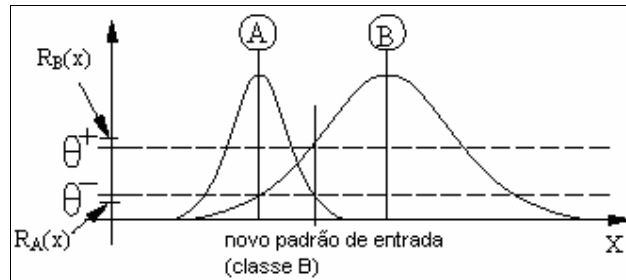


Figura 12. O algoritmo DDA classifica corretamente entre duas classes conflitantes, através de limiares diferentes

Dois equações são utilizadas para cada padrão de entrada \vec{x} da classe c nos dados de treinamento:

$$\exists i : R_i^c(\vec{x}) \geq \theta^+ \quad (1.12)$$

$$\forall k \neq c, 1 \leq j \leq m_k : R_j^k(\vec{x}) < \theta^- \quad (1.13)$$

O algoritmo 1 mostra uma época de treinamento para um padrão de entrada x da classe c .

Primeiro, todos os pesos são iniciados com valor zero para não acumularem valores duplicados sobre os padrões de treinamento [passo 1 do algoritmo 1]. Depois, para todos os padrões de treinamento é verificado: se ele for classificado corretamente, o peso do protótipo é incrementado. Caso contrário, um novo protótipo é criado, sendo o centro do protótipo, este padrão [passo 2 do algoritmo 1]. A última etapa do algoritmo “encolhe”, ou seja, aproxima os valores da função ao centro, todos os protótipos de classes conflitantes, se suas ativações forem muito altas para o padrão específico [passo 6 do algoritmo 1]. A Fig. 13 ilustra os passos de treinamento do algoritmo DDA.

No exemplo da Fig.10, inicialmente [Fig. 13 (a)] um padrão de entrada da classe A é apresentado e uma nova RBF é criada. Depois [Fig. 13 (b)], um novo padrão é apresentado, e como ele passa pelo protótipo A mas está abaixo do seu limiar negativo, ele não é reconhecido pelo padrão A, com isso, um novo protótipo B é criado, e o raio da RBF da classe A é diminuído. Outro padrão da classe B é apresentado [Fig. 13 (c)] e classificado corretamente, então o peso do protótipo B é incrementado, e o protótipo A é novamente encolhido. Por fim, [Fig. 13 (d)] um novo padrão da classe A é apresentado e é criado um novo protótipo dessa classe.

Algoritmo 1. Algoritmo DDA para treinamento RBF

```

1: //inicializa os pesos:
  FORALL protótipos  $p_i^k$  DO
     $A_i^k = 0$ 
  ENDFOR
2: //Treina uma época completa
  FORALL padrões de treinamento  $(\vec{x}, c)$  DO
    IF  $\exists p_i^c : R_i^c(\vec{x}) \geq \theta^+$  THEN
3:      $A_i^c += 1.0$ 
    ELSE
4:     “//commit”: introduz um novo protótipo
    Adiciona novo protótipo  $p_{m_c+1}^c$  com:
       $\vec{r}_{m_c+1}^c = \vec{x}$ 
       $A_{m_c+1}^c = 1.0$ 
       $m_c += 1$ 
5:      $\sigma_{m_c+1}^c = \max_{k \neq c, 1 \leq j \leq m_k} \{ \sigma : R_{m_c+1}^c(\vec{r}_j^k) < \theta^- \}$ 
    ENDIF
6:     “//shrink”: ajustar protótipos conflitantes
    FORALL  $k \neq c, 1 \leq j \leq m_k$  DO
       $\sigma_j^k = \max \{ \sigma : R_j^k(\vec{x}) < \theta^- \}$ 
    ENDFOR
  ENDFOR
  ENDFOR

```

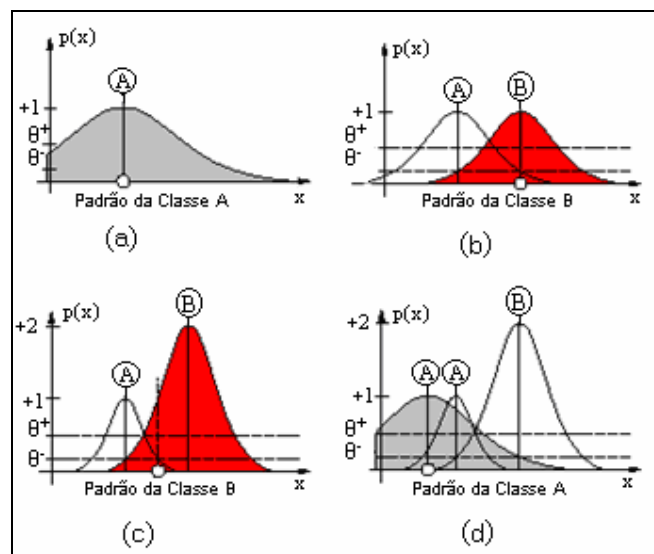


Figura 13. Exemplo do algoritmo DDA

Um fator que pode influenciar no desempenho da RNA RBF é o limiar negativo, θ^- [29, 26, 28], pois ele influencia no número de unidades RBF adicionadas pelo DDA durante o treinamento. Se o treinamento for feito com um θ^- muito pequeno, o algoritmo criará muitas unidades RBF, e a rede acabará perdendo sua generalização, o que acarretará em *overfitting*. Uma proposta para otimização do algoritmo DDA através do θ^- [7, 29] é dividir os dados de treinamento em dados de treinamento e validação. Primeiro, a rede RBF-DDA é treinada com um θ^- inicial, e os dados de validação são utilizados para medir a generalização da rede durante o treinamento. Nesse treinamento, o valor do θ^- é diminuído até chegar num θ^- ótimo. Depois a rede é treinada novamente com o valor do θ^- igual ao θ^- ótimo. Assim, espera-se melhorar o desempenho da RBF-DDA em alguns tipos de problemas [29].

1.2 Máquinas de vetor suporte

Máquinas de vetor suporte (SVM) [1, 8, 44] constituem uma recente técnica para classificação e regressão que tem conseguido uma precisão notável em vários problemas importantes [41, 9, 35, 34, 10]. O algoritmo de aprendizagem SVM pode ser usado para construir diversos tipos de máquinas de aprendizagem, como por exemplo máquinas de aprendizado polinomial, RBFs e MLPs, e o número de unidades escondidas em cada um desses casos é automaticamente determinado pelo algoritmo de aprendizagem SVM.

Basicamente, o SVM é um algoritmo linear que constrói hiperplanos, com o objetivo de encontrar hiperplanos ótimos, ou seja, hiperplanos que maximizem a margem de separação das classes, para separar os padrões de treinamento em diferentes classes. A Fig. 14 mostra dois exemplos de hiperplanos. No lado esquerdo (Fig.14 (a)), está um hiperplano com uma margem de separação pequena, e no lado direito (Fig.14 (b)) está ilustrado um hiperplano com uma margem de separação maior, e com isso, é esperado que esse obtenha uma melhor generalização para a classificação dos padrões.

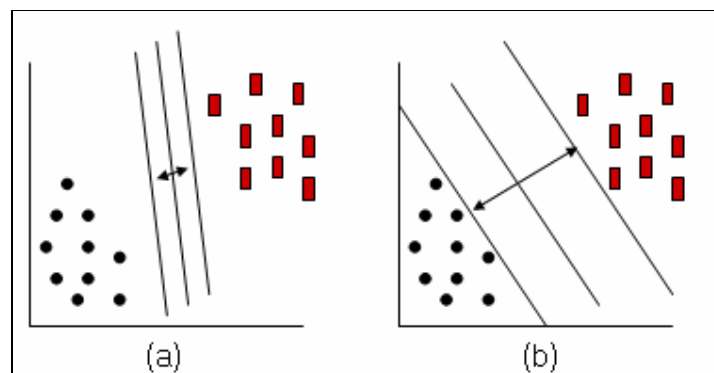


Figura 14: (a) Hiperplano com uma margem de separação menor (b) Hiperplano com uma margem de separação maior

A técnica SVM minimiza a equação 1.14 submetida à condição especificada na equação 1.15.

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^l \xi_i \quad (1.14)$$

$$y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \quad (1.15)$$

$$\xi_i \geq 0.$$

Os vetores de treinamento x_i são mapeados num espaço de dimensionalidade maior através da função ϕ . Depois, a SVM encontra um hiperplano linear de separação com a margem máxima nesse espaço de dimensão maior. Um Kernel $K(\bar{x}, \bar{y})$ é o produto interno em algum espaço de características, $K(\bar{x}, \bar{y}) = \phi^T(\bar{x})\phi(\bar{y})$. Diferentes kernels têm sido propostos na literatura [35, 34, 10, 46]. Alguns exemplos 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): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \gamma > 0$

Neste trabalho, foi utilizado o kernel função de base radial (RBF), que é o kernel usado mais frequentemente.

SVMs com kernel RBF possuem dois parâmetros, chamados C (o parâmetro de penalidade do termo de erro ($C > 0$)) e γ , a largura dos kernels RBF. Esses parâmetros têm grande influência na classificação dos dados e na generalização dos mesmos e, portanto, esses valores devem ser cuidadosamente selecionados de acordo com o problema. Neste trabalho, a seleção do modelo é feito usando validação cruzada (*10-fold*) [11] nos dados de treinamento. O método de escolha dos parâmetros C e γ foi o desempenho, onde pares de (C, γ) são testados e o que produzir melhor precisão é o selecionado [19]. Um método prático para identificar bons parâmetros consiste em treinar exponencialmente seqüências crescentes de C e γ . Utilizamos a seqüência $C = 2^{-5}, 2^{-3}, \dots, 2^{15}$, e $\gamma = 2^{-15}, 2^{-13}, \dots, 2^3$ [12].

1.3 Regra do vizinho mais próximo

A regra de classificação dos K vizinhos mais próximos (KNN) [1, 39, 30] é um método de classificação supervisionado e não-paramétrico, onde um padrão é dito pertencer a uma classe de acordo com a quantidade de vizinhos que pertençam a essa classe, conforme um critério de distância (distância Euclidiana, geralmente).

O método de classificação KNN não possui processamento na fase de treinamento, por isso a idéia da classificação é bastante simples: primeiro deve-se armazenar os padrões de treinamento (essa seria a fase de treinamento do KNN). Depois, para um novo padrão, é calculada a distância Euclidiana desse padrão para todos os padrões de treinamento. O padrão de treinamento que fornecer a menor distância Euclidiana irá determinar a classe desse novo padrão.

Essa classificação é para o caso em que o parâmetro K é igual a 1 (NN). Se o K for maior que um, por exemplo, $K=3$, são considerados três vizinhos do novo padrão, ou seja, são analisadas as três menores distâncias do novo padrão para os padrões de treinamento. A classe que tiver o maior número de padrões entre essas distâncias é que irá determinar a classe do novo

padrão. A Fig. 15 ilustra esse processo de classificação. Para classificar um ponto, primeiro toma-se os K-vizinhos mais próximos dele e, dentro desse conjunto, encontra-se a classe mais representativa [16]. Na Fig. 15, o ponto desconhecido 1 será classificado como classe B e o ponto desconhecido 2 será classificado como classe A.

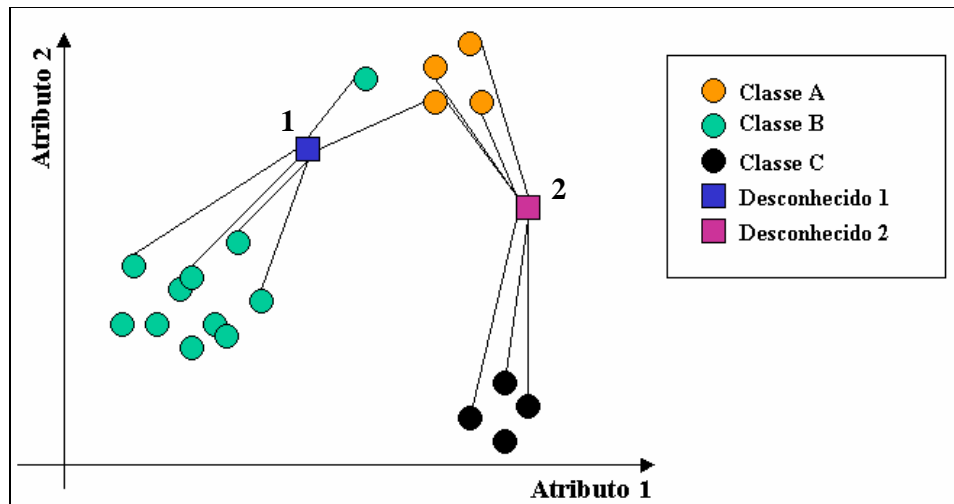


Figura 15. Esquema de classificação pelo Método KNN

Na escolha do parâmetro K deve-se ter cuidado para escolher um número ímpar, pois um número par poderia causar conflito, quando um ponto tivesse o mesmo número de vizinhos de cada classe, por exemplo.

A classificação KNN é bem simples porém tem a desvantagem de ter que armazenar todos os dados na memória, ou seja, gerar classificadores complexos, por causa do armazenamento.

1.4 Regra do vizinho mais próximo com seleção de padrões (NNSRM)

O método NNSRM [23, 22] visa a minimizar o risco estrutural da técnica KNN. Esse método foi proposto recentemente por Bilge Karaçali e Hamid Krim na revista *IEEE Transactions on Neural Networks* [23] e têm obtido resultados de uma redução de custo computacional em relação ao KNN e SVM.

O método consiste em diminuir a quantidade de padrões de treinamento armazenados, porém sem prejudicar o desempenho da classificação. Ele é baseado na minimização estrutural do risco, técnica que também deu origem às SVMs

1.4.1 Construção do classificador

Seja $I = \{1, 2, \dots, n\}$ o conjunto de todos os padrões de treinamento. O objetivo desse método é criar um conjunto $J \subset I$ que resulte em uma classificação igual ou aproximada (no conjunto de teste), caso o conjunto fosse o I .

O algoritmo 2 mostra os passos para a construção do classificador NNSRM para 2 classes distintas.

Algoritmo 2. Algoritmo de construção do classificador NNSRM para 2 classes distintas

- 1: define-se o conjunto $D = \{d_{(1)}, d_{(2)}, \dots, d_{(k)}\}$, onde $d_{(k)}$ é o elemento de ordem k do conjunto D ;
- 2: inicializa $J = \emptyset, k=1$;
- 3: enquanto $R^{emp}(f_J) > 0$ faça
- 4: encontrar x_i e x_j , tal que $\rho(x_i, x_j) = d_{(k)}, y_i = -1, y_j = 1$;
- 5: se $\{i, j\} \not\subset J$, atualizar $J \leftarrow J \cup \{i, j\}$;
- 6: incrementar $k \leftarrow k + 1$.

Inicialmente, é calculado o conjunto de todas as distâncias Euclidianas dos padrões x_i e x_j , tal que x_i e x_j são de padrões diferentes [passo 1 do algoritmo 2]. Depois, inicializa-se o conjunto J como conjunto vazio e o número inteiro k com o valor 1 [passo 2 do algoritmo 2]. A partir daí, calcula-se o erro no conjunto de treinamento ($R^{Emp}(f_J)$), com os padrões do conjunto J como padrões de teste. Então, enquanto esse erro for diferente de zero [passo 3], repete-se os próximos passos: procura-se os dois padrões x_i e x_j , de padrões diferentes, tal que a distância Euclidiana entre eles é igual a $d_{(k)}$ [passo 4]. E, se i e j não pertencerem à J , eles são adicionados ao conjunto J [passo 5]. O k é então incrementado uma unidade [passo 6]. Quando o erro no conjunto de treinamento for zero, a construção do classificador é finalizada. Então, o conjunto J é utilizado como conjunto de treinamento e o conjunto de teste é utilizado para verificar a generalização do classificador, através da técnica de KNN.

O pior caso é quando $J = I$, pois não altera nada em relação ao KNN, e ainda gasta tempo de execução do algoritmo. Porém, geralmente o conjunto J é bem menor que o conjunto I , diminuindo, assim, o custo computacional de classificação de um novo padrão.

A Fig. 16 ilustra, graficamente, o algoritmo de construção do classificador NNSRM para duas classes. Os padrões de cores azul e rosa são os selecionados, entre todos os padrões de treinamento, para o conjunto J .

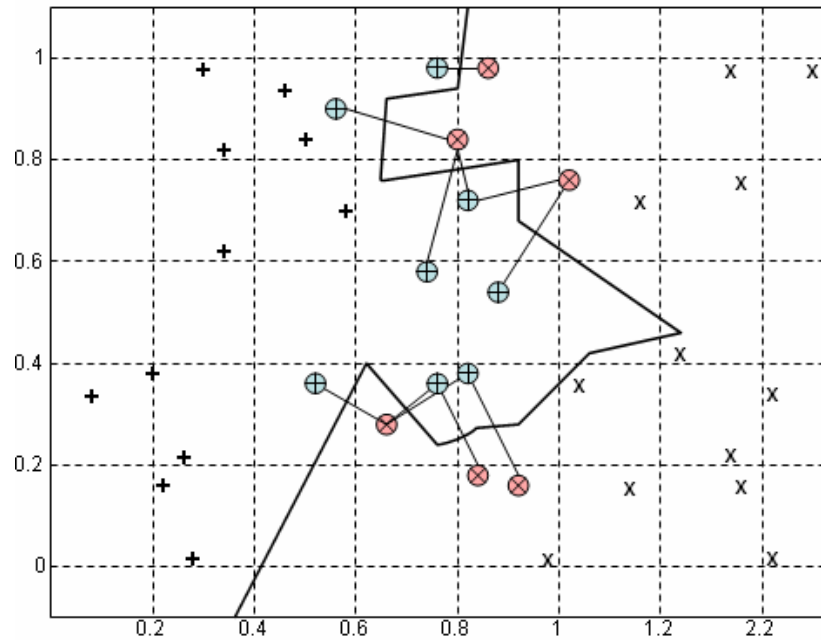


Figura 16. Ilustração do classificador NNSRM para duas classes

Esse método também se aplica a problemas com mais de duas classes. Neste trabalho, o algoritmo foi implementado para sete classes diferentes. O algoritmo 3 mostra os passos para a construção do classificador NNSRM para mais de 2 classes distintas.

Algoritmo 3. Algoritmo de construção do classificador NNSRM para mais de 2 classes distintas

Considere:

- $X_T = \{x_1^{m_1}, x_2^{m_1}, \dots, x_{l_1}^{m_1}, x_1^{m_2}, \dots, x_{l_M}^{m_M}\}$ é o conjunto de treinamento, onde l_i é o número de padrões da classe m_i no conjunto de treinamento, para $i = 1, \dots, M$.
- $f_R(x_0) = m_i$ é a notação para o classificador, onde $R \subset X_T$.
- $d_{(k)}^{(i,j)}$ são todos os pares de distância dos pontos em m_i e m_j , para $i = 1, \dots, M - 1$, $j = i + 1, \dots, M$ e $k = 1, \dots, l_i l_j$.

- 1: Inicializa $R = k^{(i,j)} = 1$ para todo (i, j) .
- 2: Enquanto $R^{emp}(f_R) > 0$ faça
- 3: Para todo (i, j) que foram classificados erroneamente
- 4: Achar os pontos x^{m_i} e x^{m_j} onde $\rho(x^{m_i}, x^{m_j}) = d_{k^{(i,j)}}^{(i,j)}$
- 5: Incrementar $k^{(i,j)} \leftarrow k^{(i,j)} + 1$
- 6: Se $\{x^{m_i}, x^{m_j}\} \notin R$, atualizar $R \leftarrow R \cup \{x^{m_i}, x^{m_j}\}$
 Se não, ir para passo 4.

Nesse algoritmo, o conjunto R é o conjunto dos padrões que irão ser armazenados. Inicialmente ele é inicializado com 1 [passo 1 do algoritmo 3]. Depois, enquanto o erro no conjunto de

treinamento for diferente de zero, para cada par de classes classificadas erroneamente [passo 3], o algoritmo processa da mesma maneira que o algoritmo para apenas duas classes distintas.

Primeiro ele acha dois padrões de classes distintas (um de cada classe analisada) que possuam a menor distância do conjunto ρ [passo 4]. Depois ele incrementa o contador [passo 5], e caso os dois padrões não pertençam ao conjunto R, ele serão incluídos no mesmo. Note que, para $M = 2$, este algoritmo coincide com o algoritmo de construção do classificador NNSRM para 2 classes distintas.

Capítulo 2

Comparação de classificadores

Este capítulo apresenta técnicas usadas para comparar classificadores em um determinado problema (ou conjunto de dados). A principal técnica utilizada neste trabalho é a validação cruzada [11, 13], que é comparada com outras como, por exemplo, *holdout* [40, 32], e *leave-one-out* [12, 32], explicando suas definições, sua importância e como foi utilizada neste projeto.

2.1 Holdout

O método holdout [40, 32] é um tipo bem simples de método para teste de classificadores. Nele, a base de dados é dividida em dois conjuntos: conjunto de treinamento e conjunto de teste. O conjunto de treinamento fornece os dados para o treinamento da técnica utilizada e o conjunto de teste fornece dados novos, para testar a generalização da técnica. Devem ser feitos vários testes (no caso de MLPs, com diferentes inicializações dos pesos), acumulando-se os erros obtidos. É calculada, então, uma média desses erros para se avaliar o resultado.

Uma vantagem desse método é que ele é simples e computacionalmente rápido, porém sua avaliação pode ter uma alta variância. O resultado da avaliação pode depender, por exemplo, de que ponto terminaram os dados de treinamento e começaram os dados de teste, ou seja, da quantidade de padrões existente em cada conjunto. A avaliação depende, então, da maneira que é feita esta divisão. Outro fator que influencia na avaliação é a quantidade de padrões existentes de cada classe no conjunto de treinamento. Por exemplo, uma classe A com uma grande quantidade de padrões deverá “influenciar” mais o resultado final, ao contrário de uma classe B com poucos padrões do seu tipo, ou seja, a rede treinada (se estiver utilizando redes neurais) terá uma melhor generalização para os dados da classe A do que para a classe B.

2.2 Validação cruzada

Nesse método os dados são divididos em uma parte para treinamento e outra parte para teste, porém se diferencia do holdout por continuar o treinamento para todos os padrões, ou seja, os conjuntos de treinamento e teste variam em um mesmo conjunto total de padrões. Um exemplo de validação cruzada [11, 13] consiste em dividir o conjunto total de padrões em k grupos de tamanhos aproximadamente iguais (*k-fold cross validation*). Com isso, o treinamento é realizado

k vezes, a cada vez deixando um dos grupos para teste. Ou seja, se $K = 5$, a rede será treinada cinco vezes, na primeira vez o primeiro grupo será usado para teste e os outros quatro serão usados para treinamento. Na segunda vez, o segundo grupo será para teste e os outros quatro serão para treinamento, e assim sucessivamente. Uma demonstração gráfica está na Fig. 17.

A cada treinamento é calculado um erro de classificação, e, no final, será calculada a média e o desvio padrão desses erros para se chegar a um resultado final. Neste trabalho, os dados serão divididos em grupos de dez, o que na literatura é denominado validação cruzada *10-fold* [11].

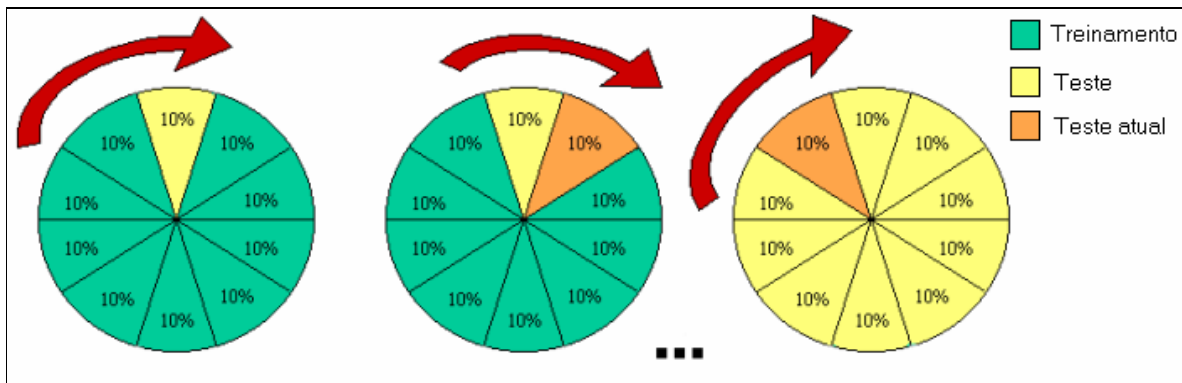


Figura 17. Demonstração gráfica do procedimento de validação cruzada *10-fold*

A vantagem de se usar a validação cruzada ao invés do método *holdout* é que nela o treinamento é feito com todos os dados, e por isso gera um resultado mais confiável, uma vez que no método *holdout* os dados são divididos e essa divisão pode não gerar um resultado representativo dos padrões. Por exemplo, questões como ‘Quais dados deve-se colocar para treinamento e quais deve-se colocar para teste?’ ou ‘Quantos dados deve ter o conjunto de treinamento e quantos dados deve ter o conjunto de teste?’ devem ser bem estudadas no *holdout* porque não existe um padrão de resposta para essas perguntas, variando para cada problema.

O método *holdout*, porém é mais simples, e pode ser bem utilizado quando o número de padrões existente é grande e bem dividido entre as classes. Já, quando o número de padrões é pequeno, é aconselhável utilizar a validação cruzada.

2.3 Leave-one-out

O método *leave-one-out* [12, 32] é uma simplificação da validação cruzada. Nele, os N padrões são divididos em dois conjuntos, um com 1 elemento e o outro com N-1 elementos. O método vai modificando os conjuntos como no método da validação cruzada. Por exemplo, se a base de dados possui cem padrões, primeiro treina-se com 99 padrões e testa com o restante. Depois o treinamento é feito com outros 99 padrões e o teste é feito com o restante. Ao término foram realizados 100 treinamentos, gerando 100 valores de erros. Então é calculada a média desses erros para gerar o erro final. Esse método possui a mesma vantagem de confiabilidade da validação cruzada, pois com uma base de 100 dados, são realizados cem treinamentos, e todos os padrões passam pelas fases de treinamento e de teste.

Capítulo 3

Ferramentas de simulação de aprendizagem de máquina

Diversas ferramentas de simulação têm sido desenvolvidas para o uso de técnicas de aprendizagem de máquina. Este capítulo apresenta, brevemente, alguns simuladores que foram utilizados, como o SNNS [37] para as redes neurais e o LIBSVM [19] para a técnica de SVM. Este capítulo também descreve as implementações realizadas (em Java) para as simulações das técnicas KNN e NNSRM.

3.1 SNNS

Para simulação das redes neurais MLP e RBF-DDA foi utilizado o software SNNS (Stuttgart Neural Network Simulator), versão 4.2 [37]. Ele foi desenvolvido no Instituto para Sistemas de Alto Desempenho Paralelos e Distribuídos (Institut für Parallele und Verteilte Höchstleistungsrechner — IPVR) na universidade de Stuttgart, em 1989. Essa ferramenta oferece um ambiente de simulação para pesquisas e testes com redes neurais e dá suporte a vários tipos de arquitetura e sistemas operacionais, como Windows, Linux e HP-UX, sendo *freeware*.

O simulador SNNS consiste de quatro componentes principais: o kernel do simulador, a interface gráfica com o usuário, interface de execução em batch (bachman) e o compilador de redes snns2c [21]. O kernel é o responsável por todas as operações sobre a estrutura de dados das redes neurais, e a interface gráfica trabalha de acordo com o kernel, criando uma representação gráfica das redes neurais geradas por ele.

A Fig. 18 mostra a interface gráfica do painel inicial (SNNS Manager Panel). Através dele, pode-se ter acesso à todas as funcionalidades do simulador, como a criação das redes neurais, acesso aos arquivos gerados, e outras funções.

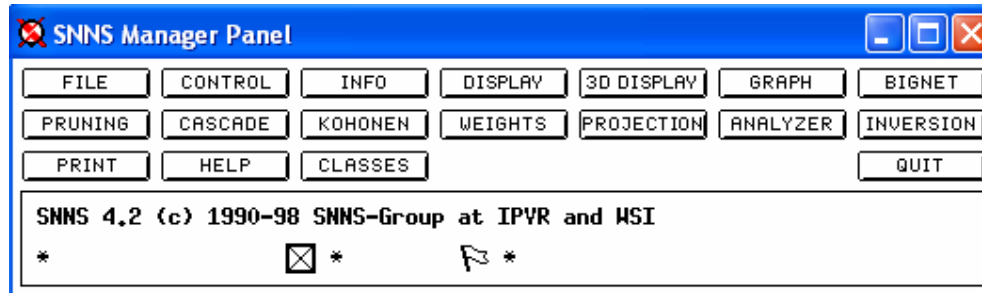


Figura 18. Painel inicial do SNNS

3.1.1 Criação da rede

O SNNS permite a criação de diversos tipos de redes neurais. Aqui são explicados como construir as redes que foram utilizadas.

Para este trabalho, foram criadas duas redes neurais: uma do tipo MLP e outra do tipo RBF. Ambas são criadas através do menu BIGNET -> general, como mostra a Fig. 19. Outros tipos de rede podem ser vistos neste menu também, mas para nosso propósito é explicado apenas o tipo de rede geral (*general*).

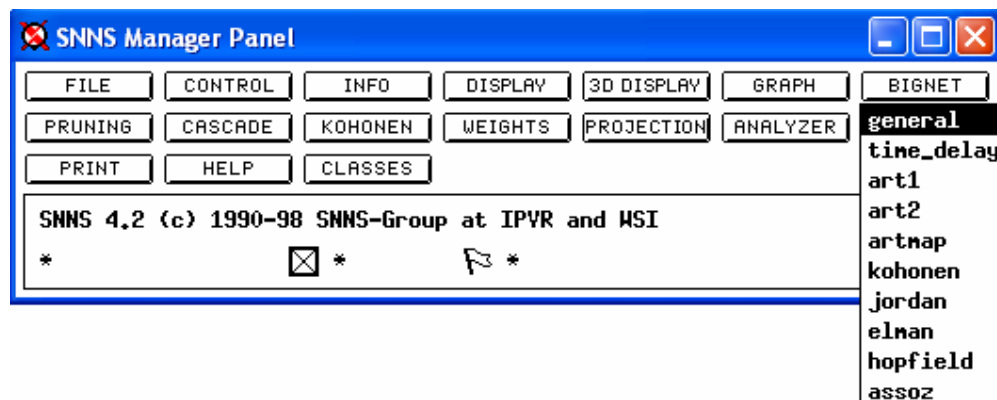


Figura 19. Menu para criação das redes neurais pelo SNNS

Depois de escolher o tipo de rede geral, o SNNS abre um outro painel, ilustrado na Fig. 20, onde se define as propriedades da rede como a topologia das camadas e o tipo de conexões.

Os botões *TYPE* e *POS* definem o tipo e a posição dos neurônios da rede. Na rede MLP foram criados 10 neurônios de entrada, 5 neurônios na camada escondida e 7 neurônios de saída. Já na rede RBF apenas foram criados 10 neurônios na camada de entrada e 7 neurônios na camada de saída, pois, nessa rede, os neurônios da camada escondida são definidos durante o treinamento. O botão *FULL CONNECTION* fornece a opção de criar uma rede completamente conectada, ou seja, de serem criadas todas as conexões entre todas as camadas (essa opção só foi usada para MLPs). Existem diversas outras funcionalidades nesse painel, porém não se entra em detalhes aqui por fugirem do escopo do projeto. O botão *CREATE NET* finaliza então a definição da rede construindo-a de acordo com as opções escolhidas.

SNNS BigNet (General Type)

Plane	Current Plane	Edit Plane
Plane:	<input type="text"/>	
Type:	<input type="text"/>	input
No. of units in x-direction:	<input type="text"/>	<input type="text"/>
No. of units in y-direction:	<input type="text"/>	<input type="text"/>
z-coordinates of the plane:	<input type="text"/>	<input type="text"/>
Rel. Position:	<input type="text"/>	right

Edit Plane: ENTER INSERT OVERWRITE DELETE
 PLANE TO EDIT TYPE POS
 Current plane: [Left] [Right] [Up] [Down]

	Current Link		Edit Link	
	Source	Target	Source	Target
Plane	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Cluster				
Coordinates				
x:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
y:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
width :	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
height:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Unit				
Coordinates				
x:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
y:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Move				
dx:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
dy:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Edit Link: ENTER OVERWRITE LINK TO EDIT DELETE
 FULL CONNECTION SHORTCUT CONNECTION
 Current Link: [Left] [Right] [Up] [Down]

CREATE NET DONE CANCEL

Figura 20. Painel para criação das redes neurais pelo SNNS

3.1.2 Visualização da rede

Após criada a rede, o SNNS fornece a opção de visualização da mesma. Essa opção pode ser acessada pelo painel principal, opção *DISPLAY*, como indicado na Fig. 21. A Fig. 22 ilustra a visualização da rede MLP criada para este trabalho pelo SNNS. Ela possui 10 neurônios na camada de entrada, 5 neurônios na camada escondida e 7 neurônios na camada de saída, e é completamente conectada, como já definido. A rede RBF criada para este trabalho está apresentada na Fig. 23. Pode-se observar que ela não é conectada, pois ela não está completa. Ela ficará completa após o treinamento, quando se definir os neurônios da camada escondida.

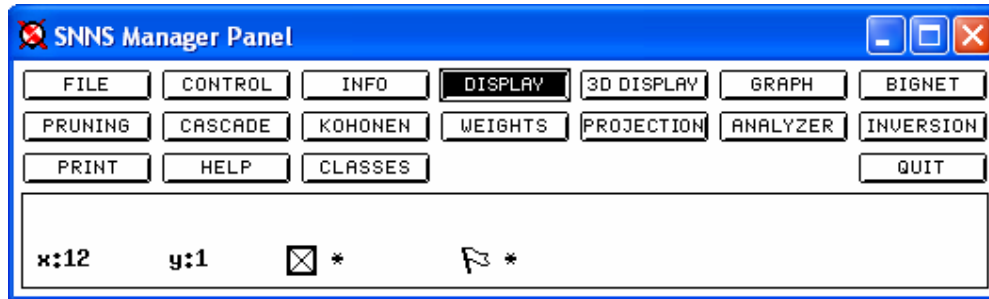


Figura 21. Opção de visualização da rede neural pelo SNNS

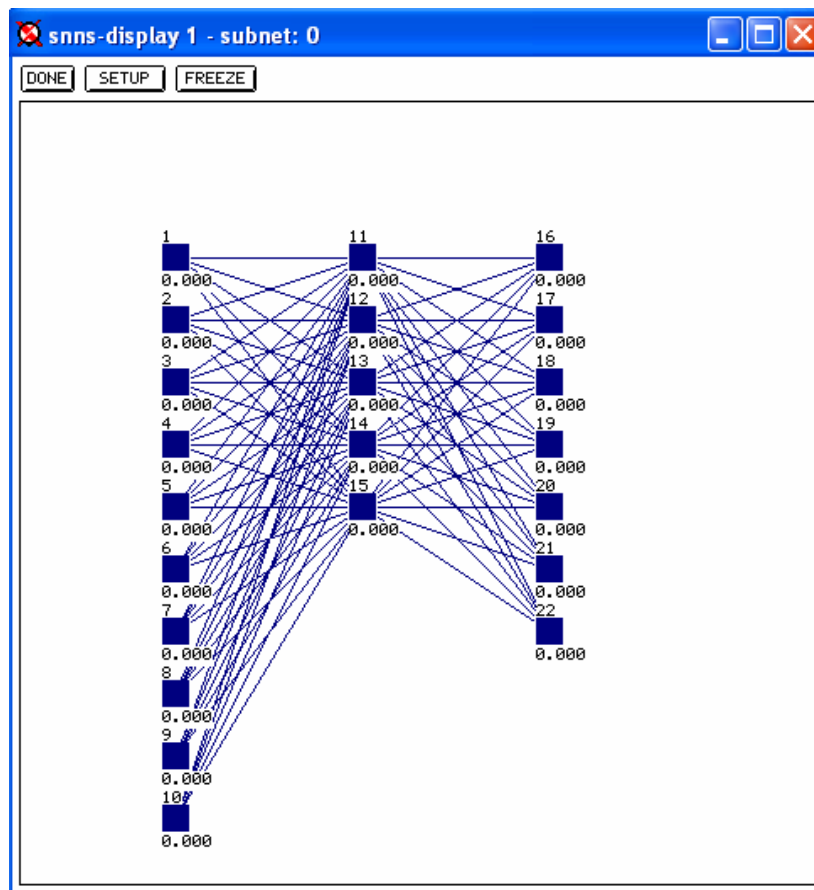


Figura 22. Visualização da rede MLP criada no SNNS

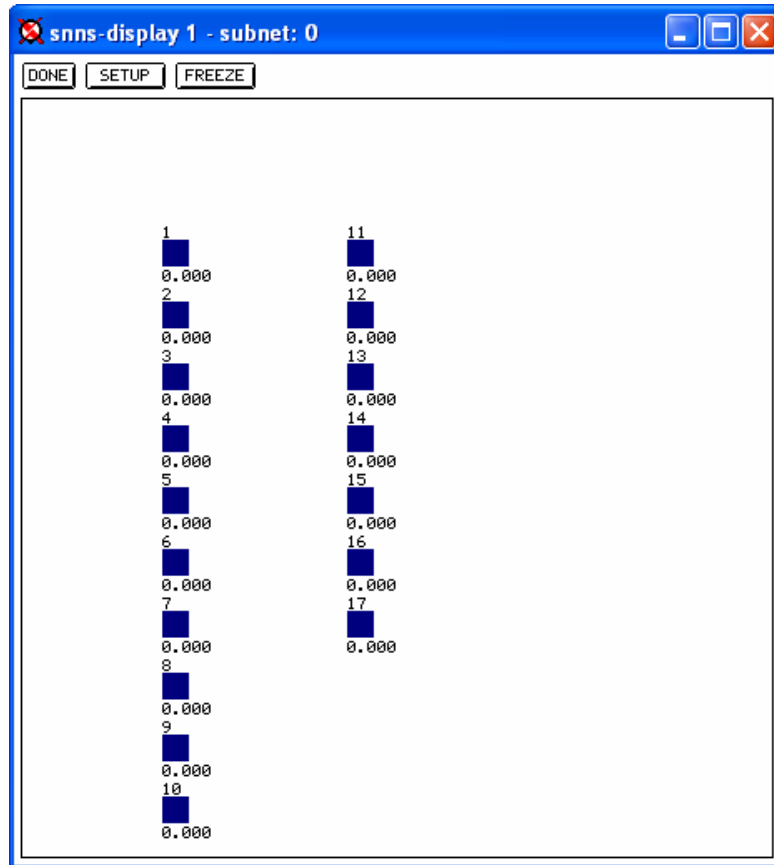


Figura 23. Visualização da rede RBF criada no SNNS

3.1.3 Arquivos

O simulador SNNS suporta cinco tipos de arquivos, porém os principais e utilizados no nosso projeto são:

- NET : esses arquivos possuem informações de definição da rede neural. Ele contém informações da topologia da rede e regras de aprendizado. Arquivos desse tipo possuem extensão .net.
- PAT: são arquivos de padrões. Neles ficam os padrões de treinamento, validação e teste. Arquivos desse tipo possuem extensão .pat.
- RES: esses são os arquivos de saída, onde ficam os resultados, diferentes para cada tipo de problema. Arquivos desse tipo possuem extensão .res.

Para criar, abrir ou carregar esses arquivos, o SNNS tem a opção *FILE* no painel principal. Essa opção está ilustrada na Fig. 24. Ela leva à janela de manipulação de arquivos (Fig. 25). Nessa janela o usuário pode escolher o tipo e o nome do arquivo, o diretório onde ele está ou será gravado, assim como pode escolher a opção de carregar ou salvar o arquivo. Após essas definições, a opção *DONE* deverá ser escolhida para finalizar a operação.

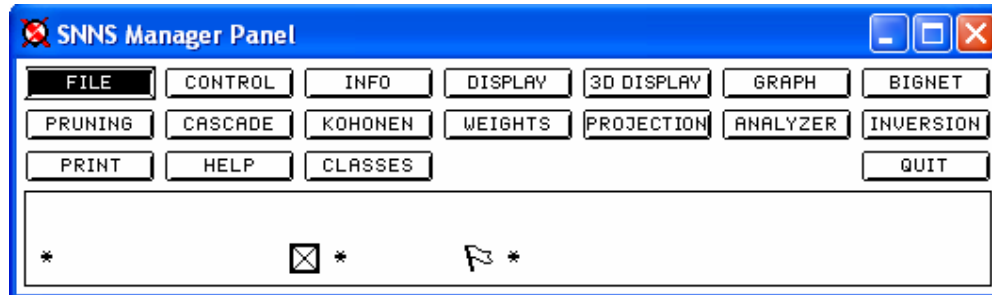


Figura 24. Opção de arquivos do SNNS

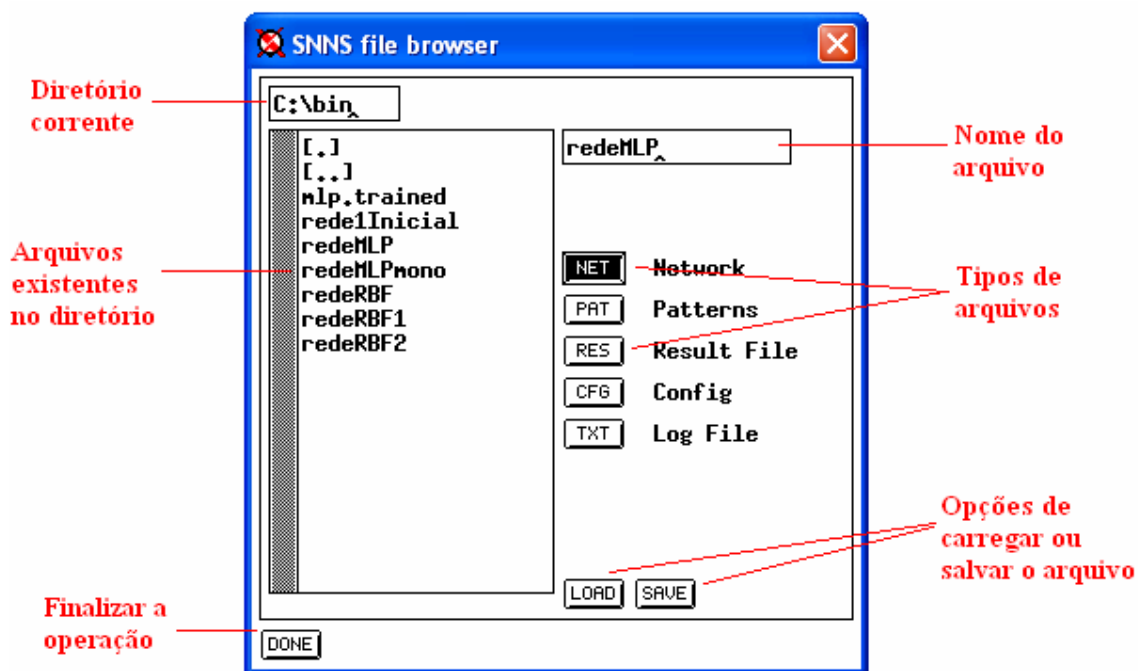


Figura 25. Janela de manipulação de arquivos do SNNS

3.2 LIBSVM

A ferramenta de simulação LIBSVM [19] é um software integrado para testes com máquinas de vetor suporte (SVMs). Ele possui uma interface simples, que pode ser facilmente utilizada com outros programas [19]. Duas grandes vantagens desse software é que ele já possui métodos que normalizam os dados e que implementam a técnica de validação cruzada.

A Fig. 26 mostra um applet que vem no software demonstrando a classificação e regressão através do LIBSVM. Para usá-lo deve-se clicar na tela preta para marcar os pontos; e para mudar a cor, significando mudar o tipo, deve-se clicar em *Change*. Esse applet permite a configuração de alguns parâmetros, tais como:

- -t : tipo da função kernel (padrão 2). O valor 2 representa o tipo função de base radial, porém existem outras opções como função linear (0), função polinomial (1) e função sigmóide (3).

- -c: define o parâmetro C do SVM.
- -p: define o parâmetro γ do SVM.
- -v n: divide os dados em n partes e calcula a classificação pela validação cruzada n-fold.

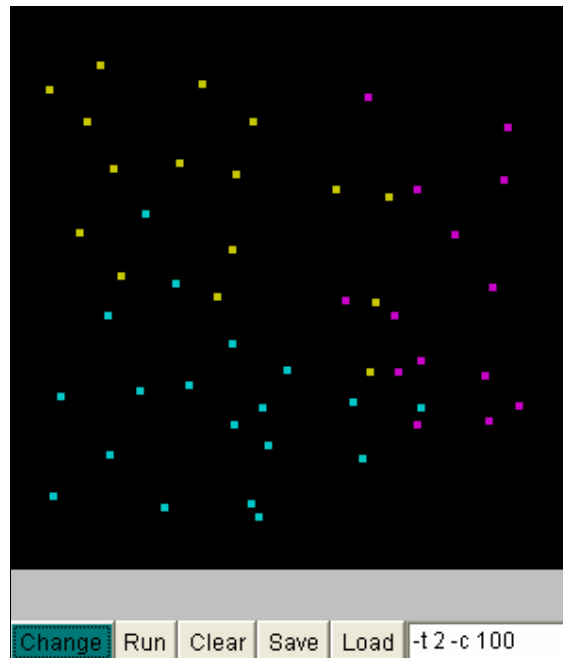


Figura 26. Applet do LIBSVM

Ao se definir os pontos e os parâmetros basta clicar em *Run* e o applet mostra a representação gráfica da classificação. Esta representação está ilustrada na Fig. 27.

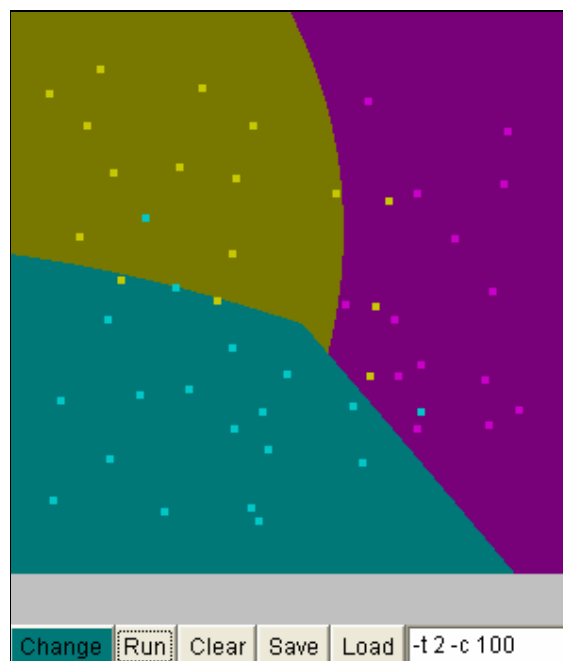
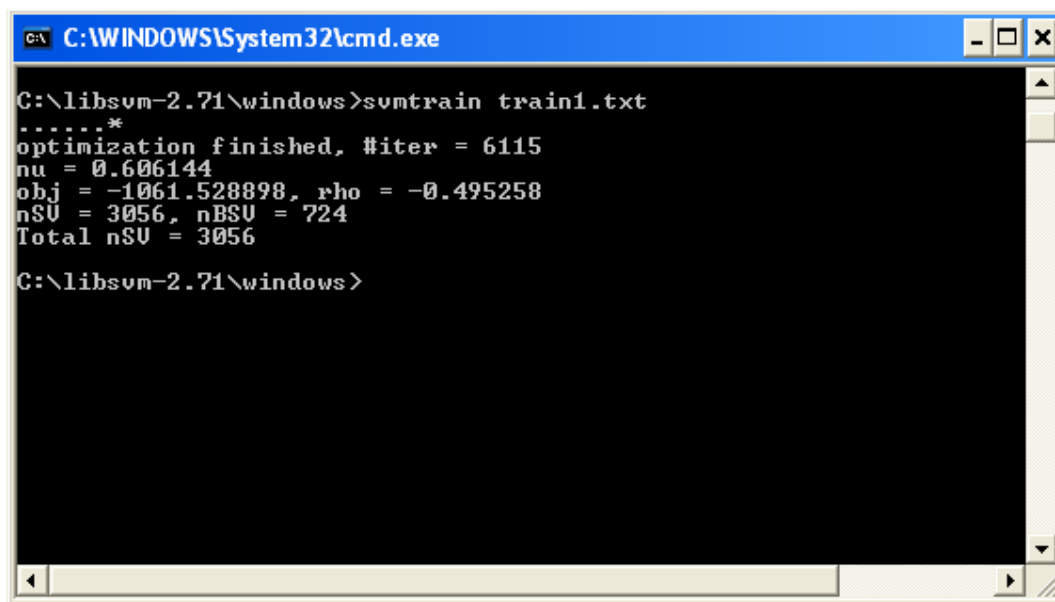


Figura 27. Representação gráfica da classificação pelo applet do LIBSVM

Como mostra a Fig. 27, o applet também fornece as opções de limpar a tela (*Clear*), salvar uma simulação (*Save*) ou abrir uma já existente (*Load*).

Para a simulação com os arquivos de dados reais, o procedimento é feito através do prompt de comando. O LIBSVM vem com alguns aplicativos para tratamento dos dados e para a simulação do SVM. É recomendado também, que se use o software Python juntamente com o LIBSVM, pois com ele pode-se fazer a seleção dos melhores parâmetros, C e γ , através do programa *grid.py* que vem em sua instalação.

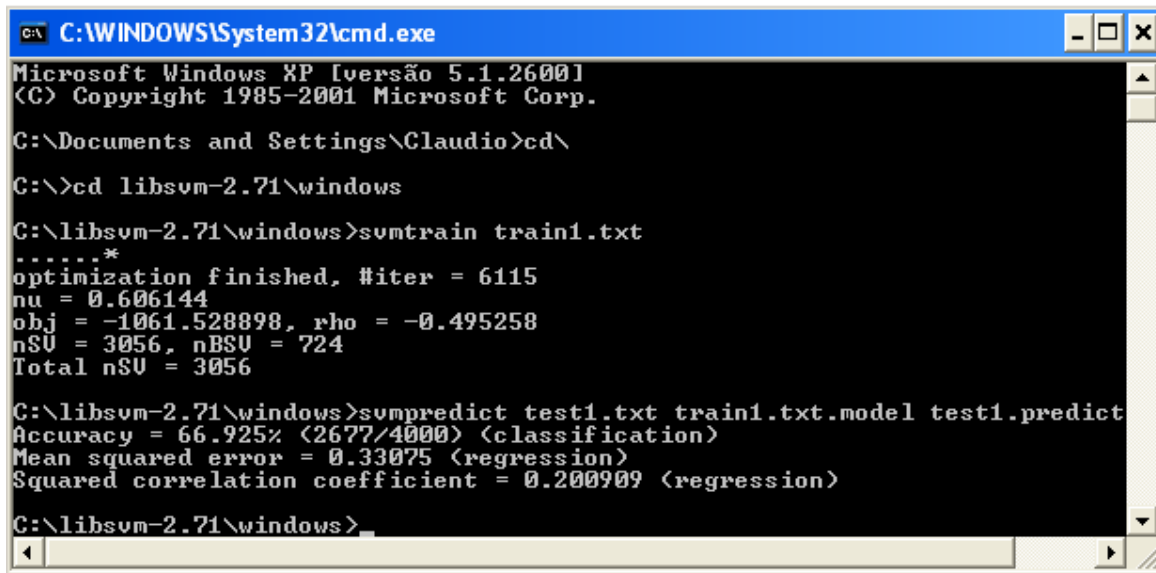
Para o treinamento do SVM o LIBSVM disponibiliza o aplicativo *svmtrain*. A Fig 28 mostra a execução desse aplicativo.



```
C:\WINDOWS\System32\cmd.exe
C:\libsvm-2.71\windows>svmtrain train1.txt
.....*
optimization finished, #iter = 6115
nu = 0.606144
obj = -1061.528898, rho = -0.495258
nSU = 3056, nBSU = 724
Total nSU = 3056
C:\libsvm-2.71\windows>
```

Figura 28. Execução do aplicativo *svmtrain* do LIBSVM

O LIBSVM também disponibiliza um aplicativo para normalizar os dados antes do treinamento, ou seja, transformar os dados em valores entre dois limites, máximo e mínimo, escolhidos pelo usuário. Nesse trabalho, os dados ficaram entre os valores 0 e 1. Estudos mostram que normalizar os dados antes do treinamento fornece resultados melhores do que usar dados brutos [19]. Esse aplicativo é o *svmscale*. Para a classificação, o LIBSVM disponibiliza o aplicativo *svmpredict*. A Fig. 29 mostra o treinamento dos dados sem normalização e o resultado da classificação, e a Fig. 30 mostra a seqüência de normalização dos dados, treinamento e classificação.



```

C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [versão 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Claudio>cd\

C:\>cd libsvm-2.71\windows

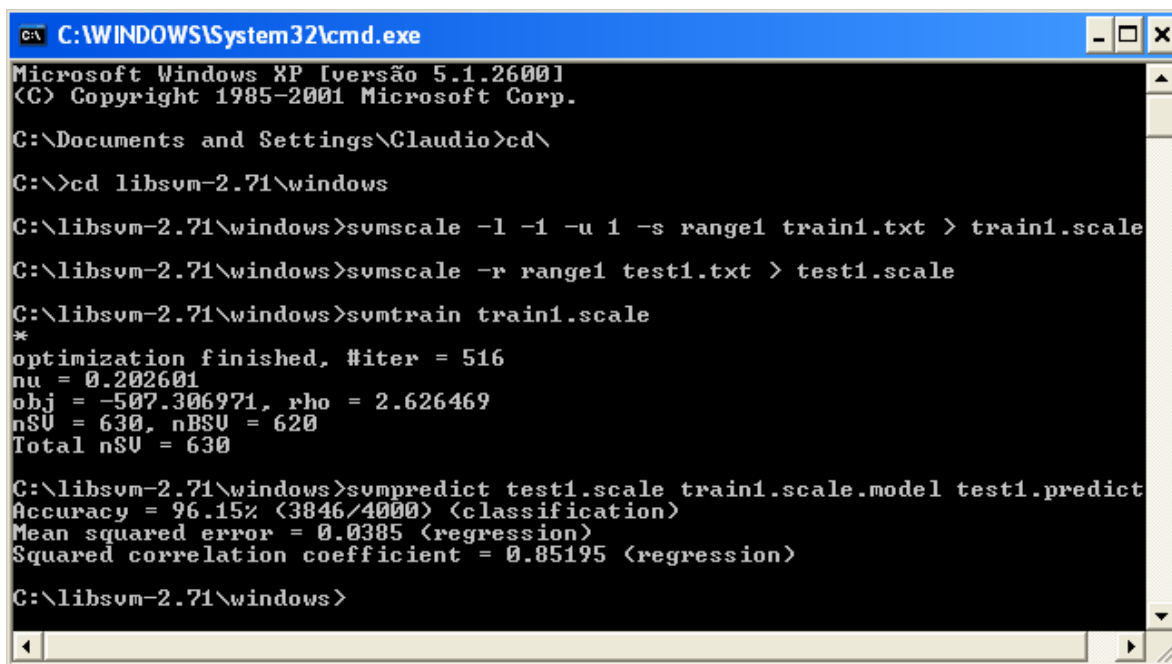
C:\libsvm-2.71\windows>svmtrain train1.txt
*****
optimization finished, #iter = 6115
nu = 0.606144
obj = -1061.528898, rho = -0.495258
nSV = 3056, nBSV = 724
Total nSV = 3056

C:\libsvm-2.71\windows>svmpredict test1.txt train1.txt.model test1.predict
Accuracy = 66.925% (2677/4000) (classification)
Mean squared error = 0.33075 (regression)
Squared correlation coefficient = 0.200909 (regression)

C:\libsvm-2.71\windows>

```

Figura 29. Treinamento dos dados sem normalização e resultado da classificação pelo LIBSVM



```

C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [versão 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Claudio>cd\

C:\>cd libsvm-2.71\windows

C:\libsvm-2.71\windows>svmscale -l -1 -u 1 -s range1 train1.txt > train1.scale
C:\libsvm-2.71\windows>svmscale -r range1 test1.txt > test1.scale
C:\libsvm-2.71\windows>svmtrain train1.scale
*
optimization finished, #iter = 516
nu = 0.202601
obj = -507.306971, rho = 2.626469
nSV = 630, nBSV = 620
Total nSV = 630

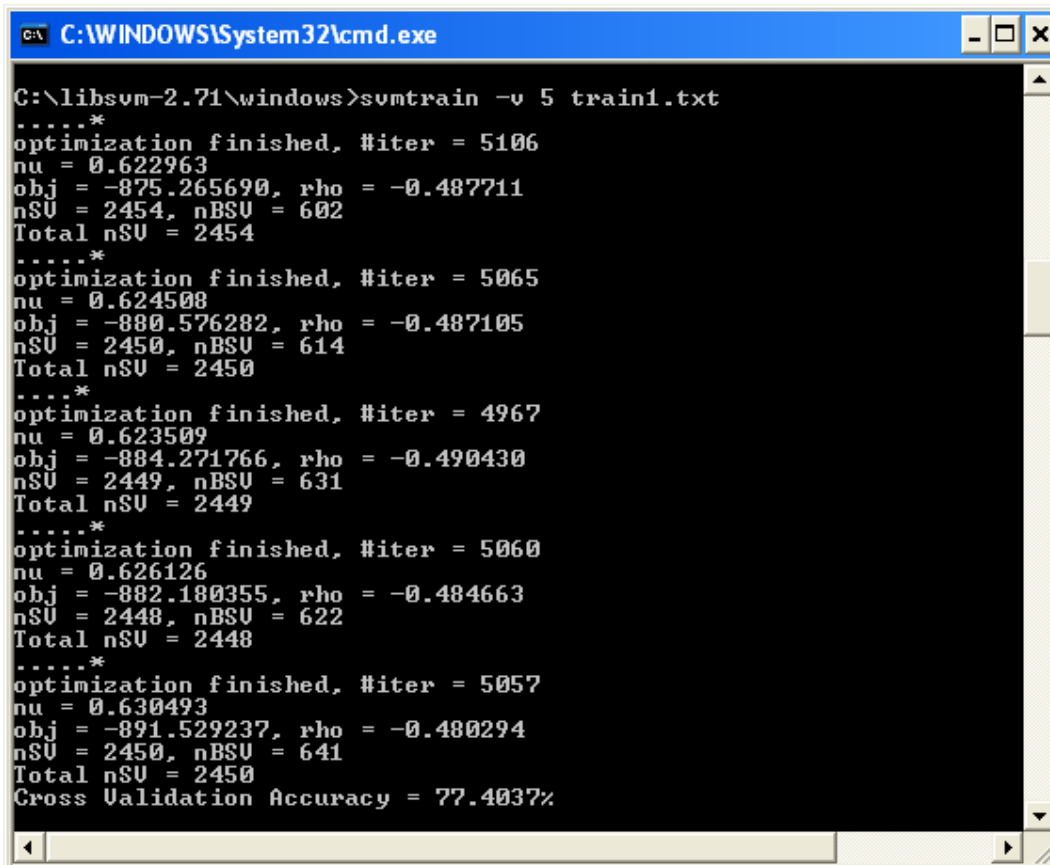
C:\libsvm-2.71\windows>svmpredict test1.scale train1.scale.model test1.predict
Accuracy = 96.15% (3846/4000) (classification)
Mean squared error = 0.0385 (regression)
Squared correlation coefficient = 0.85195 (regression)

C:\libsvm-2.71\windows>

```

Figura 30. Treinamento dos dados normalizados e resultado da classificação pelo LIBSVM

O simulador LIBSVM fornece também a opção do treinamento utilizando o método validação cruzada. A Fig 31 mostra um treinamento usando validação cruzada (5-fold). Observe que ele mostra 5 treinamentos antes de dar o resultado final.



```
C:\WINDOWS\System32\cmd.exe

C:\libsvm-2.71\windows>svmtrain -v 5 train1.txt
*****
optimization finished, #iter = 5106
nu = 0.622963
obj = -875.265690, rho = -0.487711
nSU = 2454, nBSU = 602
Total nSU = 2454
*****
optimization finished, #iter = 5065
nu = 0.624508
obj = -880.576282, rho = -0.487105
nSU = 2450, nBSU = 614
Total nSU = 2450
*****
optimization finished, #iter = 4967
nu = 0.623509
obj = -884.271766, rho = -0.490430
nSU = 2449, nBSU = 631
Total nSU = 2449
*****
optimization finished, #iter = 5060
nu = 0.626126
obj = -882.180355, rho = -0.484663
nSU = 2448, nBSU = 622
Total nSU = 2448
*****
optimization finished, #iter = 5057
nu = 0.630493
obj = -891.529237, rho = -0.480294
nSU = 2450, nBSU = 641
Total nSU = 2450
Cross Validation Accuracy = 77.4037%
```

Figura 31. Treinamento dos dados através de validação cruzada (5-fold) pelo LIBSVM

3.3 Implementações KNN e NNSRM

Para a simulação da técnica KNN [1, 39] não se usou nenhum simulador já existente. O método NNSRM [23, 22] foi escolhido também para usar na simulação por ser um algoritmo bastante recente e que, segundo os autores, pode rivalizar com SVM [23]. As técnicas de KNN utilizadas neste trabalho (KNN tradicional e NNSRM) foram implementadas pelos autores deste projeto.

As duas técnicas foram implementadas na linguagem JAVA [2]. Ela foi escolhida por ser uma linguagem simples e portátil. O ambiente de desenvolvimento utilizado em todas as implementações foi o Eclipse [14], uma ferramenta gratuita cuja funcionalidade não é somente para o desenvolvimento de códigos java, e sim para visualização de base de dados, edição de arquivos XML, JSP, HTML, JavaScripts entre outras funcionalidades.

Os códigos fonte encontram-se no apêndice desta Monografia.

Neste trabalho, foram feitas implementações da técnica KNN tradicional, a qual foi implementada com os métodos *holdout* e validação cruzada (10-fold), assim como foi implementada com os valores de K iguais a 1, 3 e 5. Também foram feitas implementações da técnica NNSRM. Com essa técnica também foram feitas implementações com o método *holdout* e validação cruzada (10-fold).

Para as implementações usando o método *holdout*, o programa recebe como entrada dois arquivos, um contendo os dados de treinamento e o outro contendo os padrões que irão testar o classificador. A divisão dos dados obteve 78 padrões para treinamento e 79 padrões para teste.

Com o método validação cruzada (*10-fold*) os dados não foram divididos em dois conjuntos, o programa recebe apenas um arquivo contendo todos os padrões. Para a escolha dos dados de treinamento o arquivo foi “dividido” (pelo programa Java) em 10 partes, pois na validação cruzada *10-fold* ocorrem dez treinamentos, e a cada treinamento os dados variam. Como no total são 157 padrões, os dados foram divididos em 7 conjuntos com 16 padrões, e 3 conjuntos com 15 padrões. A cada treinamento são armazenados 9 conjuntos e o outro é utilizado para teste.

Para o cálculo do erro de classificação final, foi utilizada a equação:

$$E = 100 - \frac{100 * \textit{acertos}}{\textit{total}} \quad (6)$$

onde *acertos* é a quantidade de padrões de teste classificados corretamente e *total* é a quantidade total de padrões de teste, que no caso do método *holdout* é 78, e no caso de validação cruzada essa quantidade varia.

Para as implementações deste trabalho foram utilizadas as seguintes classes de java:

- `java.io.BufferedReader;`
- `java.io.File;`
- `java.io.FileInputStream;`
- `java.io.IOException;`
- `java.io.InputStreamReader;`
- `java.util.ArrayList;`
- `java.util.Collections;`
- `java.util.StringTokenizer;`

Capítulo 4

Experimentos

4.1 Base de dados

A base de dados possui características de pacientes que fizeram implantes. São sete características analisadas, escolhidas por terem influência no resultado da pesquisa. As características analisadas e seus possíveis valores estão apresentadas na Tabela 1.

Tabela 1. Características analisadas e possíveis valores

Nome	Possíveis valores
Idade (anos)	17 a 74
Sexo	{masculino, feminino}
Localização do implante	{maxilar superior posterior, maxilar superior anterior, mandíbula posterior, mandíbula anterior}
Tipo de implante	{convencional, tratamento de superfície}
Técnica cirúrgica	{convencional, complexa (enxerto, membrana)}
Fumante	{sim, não}
Possui alguma doença anterior ao implante?	{não, sim (diabete), sim (osteoporose), sim (radioterapia)}

Nossa base de dados possui sete tipos de classes, e sua distribuição está apresentada na Tabela 2. Foram coletados um total de 157 padrões.

Tabela 2. Distribuição das classes na base de dados

Classe	Freqüência	Porcentagem
1 (Sucesso - até 1 ano)	2	1,27%
2 (Sucesso - 1 a 2 anos)	24	15,29%
3 (Sucesso - 2 a 3 anos)	25	15,92%
4 (Sucesso - 3 a 4 anos)	21	13,38%
5 (Sucesso - 4 a 5 anos)	16	10,19%
6 (Sucesso - 5 anos ou mais)	62	39,49%
7 (Insucesso)	7	4,46%
Total	157	100%

Por causa da pequena base de dados inicial, foram gerados dez conjuntos, a partir do original, para a aplicação da validação cruzada (*10-fold*). Cada conjunto se diferencia do outro pela ordem dos padrões no mesmo. Essa ordem foi alternada para fornecer um resultado mais significativo, pois não se tem um padrão para a escolha de quais dados devem ser usados para treinamento e quais devem ser usados para teste. A ordem dos padrões nos conjuntos foi dada através de uma função pseudo-aleatória, que vem implementada no Microsoft ExcelTM.

Os valores das idades foram normalizados entre 0 e 1, usando a expressão

$$x_{norm} = \frac{x - x_{mín}}{x_{máx} - x_{mín}} \quad (7)$$

onde x_{norm} é o valor normalizado correspondente à idade original, $x_{mín}$ e $x_{máx}$ são, respectivamente, o menor e o maior valor entre os 283 possíveis valores de idade. As outras características foram representadas com valores binários, não necessitando, assim, de normalização. A Fig. 32 mostra a distribuição das idades normalizadas entre 0 e 1.

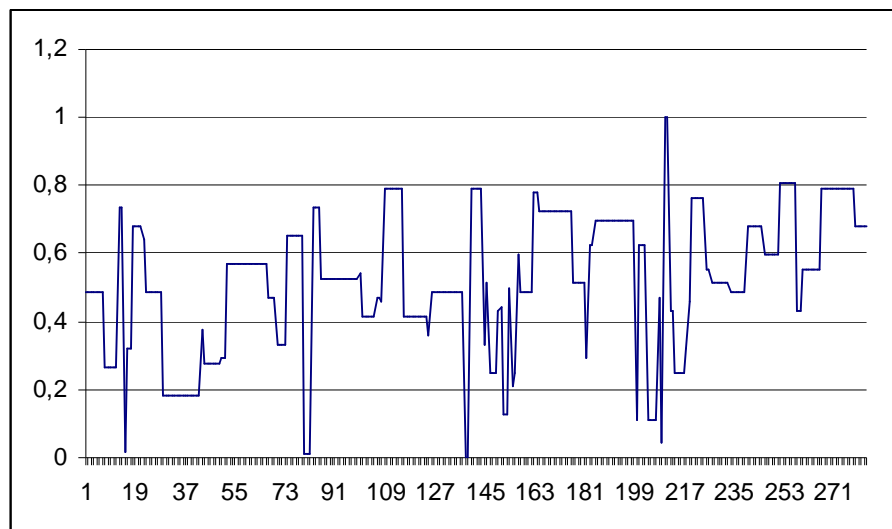


Figura 32: Idades normalizadas entre 0 e 1

Após a normalização, foi aplicado o método da validação cruzada (*10-fold*) em cada um dos conjuntos. Portanto a base de dados original gerou dez conjuntos de dados, e cada conjunto foi treinado dez vezes (devido à validação cruzada *10-fold*), realizando assim um total de cem treinamentos com cada técnica. Também foram utilizadas as técnicas de *holdout* e *leave-one-out* na base de dados, para efeito de comparação. Foram utilizados os mesmos conjuntos de dados tanto para experimentos com redes neurais quanto para SVM, KNN e NNSRM.

4.2 Experimentos com redes neurais MLP

Para usar redes neurais artificiais do tipo MLP foram definidas como entradas as características dos pacientes e como saída o resultado da previsão do sucesso do implante dentário. A base de dados forneceu, então, dez entradas e sete saídas para a rede neural.

4.2.1 Topologia das redes neurais

Foram utilizadas redes neurais com apenas uma camada escondida, e a função de ativação desta camada foi a sigmóide logística. A camada de saída usa a classificação *winner-takes-all* [17], a qual diz que o padrão de entrada será da mesma classe da saída que possuir um maior valor no final. As redes são totalmente conectadas, isto é, possuem todas as conexões possíveis. Todas as redes possuem 10 entradas e 7 saídas, porém o número de neurônios da camada escondida varia entre 2, 5 e 10.

4.2.2 Divisão dos dados

Para utilizar redes MLP os dados foram divididos em três partes. Uma para treinamento, outra para validação, e outra para teste. Os dados de validação servem para testar a rede no meio do treinamento, ou seja, para verificar a generalização da mesma. Os três conjuntos tiveram, aproximadamente, a mesma quantidade de dados, assim como a escolha dos dados em cada conjunto prezou, aproximadamente, a mesma quantidade de cada classe no conjunto de treinamento.

4.2.3 Metodologia de treinamento

O algoritmo de treinamento utilizado nas redes MLP foi o Rprop. Para cada topologia o treinamento foi realizado 10 vezes com diferentes inicializações de pesos, em 10 conjuntos de dados diferentes. A cada vez, o treinamento termina pelo critério de parada GL5 do *Proben 1* [31] ou se a quantidade de ciclos ultrapassar 10000. O critério de parada GL5 corresponde ao método de parada antecipada baseada no conjunto de validação [25], ou seja, o treinamento acaba quando o MSE (*mean square error* – erro médio quadrático) começa a aumentar, sendo verificado pelo conjunto de validação. Para uso das redes MLP foi utilizado o simulador SNNS de redes neurais.

4.2.4 Resultados obtidos

Para as redes MLP, foi analisado o erro de classificação no conjunto de teste. A Tabela 3 fornece os resultados obtidos com as três topologias.

Pode-se observar que os resultados foram bem próximos para as três topologias de rede. O erro de classificação ficou perto de 59% em todos os resultados. Apesar da pequena diferença, a topologia que obteve o melhor resultado foi a rede neural com apenas 2 neurônios na camada escondida, com erro de classificação igual a 59,423%, e a que obteve um erro de classificação maior foi a rede com 9 neurônios na camada escondida, com valor igual a 59,999%.

Tabela 3: Erro de classificação de cada topologia de rede MLP

	Neurônios da camada escondida (N)		
	N=2	N=5	N=10
Erro de classificação	51,92%	51,92%	51,92%
	63,46%	65,38%	59,62%
	61,54%	59,62%	61,54%
	61,54%	61,54%	61,54%
	61,54%	63,46%	65,38%
	61,54%	61,54%	57,69%
	61,54%	63,46%	63,46%
	57,69%	57,69%	65,38%
	55,77%	55,77%	55,77%
	57,69%	57,69%	57,69%
Média:	59,423%	59,807%	59,999%
Desvio padrão:	3,564514%	4,099515%	4,328196%

4.3 Experimentos com redes neurais RBF

As redes neurais RBF, assim como as redes MLP, possuem 10 entradas e 7 saídas, porém o número de neurônios da camada escondida é definido dinamicamente durante a fase de treinamento. A camada de saída das redes RBFs também usa a classificação *winner-takes-all* [44].

4.3.1 Divisão dos dados

Para os experimentos com as redes RBFs, foram gerados 10 conjuntos a partir da base de dados inicial, conforme explicado na seção 4.1. Primeiro, cada conjunto de dados foi dividido em duas partes, uma para treinamento e outra para teste. Foram então realizados os treinamentos com estes conjuntos gerados. Depois, na base de dados, foi aplicada a técnica de validação cruzada, para comparação com as experiências anteriores. Foi aplicada a validação cruzada (*10-fold*) em cada um dos 10 conjuntos gerados.

4.3.2 Metodologia de treinamento

O treinamento realizado nas redes RBF foi através do algoritmo DDA e foi feito nos 10 conjuntos gerados a partir da base de dados original. O treinamento das RBFs foi realizado inicialmente com valores default da ferramenta de simulação, depois foi feita a seleção do melhor parâmetro θ^- e realizado testes com ele. Para uso desta técnica foi utilizado o simulador SNNS, e como ele não possui a opção de validação cruzada, uma implementação adicional teve de ser feita.

4.3.3 Resultados obtidos

Os resultados analisados para as redes RBF são quanto ao erro de classificação e a quantidade de nodos na camada escondida. A Tabela 4 apresenta o resultado de classificação obtido para cada conjunto de parâmetros θ^+ e θ^- testados.

Tabela 4: Resultado da classificação da redes RBF variando o parâmetro θ^-

	$\theta^+ = 0,4$ $\theta^- = 0,1$	$\theta^+ = 0,4$ $\theta^- = 0,01$	$\theta^+ = 0,4$ $\theta^- = 0,001$
	Erro de Classificação	Erro de Classificação	Erro de Classificação
	75,95%	75,95%	75,95%
	34,62%	32,05%	32,05%
	41,03%	37,18%	23,08%
	43,59%	43,59%	38,46%
	51,28%	51,28%	51,28%
	37,18%	35,90%	34,62%
	44,87%	43,59%	41,03%
	51,28%	52,56%	44,87%
	42,31%	42,31%	39,74%
	38,46%	37,18%	37,18%
Média:	46,05%	45,15%	41,82%
Desvio Padrão:	11,84285%	12,64152%	14,14175%

O resultado dos experimentos mostram que o valor padrão de θ^- (0,1) obteve o pior resultado, com erro de classificação igual a 46,05%. O resultados mostram também que o parâmetro θ^- influencia no resultado [29]. Ao diminuir o valor do θ^- de 0,4 para 0,04, o erro de classificação cai de 46,05% para 45,15%, e ao diminuir o valor do θ^- ainda mais, o erro de classificação cai para 41,82%, sendo este o melhor resultado obtido.

Outro experimento foi feito utilizando o método da validação cruzada (10-fold). RBF-DDA com valores padrão gerou os resultados mostrados na tabela 5. O erro de classificação variou nos 10 conjuntos de dados entre 22,091% e 29,636%, e foram geradas redes com aproximadamente 73 neurônios escondidos.

Tabela 5: RBF com parâmetro default e validação cruzada (10-fold)

RBF-DDA com parâmetros default		
θ^-	Erro de classificação	Neurônios escondidos
0,1	27,152%	73,90
0,1	22,091%	73,90
0,1	23,607%	73,20
0,1	24,759%	73,70
0,1	24,727%	73,20
0,1	25,181%	73,70
0,1	25,607%	73,90
0,1	29,636%	72,90
0,1	27,393%	73,10
0,1	25,393%	73,90
Média	25,554%	73,54
Desvio padrão	2,105%	0,395

Ao selecionar o melhor valor de θ^- obteve-se uma melhoria no erro de classificação, e o número de unidades escondidas continuou praticamente o mesmo. Esses dados estão mostrados na tabela 6. Pode-se observar que o θ^- ótimo gerado foi quase sempre 0,01, porém, no segundo e no

terceiro conjunto de dados o θ^- ótimo foi 0,1. No segundo conjunto o erro de classificação e o número de neurônios escondidos foi o mesmo para $\theta^- = 0,1$ e $\theta^- = 0,01$.

Tabela 6: RBF-DDA com parâmetros selecionados

RBF-DDA com parâmetros selecionados		
Theta ótimo	Erro de classificação	Neurônios escondidos
0,01	26,03%	73,90
0,1/0,01	22,09%	73,90
0,1	23,61%	73,20
0,01	24,09%	73,70
0,01	22,73%	73,70
0,01	24,52%	73,70
0,01	24,94%	73,90
0,01	26,97%	73,70
0,01	26,06%	73,20
0,01	24,06%	73,90
Média	24,51%	73,68
Desvio padrão	1,53%	0,27

4.4 Experimentos com SVM

4.4.1 Metodologia de treinamento

Os treinamentos das SVMs foram realizados através do software LIBSVM [19]. Primeiro foram realizados treinamentos com os parâmetros padrão do simulador. Foi feito um treinamento com os conjuntos divididos em dados de treinamento e teste (*holdout*), e foi feito um treinamento utilizando a validação cruzada. Depois foi feita uma seleção dos melhores parâmetros C e γ , e foi feito o treinamento utilizando esses valores e validação cruzada (*10-fold*). Como o software LIBSVM já fornece a opção de validação cruzada, nenhuma implementação adicional foi preciso.

4.4.2 Resultados obtidos

Os resultados dos treinamentos das SVMs com parâmetros padrão estão mostrados na Tabela 7. Os resultados mostram que o erro de classificação para o treinamento usando validação cruzada foi menor do que o erro de classificação usando *holdout*, e com uma variação menor também. A Tabela 8 mostra os resultados dos treinamentos com os valores dos parâmetros selecionados e com validação cruzada (*10-fold*). O erro médio de classificação do treinamento com seleção de parâmetros foi bem menor do que o erro médio de classificação do treinamento com parâmetros padrão. Isso mostra a importância de se escolher bem os parâmetros antes de qualquer treinamento.

Tabela 7: SVM com parâmetros default

SVM com parâmetros default		
	Erro de classificação	Erro de classificação 10-fold CV
	70,51%	59,61
	62,82%	58,97
	55,13%	60,25
	61,54%	60,89
	64,10%	58,33
	61,53%	58,33
	60,25%	58,33
	64,10%	60,25
	60,25%	59,61
	64,10%	59,61
Média	62,43%	59,42
Desvio padrão	3,92%	0,91

Tabela 8: SVM com seleção de parâmetros

SVM com seleção de parâmetros via 10-fold CV		
	Erro de classificação	Neurônios escondidos
	25,64%	111,60
	24,36%	101,00
	23,08%	108,70
	23,08%	102,50
	24,36%	106,50
	24,36%	101,60
	23,72%	101,60
	24,36%	97,50
	24,36%	107,20
	23,08%	102,30
Média	24,04%	104,05
Desvio padrão	0,81%	4,27

4.4.3 SVMs X RNAs

A Tabela 9 mostra a comparação entre os erros de classificação para cada um dos 10 conjuntos, nas duas técnicas, assim como o número de unidades escondidas geradas. Para essa comparação foram pegos os melhores resultados obtidos de cada técnica. Observe que as duas técnicas obtiveram as médias do erro de classificação bem próximas, porém a RBF-DDA com a seleção do melhor θ^- gerou redes com menor número de unidades escondidas.

Tabela 9: comparação entre as técnicas RBF-DDA e SVM

	RBF-DDA com seleção de θ		SVM com seleção de parâmetros	
	Erro de classificação	Unidades escondidas	Erro de classificação	Unidades escondidas
Conj.de dados 1	26,03%	73,90	25,64%	111,60
Conj.de dados 2	22,09%	73,90	24,36%	101,00
Conj.de dados 3	23,61%	73,20	23,08%	108,70
Conj.de dados 4	24,09%	73,70	23,08%	102,50
Conj.de dados 5	22,73%	73,70	24,36%	106,50
Conj.de dados 6	24,52%	73,70	24,36%	101,60
Conj.de dados 7	24,94%	73,90	23,72%	101,60
Conj.de dados 8	26,97%	73,70	24,36%	97,50
Conj.de dados 9	26,06%	73,20	24,36%	107,20
Conj.de dados 10	24,06%	73,90	23,08%	102,30
Média	24,51%	73,68	24,04%	104,05
Desvio Padrão	1,53%	0,27	0,81%	4,27

4.5 Experimentos com KNN e NNSRM

4.5.1 Metodologia de treinamento

Para experimentos com KNN, foi feita a implementação da técnica, assim como algumas variações propostas [23, 22], substituindo o uso de simuladores pré-existentes, como foi feito nas técnicas anteriores. Com essa técnica também foram feitas experiências com os dez conjuntos gerados a partir da base de dados original.

Primeiro, a técnica foi testada com os dados divididos de acordo com o método holdout, uma parte para treinamento e outra para teste. Depois, foi utilizado o método de validação cruzada nos dados. Também foram realizados experimentos com a técnica NNSRM, como explicada na seção 3.3, assim como foram feitos testes com esta técnica utilizando validação cruzada (*10-fold*) nos dados.

4.5.2 Resultados obtidos

Os resultados da técnica KNN com o método *holdout* estão apresentados na Tabela 10. Foram feitos testes com $K=1$, $K=3$ e $K=5$. Os resultados dos experimentos mostraram que o KNN com $K=1$ obteve o melhor resultado, com erro de classificação igual a 37,81%, e o $K=5$ foi o que obteve o pior resultado, com erro de classificação igual a 62,43%.

A Tabela 11 mostra o resultado dos experimentos com validação cruzada (*10-fold*). Também foram testados valores de K iguais a 1, 3 e 5. O uso da validação cruzada resultou mais uma vez em erros de classificação menores. Enquanto que o melhor resultado com o método *holdout* gerou erro de classificação 37,81%, dois dos resultados com validação cruzada forneceram erros mais baixos, sendo o menor igual a 24,07%. Em ambos os casos o melhor resultado foi KNN com $K=1$.

Tabela 10: KNN com holdout

	K=1	K=3	K=5
	Erro de classificação	Erro de classificação	Erro de classificação
	76,92%	83,33%	91,02%
	34,61%	38,46%	57,69%
	35,89%	46,15%	55,12%
	26,92%	37,17%	44,87%
	38,46%	58,97%	71,79%
	33,33%	46,15%	53,84%
	33,33%	46,15%	66,66%
	29,48%	42,30%	64,10%
	34,61%	51,28%	62,82%
	34,61%	46,15%	56,41%
Média	37,81%	49,61%	62,43%
Desvio Padrão	14,11%	13,36%	12,56%

Tabela 11: KNN com validação cruzada (10-fold)

	K=1	K=3	K=5
	Erro de classificação	Erro de classificação	Erro de classificação
	24,20%	34,39%	48,40%
	25,47%	33,12%	49,68%
	22,92%	30,57%	45,85%
	24,84%	33,75%	49,04%
	22,29%	36,30%	47,77%
	22,92%	33,12%	45,85%
	22,92%	29,29%	47,13%
	26,75%	37,54%	48,40%
	23,56%	30,57%	46,49%
	24,84%	31,84%	45,85%
Média	24,07%	33,04%	47,44%
Desvio Padrão	1,41%	2,60%	1,42%

A seguir, estão os resultados do algoritmo NNSRM, com K=1. A Tabela 12 mostra o erro de classificação de cada um dos 10 conjuntos e a quantidade de padrões de teste selecionada pelo algoritmo. O erro de classificação obtido foi 39,09% e a quantidade de padrões selecionada ficou em torno de 45. Essa técnica teve o erro de classificação próximo ao KNN tradicional, porém tem a vantagem de armazenar menos padrões de treinamento na memória (45 ao invés de 78, que é a quantidade total de padrões de treinamento).

Tabela 12: NNSRM com holdout

	Erro de classificação	Padrões selecionados
	74,35%	31
	34,61%	52
	39,74%	43
	32,05%	51
	39,74%	46
	34,61%	45
	29,48%	47
	29,48%	47
	37,17%	42
	39,74%	47
Média	39,09%	45,10
Desvio padrão	13,01%	5,84

Também foram feitos experimentos com o método NNSRM via validação cruzada (10-*fold*). Os resultados estão na Tabela 13.

Tabela 13: NNSRM com validação cruzada (10-*fold*)

	Erro de classificação	Padrões selecionados
	26,75%	54
	27,38%	53
	27,38%	54
	28,02%	55
	26,11%	57
	26,11%	57
	29,93%	52
	27,38%	51
	27,38%	54
	31,84%	52
Média	27,82%	53,90
Desvio padrão	1,78%	2,02

O erro de classificação diminuiu em relação ao erro de classificação sem validação cruzada. Antes o erro era de 39,09% e, com a validação cruzada, o erro foi para 27,82%. Porém a quantidade de padrões selecionados aumentou de 45,10 para 53,90.

A Tabela 14 mostra um comparativo entre os melhores resultados da técnica KNN.

Tabela 14: Comparativo entre os melhores resultados da técnica KNN

	KNN tradicional				NNSRM			
	Holdout		Validação cruzada		Holdout		Validação cruzada	
	Erro	#padrões	Erro	#padrões	Erro	#padrões	Erro	#padrões
	76,92%	78	24,20%	141	74,35%	31	26,75%	54
	34,61%	78	25,47%	141	34,61%	52	27,38%	53
	35,89%	78	22,92%	141	39,74%	43	27,38%	54
	26,92%	78	24,84%	141	32,05%	51	28,02%	55
	38,46%	78	22,29%	141	39,74%	46	26,11%	57
	33,33%	78	22,92%	141	34,61%	45	26,11%	57
	33,33%	78	22,92%	141	29,48%	47	29,93%	52
	29,48%	78	26,75%	142	29,48%	47	27,38%	51
	34,61%	78	23,56%	142	37,17%	42	27,38%	54
	34,61%	78	24,84%	142	39,74%	47	31,84%	52
Média	37,81%	78	24,07%	141,3	39,09%	45,1	27,82%	53,9
Desvio Padrão	14,11%	0	1,40%	0,483046	13,01%	5,8395	1,77%	2,0248

A tabela 14 mostra que o menor erro de classificação foi obtido com a técnica KNN tradicional e validação cruzada, tendo o valor 24,07%. A técnica NNSRM com validação cruzada obteve um erro de classificação um pouco maior, porém com a vantagem da quantidade de padrões de teste armazenados ser bem menor. Essa diferença da quantidade de padrões está ilustrada na Fig. 33. Isso implica em um espaço de memória menor para armazenar os padrões. Quando se usa uma base de dados pequena, como a deste trabalho por exemplo, isso pode não ser muito importante, porém para base de dados maiores esse fator pode ser muito relevante.

Um outro fator que se destaca nesses resultados é o custo computacional. Com um número menor de dados de treinamento armazenados, menor é o tempo para classificar um novo padrão. Na base de dados usada, o número de padrões armazenados diminuiu de 141, em média, (KNN tradicional) para 53, em média, (NNSRM), portanto é de se esperar que o tempo de espera necessário para a classificação de um novo padrão diminua pelo menos a metade. E, para bases de dados maiores, é muito importante a economia deste tempo gasto.

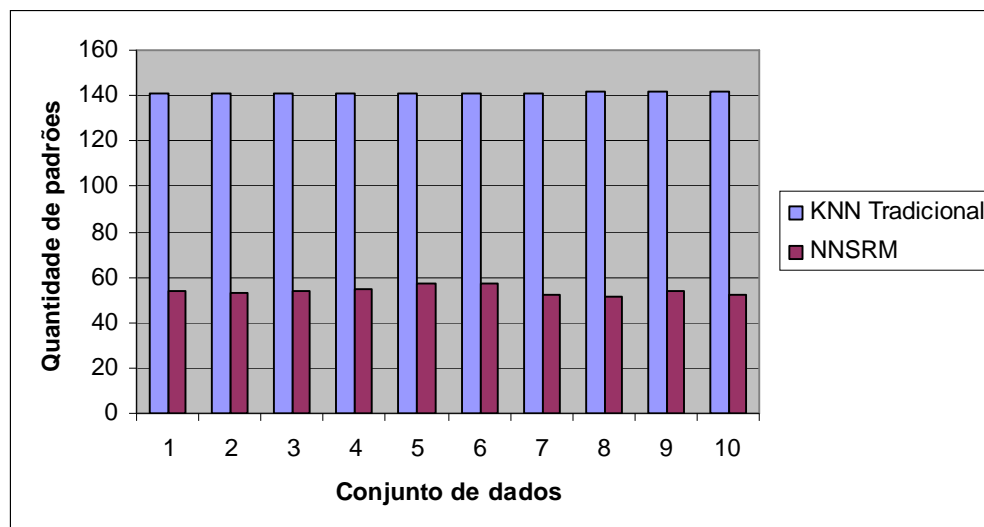


Figura 33: Quantidade de padrões armazenados nas técnicas KNN e NNSRM para cada conjunto de dados

4.5.3 RBF-DDA x SVM x KNN x NNSRM

A tabela 15 mostra um resumo dos erros de classificação obtidos pelas técnicas usando validação cruzada (10-*fold*). Percebe-se que os resultados obtidos foram bem próximos, apenas a técnica NNSRM obteve um resultado um pouco maior. Esse comparativo está mostrado graficamente na Fig. 34

Tabela 15: Comparativo entre os erros de classificação obtidos pelas técnicas via validação cruzada

	RBF-DDA com seleção de θ^-	SVM com seleção de parâmetros	KNN tradicional	NNSRM
	Erro de classificação	Erro de classificação	Erro de classificação	Erro de classificação
	26,03%	25,64%	24,2%	26,75%
	22,09%	24,36%	25,47%	27,38%
	23,61%	23,08%	22,92%	27,38%
	24,09%	23,08%	24,84%	28,02%
	22,73%	24,36%	22,29%	26,11%
	24,52%	24,36%	22,92%	26,11%
	24,94%	23,72%	22,92%	29,93%
	26,97%	24,36%	26,75%	27,38%
	26,06%	24,36%	23,56%	27,38%
	24,06%	23,08%	24,84%	31,84%
Média	24,51%	24,04%	24,07%	27,82%
Desvio Padrão	1,53%	0,81%	1,41%	1,77%

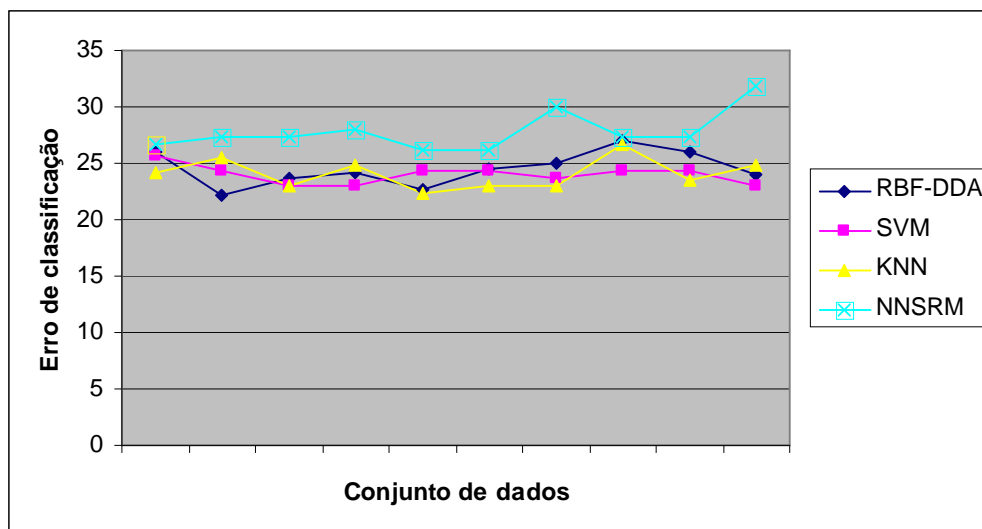


Figura 34: Comparativo entre os erros de classificação obtidos por cada técnica via validação cruzada

A tabela 16 mostra um comparativo entre a complexidade das técnicas utilizadas neste trabalho, ou seja, a quantidade de padrões armazenados para a devida classificação. A tabela

mostra que a técnica NNSRM obteve uma menor complexidade, seguida pela técnica RBF-DDA. Isto indica que a técnica NNSRM possui um treinamento mais rápido e precisa de menos memória do que as outras técnicas aqui estudadas. O resultado está mostrado graficamente na Fig. 35. Podemos observar também que a técnica SVM obteve um desvio padrão maior que as outras, apresentando uma grande variação na quantidade de padrões armazenados.

Tabela 16: Comparativo entre a quantidade de padrões armazenados em cada técnica

	RBF-DDA com seleção de θ^-	SVM com seleção de parâmetros	KNN via validação cruzada (10-fold)	NNSRM via validação cruzada (10-fold)
	#padrões	#padrões	#padrões	#padrões
	73,90	111,60	141	54
	73,90	101,00	141	53
	73,20	108,70	141	54
	73,70	102,50	141	55
	73,70	106,50	141	57
	73,70	101,60	141	57
	73,90	101,60	141	52
	73,70	97,50	142	51
	73,20	107,20	142	54
	73,90	102,30	142	52
Média	73,68	104,05	141,3	53,9
Desvio Padrão	0,27	4,27	0,483046	2,0248

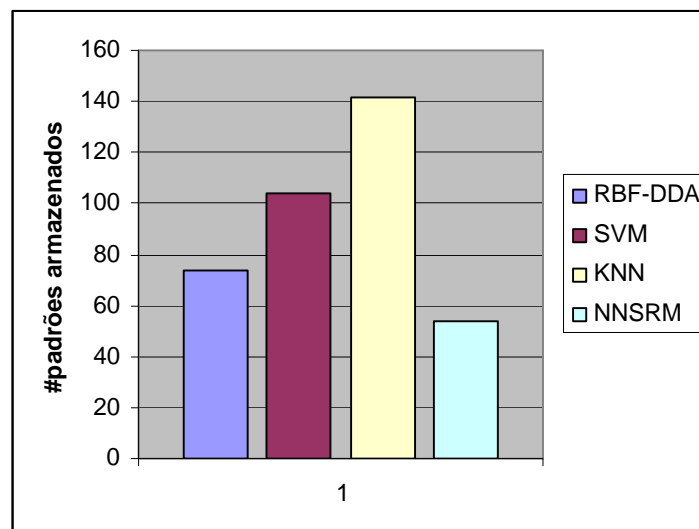


Figura 35: Comparativo entre a complexidade das técnicas RBF-DDA, SVM, KNN e NNSRM

Conclusões e Trabalhos Futuros

Este trabalho teve como objetivo fazer uma comparação entre técnicas de aprendizagem de máquina para o problema de previsão de sucesso em implantes dentários. Foram testadas técnicas já bastante utilizadas atualmente, como RNAs do tipo MLP e RBF-DDA, SVM, KNN, e foi implementada a técnica NNSRM, uma técnica recente que tende a rivalizar com o SVM.

Foram apresentados, também, algumas técnicas de comparação de classificadores, pois como a quantidade de dados utilizada neste trabalho é pequena, houve a necessidade de se utilizar alguns métodos para simular uma base de dados maior.

Neste trabalho, foram utilizados alguns simuladores para o treinamento das técnicas, porém, como o método NNSRM é muito recente, não foi utilizado simulador para o seu treinamento, e sim, o método foi implementado para treinar e classificar novos padrões (em Java). Para a técnica KNN também não foi utilizado simulador, a técnica também foi implementada. O simulador LIBSVM já oferece a opção do treinamento com validação cruzada, porém, para as outras técnicas, este método teve que ser implementado.

Foram realizados treinamentos com RNAs do tipo MLP com 2, 5 e 10 neurônios na camada escondida, obtendo resultados bem próximos. Para as técnicas RBF-DDA e SVM foi feita uma seleção de parâmetros, antes do treinamento, obtendo resultados melhores do que o treinamento com parâmetros *default*. Com a técnica KNN foram feitos treinamentos com o valor de K igual a 1, 3 e 5, com o K=1 obtendo melhor desempenho.

Mostramos que, neste problema, RBF-DDA com seleção de θ^- consegue obter desempenho similar a SVM com seleção de C e γ , com a vantagem de gerar redes com menor número de unidades escondidas, assim como a técnica KNN também obteve desempenho próximo. A técnica NNSRM teve um desempenho um pouco mais baixo, porém com a vantagem de menor necessidade de memória e maior velocidade de classificação de um novo padrão. Essas características não influenciaram muito este problema, pois a base de dados utilizada é pequena, porém, para base de dados maiores, estas características são fundamentais, levando à decisão do uso ou não da técnica. Portanto, neste trabalho, a técnica que obteve um melhor resultado, em relação ao desempenho, complexidade do classificador e tempo pra classificar novos padrões, foi a técnica NNSRM, pois as vantagens obtidas em relação à complexidade do classificador e tempo pra classificar novos padrões supera a pequena diferença de desempenho obtida.

Este trabalho poderá ser usado para auxílio na tomada de decisão dos profissionais de odontologia, visando a uma maior segurança na previsão do resultado final do implante. O paciente também será beneficiado porque terá maiores informações sobre a probabilidade de sucesso do seu tratamento auxiliando-o na sua decisão de fazer ou não o planejamento proposto.

Para trabalhos futuros, devem-se coletar mais dados de pacientes, para realizar as experiências com uma base de dados maior, fornecendo mais segurança ao resultado. Uma outra opção é trabalhar com dados de pacientes com idades em um intervalo menor. Também sugerimos a utilização de novas técnicas para comparação com as estudadas neste trabalho, assim

como serem usados novos classificadores, para o treinamento das técnicas. Uma outra proposta é analisar a influência das entradas no desempenho de classificação (como feito em [11]).

Bibliografia

- [1] AMENDOLIA, S. R. et al. *A comparative study of K-Nearest Neighbour, Support Vector Machine and Multi-Layer Perceptron for Thalassemia screening*. Chemometrics and Intelligent Laboratory Systems, Volume 69, Issues 1-2, 28 November 2003, Pages 13-20.
- [2] A Sun Developer Network Site. Disponível em <java.sun.com>.
- [3] BALDISSEROTTO, J. *Efeitos de uma dieta salínica na qualidade óssea e no processo de osseointegração em ratos durante o envelhecimento*. Tese de doutorado. Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS. Outubro, 2003.
- [4] BERTHOLD, M. R. e DIAMOND, J. *Boosting the performance of RBF networks with dynamic decay adjustment*. In G. Tesauro et al, editor, *Advances in Neural Information Processing*, volume 7, pages 521–528. MIT Press, 1995.
- [5] BERTHOLD, M. R. e DIAMOND, J. *Constructive training of probabilistic neural networks*. *Neurocomputing*, 19:167–183, 1998.
- [6] BRAGA, A. P. et al. *Redes Neurais Artificiais: Teoria e aplicações*. Livros Técnicos e Científicos editora. Rio de Janeiro. 2000.
- [7] BROOMHEAD, D.S. e LOWE, D. *Multivariate functional interpolation and adaptive networks*. *Complex Systems*, 2:321-355, 1988.
- [8] BURGESS, C. J. C. *A tutorial on support vector machines for pattern recognition*. *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 1–47, 1998.
- [9] CORTES, C. e VAPNIK, V. *Support-vector network*. *Machine Learning*, pages 273–297. 1995
- [10] DAVID, V. e SANCHEZ, A. *Advanced support vector machines and kernel methods*. *Neurocomputing*, 55:5–20, 2003.
- [11] DELEN, D. et al. *Predicting breast cancer survivability: a comparison of three data mining methods*. *Artificial Intelligence in Medicine*, In Press, Corrected Proof, Available online 11 September 2004.
- [12] DIAS, D. N. e NETO, J. D. E. S. B. *Identificação dos Sintomas de Ferrugem em Áreas Cultivadas com Cana-de-Açúcar*. Abril 2004.
- [13] DUDA, R. O., HART, P. E. e STORK, D. G. *Pattern Classification*. Wiley-Interscience, second edition, 2000.
- [14] Eclipse.org. Disponível em <www.eclipse.org>.
- [15] FAUSETT, L. *Fundamentals of Neural Networks – Architecture, Algorithms, and Applications*. [S.l.]: Prentice Hall International, Inc., 1994.
- [16] GNECCO, B. B. et al. *Um Sistema de Visualização Imersivo e Interativo de Apoio ao Ensino de Classificação de Imagens*. Disponível em <www.lsi.usp.br/~brunobg/papers/wrv2001_cave.pdf>.
- [17] HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1998.

- [18] HAYKIN, S. *Redes Neurais, Princípios e Prática*. 2.edição. Porto Alegre: Bookman, 2001.
- [19] HSU, C.-W, CHANG, C.-C. e LIN, C.-J. *A Practical Guide to Support Vector Classification*. Available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>. 2004
- [20] HUDAK, M. J. *RCE classifiers: Theory and practice*. *Cybernetics and Systems*, 23:483–515, 1992.
- [21] ITO, S. A. e SERRA, T. G. *Uma aplicação de Redes Neurais Artificiais*. Universidade Federal do Rio Grande do Sul - Instituto de Informática. Arquiteturas Especiais de computadores - Trabalho II. Dezembro, 1998
- [22] KARAÇALI, B. et al. *A comparative analysis of structural risk minimization by support vector machines and nearest neighbor rule*. *Pattern Recognition Letters* 25 (2004) 63–71.
- [23] KARAÇALI, B. e KRIM, H. *Fast Minimization of Structural Risk by Nearest Neighbor Rule*. *IEEE transactions on neural networks*, Vol. 14, No. 1, Janeiro 2002.
- [24] MARK, J. L. *Introduction to Radial Basis Function Networks*. 1996.
- [25] OLIVEIRA, A. L. I.. *Neural Networks Forecasting and Classification-Based Techniques for Novelty Detection in Time Series*. Tese de doutorado. Dezembro, 2004.
- [26] OLIVEIRA, A.L.I., MELO, B.J.M., NETO, F.B.L., e MEIRA, S.R.L. (2004) *Combining data reduction and parameter selection for improving RBF-DDA performance*. *Lecture Notes in Computer Science*, Vol. 3315, pp. 778-787, Springer-Verlag. (Proc. of IX Ibero American Conference on Artificial Intelligence (IBERAMIA'2004)).
- [27] OLIVEIRA, A.L.I., NETO, F.B.L., e MEIRA, S.R.L. (2004). *Combining MLP and RBF neural networks for novelty detection in short time series*. *Lecture Notes in Computer Science*, Vol. 2972, pp. 844-853, Springer-Verlag. (Mexican International Conference on Artificial Intelligence (MICAI'2004)).
- [28] OLIVEIRA, A.L.I., NETO, F.B.L., and MEIRA, S.R.L. (2004). *Improving novelty detection in short time series through RBF-DDA parameter adjustment*. *International Joint Conference on Neural Networks (IJCNN'2004)*, Budapest, Hungary, IEEE Press.
- [29] OLIVEIRA, A. L. I., NETO, F. B. L., e MEIRA, S. R. L. *Improving RBF-DDA performance on optical character recognition through parameter selection*. In Proc. Of the 17th International Conference on Pattern Recognition (ICPR'2004), volume 4, pages 625–628, Cambridge,UK, 2004. IEEE Computer Society Press.
- [30] PAN, J., QIAO, Y. e SUN, S. *A Fast K Nearest Neighbors Classification Algorithm*. *IEICE TRANS. Fundamentals*, Vol. E87-A, No. 7 Abril 2004.
- [31] PRECHELT, L. *Proben1 – a set of neural networks benchmark problems and benchmarking rules*. Technical Report 21/94, Universität Karlsruhe, Germany. 1994.
- [32] RIBEIRO, L. N. *Rede Neural com Retropropagação: uma Aplicação na Classificação de Imagens de Satélite*. Dissertação de mestrado. Dezembro, 2003.
- [33] RIEDMILLER, M. e BRAUN, H. *A direct adaptive method for faster backpropagation learning: The RPROP algorithm*. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN 93)*, volume 1, pages 586–591, 1993.
- [34] SHAW-TAYLOR, J. e CRISTIANINI, N. *An Introduction to Support Vector Machines*. Cambridge University Press. 2000.
- [35] SHAW-TAYLOR, J. e CRISTIANINI, N. *Kernel Methods for Pattern Analysis*. Cambridge University Press. 2004
- [36] SINGH, S. e MARKOU, M. *An approach to novelty detection applied to the classification of image regions*. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):396–406, April 2004.

- [37] SNNS – *Stuttgart Neural Network Simulator. User Manual* – Version 4.2. University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1998.
- [38] TAFNER, M. A., XEREZ, M. e FILHO, I. R. *Redes Neurais Artificiais: Introdução e Princípios de Neurocomputação*. Blumenau: EKO, 1995.
- [39] TAN, S., *Neighbor-weighted K-nearest neighbor for unbalanced text corpus*. Expert Systems with Applications, Volume 28, Issue 4, May 2005, Pages 667-671.
- [40] THEODORIDIS, S. e KOUTROUMBAS, K. *Pattern Recognition*. Elsevier, second edition, 2003.
- [41] VAPNIK, V. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY. 1995.
- [42] VASCONCELOS, G. C. *An Investigation of Feedforward Neural Networks with Respect to the Detection of Spurious Patterns*. PhD thesis, University of Kent at Canterbury, 1995.
- [43] VASCONCELOS, G. C., FAIRHURST, M. C. e BISSET, D. L. *Investigating feedforward neural networks with respect to the rejection of spurious patterns*. Pattern Recognition Letters, 16(2):207–212, 1995.
- [44] WAN, V. e CAMPBELL, W. *Support vector machines for speaker verification and identification*, IEEE Proceeding, 2000.
- [45] 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.
- [46] WEBB, A. *Statistical Pattern Recognition*. Wiley, second edition. 2002.

Apêndice A

Implementações

Este apêndice mostrará os códigos fonte das implementações realizadas neste trabalho. Terá as implementações da técnica KNN e da técnica NNSRM.

KNN com K=1

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.StringTokenizer;

/**
 * @author Carolina Baldisserotto
 *
 * To change the template for this generated type comment go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
public class PrincipalkNN {

    public PrincipalkNN() {

    }

    public void algoritmo() {
        File file = new
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/Artigo/treinal.txt");
        FileInputStream fis = null;
        ArrayList arrayPadroesTreina = new ArrayList();

        File file2 = new
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/Artigo/testel.txt");
        FileInputStream fis2 = null;
        ArrayList arrayPadroesTeste = new ArrayList();
    }
}
```

```

try {
    fis = new FileInputStream(file);
    BufferedReader d = new BufferedReader(new
InputStreamReader(fis));
    String line = d.readLine();

    //Armazena os padrões de treinamento
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }

    fis2 = new FileInputStream(file2);
    BufferedReader d2 = new BufferedReader(new
InputStreamReader(fis2));
    String line2 = d2.readLine();

    //Armazena os padrões de teste
    while (line2 != null) {
        arrayPadroesTeste.add(line2);
        line2 = d2.readLine();
    }

    ArrayList dist = new ArrayList();
    int cont = 0;
    int contTotal = 0;
    for (int i = 0; i < arrayPadroesTeste.size(); i++) {
        String padrao1 = (String) arrayPadroesTeste.get(i);
        String tipo1 = padrao1.substring(24, 25);
        dist.clear();

        for (int j = 0; j < arrayPadroesTreina.size(); j++) {
            String padrao2 = (String)
arrayPadroesTreina.get(j);
            int tipo2 =
Integer.parseInt(padrao2.substring(24, 25));
            double de = distanciaEuclidiana(padrao1,
padrao2);
            dist.add(de + " ;" + tipo2);
        }

        Collections.sort(dist);
        String dstMenor = (String) dist.get(0);
        StringTokenizer str = new StringTokenizer(dstMenor,
";");

        String confereTipo = str.nextToken();
        confereTipo = str.nextToken();

        if (confereTipo.equals(tipo1)) {
            System.out.println("Acertou");
            cont++;
            contTotal++;
        } else {
            System.out.println("Errou");
            contTotal++;
        }
    }

    System.out.println("cont: " + cont);
    System.out.println("contTotal: " + contTotal);
    double f = 100-((100*cont)/contTotal);

```



```
System.out.println(f + "%");
} catch (IOException ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}
}

public static double distanciaEuclidiana(String a, String b) {
    double resultado = 0.0;
    double x1 = Double.parseDouble(a.substring(0, 5));
    double x2 = Double.parseDouble(a.substring(6, 7));
    double x3 = Double.parseDouble(a.substring(8, 9));
    double x4 = Double.parseDouble(a.substring(10, 11));
    double x5 = Double.parseDouble(a.substring(12, 13));
    double x6 = Double.parseDouble(a.substring(14, 15));
    double x7 = Double.parseDouble(a.substring(16, 17));
    double x8 = Double.parseDouble(a.substring(18, 19));
    double x9 = Double.parseDouble(a.substring(20, 21));
    double x10 = Double.parseDouble(a.substring(22, 23));
    int tipoA = Integer.parseInt(a.substring(24, 25));

    double y1 = Double.parseDouble(b.substring(0, 5));
    double y2 = Double.parseDouble(b.substring(6, 7));
    double y3 = Double.parseDouble(b.substring(8, 9));
    double y4 = Double.parseDouble(b.substring(10, 11));
    double y5 = Double.parseDouble(b.substring(12, 13));
    double y6 = Double.parseDouble(b.substring(14, 15));
    double y7 = Double.parseDouble(b.substring(16, 17));
    double y8 = Double.parseDouble(b.substring(18, 19));
    double y9 = Double.parseDouble(b.substring(20, 21));
    double y10 = Double.parseDouble(b.substring(22, 23));
    int tipoB = Integer.parseInt(b.substring(24, 25));

    resultado =Math.sqrt(
        Math.pow(x1 - y1, 2)
        + Math.pow(x2 - y2, 2)
        + Math.pow(x3 - y3, 2)
        + Math.pow(x4 - y4, 2)
        + Math.pow(x5 - y5, 2)
        + Math.pow(x6 - y6, 2)
        + Math.pow(x7 - y7, 2)
        + Math.pow(x8 - y8, 2)
        + Math.pow(x9 - y9, 2)
        + Math.pow(x10 - y10, 2));

    return resultado;
}

public static void main(String[] args) {
    PrincipalKNN p = new PrincipalKNN();
    p.algoritmo();
}
}
```

KNN com K=3

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.StringTokenizer;

/**
 * @author Carolina Baldisserotto
 *
 * To change the template for this generated type comment go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
public class Knn3 {

    public Knn3() {

    }

    public void algoritmo() {
        File file = new
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/Artigo/treinal0.txt");
        FileInputStream fis = null;
        ArrayList arrayPadroesTreina = new ArrayList();

        File file2 = new
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/Artigo/testel0.txt");
        FileInputStream fis2 = null;
        ArrayList arrayPadroesTeste = new ArrayList();
        try {
            fis = new FileInputStream(file);
            BufferedReader d = new BufferedReader(new
InputStreamReader(fis));
            String line = d.readLine();

            //Armazena os padrões de treinamento
            while (line != null) {
                arrayPadroesTreina.add(line);
                line = d.readLine();
            }

            fis2 = new FileInputStream(file2);
            BufferedReader d2 = new BufferedReader(new
InputStreamReader(fis2));
            String line2 = d2.readLine();

            //Armazena os padrões de teste
            while (line2 != null) {
                arrayPadroesTeste.add(line2);
                line2 = d2.readLine();
            }

            ArrayList dist = new ArrayList();
```

```

        int cont = 0;
        int contTotal = 0;
        for (int i = 0; i < arrayPadroesTeste.size(); i++) {
            String padrao1 = (String) arrayPadroesTeste.get(i);
            String tipo1 = padrao1.substring(24, 25);
            dist.clear();
            for (int j = 0; j < arrayPadroesTreina.size(); j++) {
                String padrao2 = (String)
arrayPadroesTreina.get(j);
                int tipo2 =
Integer.parseInt(padrao2.substring(24, 25));
                double de =
PrincipalKNN.distanciaEuclidiana(padrao1, padrao2);
                dist.add(de + " ;" + tipo2);
            }
            Collections.sort(dist);
            String dstMenor = (String) dist.get(0);
            String dstMenor2 = (String) dist.get(1);
            String dstMenor3 = (String) dist.get(2);

            StringTokenizer str1 = new StringTokenizer(dstMenor, ";");
            String confereTipo1 = str1.nextToken();
            confereTipo1 = str1.nextToken();

            StringTokenizer str2 = new StringTokenizer(dstMenor2, ";");
            String confereTipo2 = str2.nextToken();
            confereTipo2 = str2.nextToken();

            StringTokenizer str3 = new StringTokenizer(dstMenor3, ";");
            String confereTipo3 = str3.nextToken();
            confereTipo3 = str3.nextToken();

            String tipoVencedor = confereTipo1;

            if(confereTipo2.equals(confereTipo3)){
                tipoVencedor = confereTipo2;
            }

            if (tipoVencedor.equals(tipo1)) {
                System.out.println("Acertou");
                cont++;
                contTotal++;
            } else {
                System.out.println("Errou");
                contTotal++;
            }
        }
        System.out.println("cont: "+cont);
        System.out.println("contTotal: " +contTotal);
        double f = 100-((100 * cont)/78);
        System.out.println(f + "%");
    } catch (IOException ex) {
        System.out.println(ex.getMessage());
        ex.printStackTrace();
    }
}

public static void main(String[] args) {
    Knn3 p = new Knn3();
    p.algoritmo();
}

```

```
}  
}
```

KNN com K=5

```
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.StringTokenizer;  
  
/**  
 * @author Carolina Baldisserotto  
 *  
 * To change the template for this generated type comment go to  
 * Window>Preferences>Java>Code Generation>Code and Comments  
 */  
public class Knn3 {  
  
    public Knn3() {  
  
        public void algoritmo() {  
            File file = new  
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/Artigo/treinal0.txt");  
            FileInputStream fis = null;  
            ArrayList arrayPadroesTreina = new ArrayList();  
  
            File file2 = new  
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/Artigo/testel0.txt");  
            FileInputStream fis2 = null;  
            ArrayList arrayPadroesTeste = new ArrayList();  
            try {  
                fis = new FileInputStream(file);  
                BufferedReader d = new BufferedReader(new  
InputStreamReader(fis));  
                String line = d.readLine();  
  
                //Armazena os padrões de treinamento  
                while (line != null) {  
                    arrayPadroesTreina.add(line);  
                    line = d.readLine();  
                }  
  
                fis2 = new FileInputStream(file2);  
                BufferedReader d2 = new BufferedReader(new  
InputStreamReader(fis2));  
                String line2 = d2.readLine();  
  
                //Armazena os padrões de teste  
                while (line2 != null) {  
                    arrayPadroesTeste.add(line2);  
                    line2 = d2.readLine();  
                }  
            }  
        }  
    }  
}
```

```

ArrayList dist = new ArrayList();
int cont = 0;
int contTotal = 0;
for (int i = 0; i < arrayPadroesTeste.size(); i++) {
    String padrao1 = (String) arrayPadroesTeste.get(i);
    String tipo1 = padrao1.substring(24, 25);
    dist.clear();
    for (int j = 0; j < arrayPadroesTreina.size(); j++) {
        String padrao2 = (String)
arrayPadroesTreina.get(j);
        int tipo2 =
Integer.parseInt(padrao2.substring(24, 25));
        double de =
PrincipalKNN.distanciaEuclidiana(padrao1, padrao2);
        dist.add(de + " ; " + tipo2);
    }
    Collections.sort(dist);
    String dstMenor = (String) dist.get(0);
    String dstMenor2 = (String) dist.get(1);
    String dstMenor3 = (String) dist.get(2);
    String dstMenor4 = (String) dist.get(3);
    String dstMenor5 = (String) dist.get(4);

    StringTokenizer str1 = new StringTokenizer(dstMenor, ";");
    String confereTipo1 = str1.nextToken();
    confereTipo1 = str1.nextToken();

    StringTokenizer str2 = new StringTokenizer(dstMenor2, ";");
    String confereTipo2 = str2.nextToken();
    confereTipo2 = str2.nextToken();

    StringTokenizer str3 = new StringTokenizer(dstMenor3, ";");
    String confereTipo3 = str3.nextToken();
    confereTipo3 = str3.nextToken();

    StringTokenizer str4 = new StringTokenizer(dstMenor4, ";");
    String confereTipo4 = str4.nextToken();
    confereTipo4 = str4.nextToken();

    StringTokenizer str5 = new StringTokenizer(dstMenor5, ";");
    String confereTipo5 = str5.nextToken();
    confereTipo5 = str5.nextToken();

    String tipoVencedor = confereTipo1;

    if (confereTipo2.equals(confereTipo3) ||
confereTipo2.equals(confereTipo4) || confereTipo2.equals(confereTipo5)) {
        tipoVencedor = confereTipo2;
    } else if (confereTipo3.equals(confereTipo4) ||
confereTipo3.equals(confereTipo5)) {
        tipoVencedor = confereTipo3;
    } else if (confereTipo4.equals(confereTipo5)) {
        tipoVencedor = confereTipo4;
    }
    }

    if (tipoVencedor.equals(tipo1)) {
        System.out.println("Acertou");
        cont++;
    } else {

```

```
        System.out.println("Errou");
    }
}
System.out.println("cont: "+cont);
System.out.println("contTotal: " +contTotal);
double f = 100-((100 * cont)/78);
System.out.println(f + "%");
} catch (IOException ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}
}

public static void main(String[] args) {
    Knn3 p = new Knn3();
    p.algoritmo();
}
}
```

KNN com validação cruzada 10-fold

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.StringTokenizer;

/**
 * @author Carolina Baldisserotto
 *
 * To change the template for this generated type comment go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
public class KnnCross {

    public KnnCross() {
    }

    public void algoritmo() {
        File file =
            new
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/knn10.txt");
        FileInputStream fis = null;
        ArrayList arrayPadroesTreina = new ArrayList();
        ArrayList arrayPadroesTeste = new ArrayList();
        int cont = 0;
        int contTotal = 0;
        String line = "";
        try {
            for (int k = 1; k <= 10; k++) {
                arrayPadroesTreina.clear();
                arrayPadroesTeste.clear();
                fis = new FileInputStream(file);
                BufferedReader d =
                    new BufferedReader(new InputStreamReader(fis));
                if (k == 1) {
                    for (int c = 0; c < 16; c++) {
                        line = d.readLine();
                        arrayPadroesTeste.add(line);
                    }
                    line = d.readLine();
                    while (line != null) {
                        arrayPadroesTreina.add(line);
                        line = d.readLine();
                    }
                }

                if (k == 2) {
                    for (int c = 0; c < 16; c++) {
                        line = d.readLine();
```

```
        arrayPadroesTreina.add(line);
    }
    for (int c = 0; c < 16; c++) {
        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}

if (k == 3) {
    for (int c = 0; c < 32; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }
    for (int c = 0; c < 16; c++) {
        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}

if (k == 4) {
    for (int c = 0; c < 48; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }
    for (int c = 0; c < 16; c++) {
        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}

if (k == 5) {
    for (int c = 0; c < 64; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }
    for (int c = 0; c < 16; c++) {
        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}
}
```



```
if (k == 6) {
    for (int c = 0; c < 80; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }
    for (int c = 0; c < 16; c++) {
        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}

if (k == 7) {
    for (int c = 0; c < 96; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }
    for (int c = 0; c < 16; c++) {
        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}

if (k == 8) {
    for (int c = 0; c < 112; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }
    for (int c = 0; c < 15; c++) {
        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}

if (k == 9) {
    for (int c = 0; c < 127; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }
    for (int c = 0; c < 15; c++) {
        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
```

```

        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}
if (k == 10) {
    for (int c = 0; c < 142; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }

    line = d.readLine();
    while (line != null) {
        arrayPadroesTeste.add(line);
        line = d.readLine();
    }
}
ArrayList dist = new ArrayList();
for (int i = 0; i < arrayPadroesTeste.size(); i++) {
arrayPadroesTeste.get(i);

    String tipo1 = padrao1.substring(24, 25);
    dist.clear();

    for (int j = 0; j < arrayPadroesTreina.size();
j++) {
        String padrao2 = (String)
arrayPadroesTreina.get(j);
            int tipo2 =
Integer.parseInt(padrao2.substring(24, 25));
            double de =
PrincipalKNN.distanciaEuclidiana(padrao1, padrao2);
            dist.add(de + " ;" + tipo2);
        }
        Collections.sort(dist);
        String dstMenor = (String) dist.get(0);
        StringTokenizer str = new
StringTokenizer(dstMenor, ";");
        String confereTipo = str.nextToken();
        confereTipo = str.nextToken();

        if (confereTipo.equals(tipo1)) {
            System.out.println("Acertou");
            cont++;
            contTotal++;
        } else {
            System.out.println("Errou");
            contTotal++;
        }
    }
}
System.out.println("cont: " + cont);
System.out.println("contTotal: " + contTotal);
float f = 100-((100 * cont)/contTotal);
System.out.println("Erro: " + f + "%");
} catch (IOException ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}
}
}

```

```

public static void main(String[] args) {
    KnnCross p = new KnnCross();
    p.algoritmo();
}
}

```

NNSRM com K=1

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.StringTokenizer;

/**
 * @author Carolina Baldisserotto
 *
 * To change the template for this generated type comment go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
public class AlgoritmoNNSRM {

    public AlgoritmoNNSRM() {

    }

    public void algoritmo() {

        File file =
            new
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/Artigo/treinal0.txt");
        FileInputStream fis = null;
        ArrayList arrayPadroesTreina = new ArrayList();

        File file2 =
            new
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/Artigo/testel0.txt");
        FileInputStream fis2 = null;
        ArrayList arrayPadroesTeste = new ArrayList();

        ArrayList J = new ArrayList();
        ArrayList dist = new ArrayList();
        int cont = 0;
        double erro = 100.0;
        int k = 0;
        try {
            fis = new FileInputStream(file);
            BufferedReader d = new BufferedReader(new
InputStreamReader(fis));
            String line = d.readLine();

            //Armazena os padrões de treinamento
            while (line != null) {
                arrayPadroesTreina.add(line);
                line = d.readLine();
            }

```

```

//Armazena as distâncias de padrões diferentes
for (int c = 0; c < arrayPadroesTreina.size(); c++) {
    String padrao1 = (String) arrayPadroesTreina.get(c);
    String tipo1 = padrao1.substring(24, 25);

    for (int j = 0; j < arrayPadroesTreina.size(); j++) {
        String padrao2 = (String)
arrayPadroesTreina.get(j);
        String tipo2 = padrao2.substring(24, 25);
        double de =
KnnCross3.distanciaEuclidiana(padrao1, padrao2);
        if (!tipo1.equals(tipo2)) {
            dist.add(de + " ;" + padrao1 + ";" +
padrao2);
        }
    }
}
Collections.sort(dist);

while ((erro > 0)&&(k<500)) {
    String dstMenor = (String) dist.get(k);
    StringTokenizer str = new StringTokenizer(dstMenor,
";");

    String padrao = str.nextToken();
    String p1 = str.nextToken();
    String p2 = str.nextToken();

    if (!J.contains(p1)) {
        J.add(p1);
    }
    if (!J.contains(p2)) {
        J.add(p2);
    }

    erro = acharErro(J, arrayPadroesTreina);
    k++;
}
System.out.println("J: " + J.size());
fis2 = new FileInputStream(file2);
BufferedReader d2 = new BufferedReader(new
InputStreamReader(fis2));
String line2 = d2.readLine();

//Armazena os padrões de teste
while (line2 != null) {
    arrayPadroesTeste.add(line2);
    line2 = d2.readLine();
}
double erro2 = acharErro(J,arrayPadroesTeste);
System.out.println("erro final: " + erro2);

} catch (IOException ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}

}

public double acharErro(ArrayList arrayJ, ArrayList arrayTreina) {

```

```

ArrayList dist = new ArrayList();
int cont = 0;
int contTotal = 0;
for (int i = 0; i < arrayTreina.size(); i++) {
    String padrao1 = (String) arrayTreina.get(i);
    String tipo1 = padrao1.substring(24, 25);
    dist.clear();

    for (int j = 0; j < arrayJ.size(); j++) {
        String padrao2 = (String) arrayJ.get(j);
        int tipo2 = Integer.parseInt(padrao2.substring(24,
25));

        double de = PrincipalKNN.distanciaEuclidiana(padrao1,
padrao2);

        dist.add(de + " ;" + tipo2);
    }

    Collections.sort(dist);
    String dstMenor = (String) dist.get(0);
    StringTokenizer str = new StringTokenizer(dstMenor, ";");
    String confereTipo = str.nextToken();
    confereTipo = str.nextToken();

    if (confereTipo.equals(tipo1)) {
        cont++;
        contTotal++;
    } else {
        contTotal++;
    }
}
System.out.println("cont: " + cont);
System.out.println("contTotal: " + contTotal);
double f = 100 - ((100 * cont) / contTotal);
return (f);
}

public static void main(String[] args) {
    AlgoritmoNNSRM a = new AlgoritmoNNSRM();
    a.algoritmo();
}
}

```

NNSRM com validação cruzada 10-fold

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.StringTokenizer;

/**
 * @author Carolina Baldisserotto
 *
 * To change the template for this generated type comment go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
public class NNSRMCross {

    public NNSRMCross() {
    }
    int cont = 0;
    int contTotal = 0;
    public void algoritmo() {

        File file =
            new
File("C:/Carol/Monografia/dados/Arquivos/KNN/Cross/knn4.txt");
        FileInputStream fis = null;
        ArrayList arrayPadroesTreina = new ArrayList();
        ArrayList arrayPadroesTeste = new ArrayList();
        ArrayList J = new ArrayList();
        ArrayList dist = new ArrayList();
        int cont = 0;
        double erro = 100.0;
        int k = 0;
        String line = "";
        try {
            for (int n = 1; n <= 10; n++) {
                arrayPadroesTreina.clear();
                arrayPadroesTeste.clear();
                fis = new FileInputStream(file);
                BufferedReader d =
                    new BufferedReader(new InputStreamReader(fis));

                if (n == 1) {
                    for (int c = 0; c < 16; c++) {
                        line = d.readLine();
                        arrayPadroesTeste.add(line);
                    }
                    line = d.readLine();
                    while (line != null) {
                        arrayPadroesTreina.add(line);
                        line = d.readLine();
                    }
                }
            }
        }
    }
}
```

```
}  
  
if (n == 2) {  
    for (int c = 0; c < 16; c++) {  
        line = d.readLine();  
        arrayPadroesTreina.add(line);  
    }  
    for (int c = 0; c < 16; c++) {  
        line = d.readLine();  
        arrayPadroesTeste.add(line);  
    }  
    line = d.readLine();  
    while (line != null) {  
        arrayPadroesTreina.add(line);  
        line = d.readLine();  
    }  
}  
  
if (n == 3) {  
    for (int c = 0; c < 32; c++) {  
        line = d.readLine();  
        arrayPadroesTreina.add(line);  
    }  
    for (int c = 0; c < 16; c++) {  
        line = d.readLine();  
        arrayPadroesTeste.add(line);  
    }  
    line = d.readLine();  
    while (line != null) {  
        arrayPadroesTreina.add(line);  
        line = d.readLine();  
    }  
}  
  
if (n == 4) {  
    for (int c = 0; c < 48; c++) {  
        line = d.readLine();  
        arrayPadroesTreina.add(line);  
    }  
    for (int c = 0; c < 16; c++) {  
        line = d.readLine();  
        arrayPadroesTeste.add(line);  
    }  
    line = d.readLine();  
    while (line != null) {  
        arrayPadroesTreina.add(line);  
        line = d.readLine();  
    }  
}  
  
if (n == 5) {  
    for (int c = 0; c < 64; c++) {  
        line = d.readLine();  
        arrayPadroesTreina.add(line);  
    }  
    for (int c = 0; c < 16; c++) {  
        line = d.readLine();  
        arrayPadroesTeste.add(line);  
    }  
    line = d.readLine();  
}
```

```
        while (line != null) {
            arrayPadroesTreina.add(line);
            line = d.readLine();
        }
    }

    if (n == 6) {
        for (int c = 0; c < 80; c++) {
            line = d.readLine();
            arrayPadroesTreina.add(line);
        }
        for (int c = 0; c < 16; c++) {
            line = d.readLine();
            arrayPadroesTeste.add(line);
        }
        line = d.readLine();
        while (line != null) {
            arrayPadroesTreina.add(line);
            line = d.readLine();
        }
    }

    if (n == 7) {
        for (int c = 0; c < 96; c++) {
            line = d.readLine();
            arrayPadroesTreina.add(line);
        }
        for (int c = 0; c < 16; c++) {
            line = d.readLine();
            arrayPadroesTeste.add(line);
        }
        line = d.readLine();
        while (line != null) {
            arrayPadroesTreina.add(line);
            line = d.readLine();
        }
    }

    if (n == 8) {
        for (int c = 0; c < 112; c++) {
            line = d.readLine();
            arrayPadroesTreina.add(line);
        }
        for (int c = 0; c < 15; c++) {
            line = d.readLine();
            arrayPadroesTeste.add(line);
        }
        line = d.readLine();
        while (line != null) {
            arrayPadroesTreina.add(line);
            line = d.readLine();
        }
    }

    if (n == 9) {
        for (int c = 0; c < 127; c++) {
            line = d.readLine();
            arrayPadroesTreina.add(line);
        }
        for (int c = 0; c < 15; c++) {
```



```

        line = d.readLine();
        arrayPadroesTeste.add(line);
    }
    line = d.readLine();
    while (line != null) {
        arrayPadroesTreina.add(line);
        line = d.readLine();
    }
}

if (n == 10) {
    for (int c = 0; c < 142; c++) {
        line = d.readLine();
        arrayPadroesTreina.add(line);
    }

    line = d.readLine();
    while (line != null) {
        arrayPadroesTeste.add(line);
        line = d.readLine();
    }
}

//Armazena as distâncias de padrões diferentes
dist.clear();
for (int c = 0; c < arrayPadroesTreina.size(); c++) {
    String padrao1 = (String)
arrayPadroesTreina.get(c);

    String tipo1 = padrao1.substring(24, 25);

    for (int j = 0; j < arrayPadroesTreina.size();
j++) {
        String padrao2 = (String)
arrayPadroesTreina.get(j);

        String tipo2 = padrao2.substring(24, 25);
        double de =

        KnnCross3.distanciaEuclidiana(padrao1, padrao2);
        if (!tipo1.equals(tipo2)) {
            dist.add(de + " ;" + padrao1 + ";" +
padrao2);
        }
    }
}
Collections.sort(dist);

while ((erro > 0) && (k < 500)) {
    String dstMenor = (String) dist.get(k);
    StringTokenizer str = new
StringTokenizer(dstMenor, ";");
    String padrao = str.nextToken();
    String p1 = str.nextToken();
    String p2 = str.nextToken();

    if (!J.contains(p1)) {
        J.add(p1);
    }
    if (!J.contains(p2)) {
        J.add(p2);
    }
}

```

```

        erro = acharErro(J, arrayPadroesTreina);
        System.out.println("erro: " + erro);
        k++;
        System.out.println("k= " + k);
    }

    System.out.println("J: " + J.size());

    double erro2 = acharErro2(J, arrayPadroesTeste);
    System.out.println("erro final: " + erro2);
}
} catch (IOException ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}
}

public double acharErro(ArrayList arrayJ, ArrayList arrayTreina) {

    ArrayList dist = new ArrayList();
    int cont1 = 0;
    int contTotal1 = 0;
    for (int i = 0; i < arrayTreina.size(); i++) {
        String padrao1 = (String) arrayTreina.get(i);
        String tipo1 = padrao1.substring(24, 25);
        dist.clear();

        for (int j = 0; j < arrayJ.size(); j++) {
            String padrao2 = (String) arrayJ.get(j);
            int tipo2 = Integer.parseInt(padrao2.substring(24,
25));

            double de = KnnCross3.distanciaEuclidiana(padrao1,
padrao2);

            dist.add(de + " ;" + tipo2);
        }

        Collections.sort(dist);
        String dstMenor = (String) dist.get(0);
        StringTokenizer str = new StringTokenizer(dstMenor, ";");
        String confereTipo = str.nextToken();
        confereTipo = str.nextToken();

        if (confereTipo.equals(tipo1)) {
            //System.out.println("Acertou");
            cont1++;
            contTotal1++;
        } else {
            //System.out.println("Erro");
            contTotal1++;
        }
    }
    System.out.println("cont: " + cont1);
    System.out.println("contTotal: " + contTotal1);
    double f = 100 - ((100 * cont1) / contTotal1);
    return (f);
}

public double acharErro2(ArrayList arrayJ, ArrayList arrayTreina) {

```

```

ArrayList dist = new ArrayList();
//int cont = 0;
//int contTotal = 0;
for (int i = 0; i < arrayTreina.size(); i++) {
    String padrao1 = (String) arrayTreina.get(i);
    String tipo1 = padrao1.substring(24, 25);
    dist.clear();

    for (int j = 0; j < arrayJ.size(); j++) {
        String padrao2 = (String) arrayJ.get(j);
        int tipo2 = Integer.parseInt(padrao2.substring(24,
25));

        double de = PrincipalKNN.distanciaEuclidiana(padrao1,
padrao2);

        dist.add(de + " ;" + tipo2);
    }

    Collections.sort(dist);
    String dstMenor = (String) dist.get(0);
    StringTokenizer str = new StringTokenizer(dstMenor, ";");
    String confereTipo = str.nextToken();
    confereTipo = str.nextToken();

    if (confereTipo.equals(tipo1)) {
        //System.out.println("Acertou");
        cont++;
        contTotal++;
    } else {
        //System.out.println("Errou");
        contTotal++;
    }
}
System.out.println("cont: " + cont);
System.out.println("contTotal: " + contTotal);
double f = 100 - ((100 * cont) / contTotal);
return (f);
}

public static void main(String[] args) {
    NNSRMCross a = new NNSRMCross();
    a.algoritmo();
}
}

```

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.