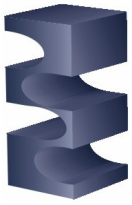


Departamento de Sistemas Computacionais

POLI

ESCOLA POLITÉCNICA DE PERNAMBUCO

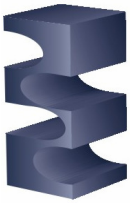
MÉTODOS BASEADOS NA REGRA DO VIZINHO MAIS PRÓXIMO PARA RECONHECIMENTO DE IMAGENS

Trabalho de Conclusão de Curso

Engenharia da Computação

Miguel Eugênio Ramalho Bezerra
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, 06 de janeiro de 2006



Departamento de Sistemas Computacionais

ESCOLA POLITÉCNICA DE PERNAMBUCO

MÉTODOS BASEADOS NA REGRA DO VIZINHO MAIS PRÓXIMO PARA RECONHECIMENTO DE IMAGENS

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.

Miguel Eugênio Ramalho Bezerra
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, 06 de janeiro de 2006

Miguel Eugênio Ramalho Bezerra

MÉTODOS BASEADOS NA REGRA DO VIZINHO MAIS PRÓXIMO PARA RECONHECIMENTO DE IMAGENS

RESUMO

Existem várias técnicas de inteligência artificial (IA) que têm como objetivo a classificação de padrões. Classificar padrões significa classificar um padrão desconhecido dentre várias classes possíveis. O uso de técnicas de aprendizado de máquina para classificação de padrões vem aumentando nos últimos anos e já existem diversas áreas onde o benefício advindo desse tipo de técnica é bastante positivo. Uma dessas áreas é a de reconhecimento de imagens. O objetivo deste trabalho é estudar, implementar e comparar algoritmos de aprendizagem de máquina baseados na regra do vizinho mais próximo para a classificação de imagens. Algoritmos baseados na regra do vizinho mais próximo, como *K-Nearest Neighbor* (KNN) e *Pairwise Opposite Class-Nearest Neighbor* (POC-NN), foram implementados para classificação de imagens e seu desempenho foi comparado com um método construtivo para treinamento de redes neurais do tipo RBF (*Radial Basis Function*). Também foram propostas e implementadas duas modificações inéditas para o algoritmo POC-NN. Essas modificações geraram dois novos algoritmos: o S-POC-KNN e S-POC-NN com núcleo RBF. O desempenho desses é comparado com o algoritmo original e com o RBF-DDA-SP.

Palavras-chave: Regra do Vizinho Mais Próximo. POC-NN. Aprendizagem de Máquina. Reconhecimento de Padrões. Reconhecimento de Imagens.

ABSTRACT

There are many techniques of Artificial Intelligence (AI) which focus the classification of patterns. Classifying patterns means to classify an unknown pattern among a several possible classes. The use of machine learning techniques for making the pattern classification has been increasing in the latest years and there are lots of areas where the benefit that came from this kind of technique is very positive. One of these areas is the image recognition. This work goal is studying, making a comparison and implementing algorithms of machine learning based on the rule of the nearest neighbor for the classification of images. Algorithms based on the nearest neighbor rule *K-Nearest Neighbor* (KNN) and *Pairwise Opposite Class-Nearest Neighbor* (POC-NN) were implemented for the classification of images and its performance was compared with a constructive method for RBF (Radial Basis Function) network training. In addition, it was proposed and implemented two unheard modifications for the POC-NN algorithm. These modifications created two new algorithms: the S-POC-KNN and S-POC-NN with a kernel RBF. The performance of both algorithms is compared with the original algorithm and with the RBF-DDA-SP.

Keywords: Nearest Neighbor Rule. POC-NN. Machine Learning. Pattern Recognition. Image Recognition.

Sumário

Índice de Figuras

Índice de Tabelas

Índice de Gráficos

Tabela de Símbolos e Siglas

IA	Inteligência Artificial
RNA	Rede Neural Artifical
NN	*Nearest Neighbor* (Regra do Vizinho mais próximo)
KNN	*K-Nearest Neighbor* (Regra dos K-Vizinhos mais próximo)
POC-NN	*Pairwise Opposite Class-Nearest Neighbor*
OCR	*Optical Character Recognition* (Reconhecimento óptico de caracteres)
RBF	*Radial Basis Function* (Função de base radial)
MCP	McCulloch *and* Pitts Perceptron (Perceptron de McCulloch e Pitts)
PDA	*Personal Digital Assistant* (Assitente pessoal digital)
η	Taxa de aprendizado de uma RNA
DDA	*Dynamic Decay Adjustment* (Ajuste de decaimento dinâmico)
θ^+	Limiar positivo do RBF-DDA
θ^-	Limiar negativo do RBF-DDA
RBF-DDA	Rede neural do tipo RBF com *Dynamic Decay Adjustment*
RBF-DDA-SP	Rede neural do tipo RBF com *Dynamic Decay Adjustment* e poda seletiva
$\|X - X_k\|$	Distância Euclidiana entre dois padrões X e X_k
$\Gamma(x)$	Função Gamma
S-POC-NN	POC-NN com seleção de protótipos
R-POC-NN	POC-NN com troca de protótipos
A	Intervalo de aceitação do POC-NN
α-Ratio	Parâmetro para calculo do intervalo de aceitação
1-n-1	*One-Against-One*
S-POC-KNN	Junção do S-POC-NN com o KNN
NNSRM	Nearest Neighbor Structural Risk Minimization
S-POC-KNN-RBF	Junção do S-POC-NN com o KNN e com o núcleo RBF
τ	Parâmetro de ajuste do desempenho do núcleo RBF

Agradecimentos

Primeiramente agradeço a Deus por me dar saúde e sabedoria em todos os momentos. Aos meus pais e ao meu irmão por todo o apoio, sem eles eu não teria conseguido. À minha namorada Patrícia por toda ajuda, incentivo e compreensão. Agradeço também à toda minha família e amigos, sempre presentes.

Sou grato a todos aqueles que contribuíram para este trabalho. Em particular, a todos os professores do curso de graduação em Engenharia da Computação da Universidade de Pernambuco por suas valiosas informações transmitidas durante o período de aprendizado, as quais serviram como base de conhecimento e fonte de inspiração.

Desejo agradecer especialmente ao meu professor orientador, Prof. Adriano Lorena, pela dedicação com que empregou seu conhecimento, supervisão e incentivo para a conclusão deste trabalho e por ter me ajudado no momento mais difícil deste. Também gostaria de agradecer à Profa. Maria Lencastre pelas orientações iniciais, pela dedicação e pelo incentivo dado quando tudo parecia perdido.

Capítulo 1
Introdução

Inteligência Artificial (IA) [1] é uma das áreas mais importantes do ramo da computação atualmente. Ela pode ser definida de forma simplista como "a inteligência exibida por qualquer coisa que tenha sido construída pelo homem" [32], no entanto, este conceito leva ao questionamento do que seria "inteligência". Segundo o dicionário Aurélio Buarque de Holanda [11], inteligência pode ser definida como "Faculdade de aprender; qualidade ou capacidade de compreender e adaptar-se facilmente" e uma definição mais precisa para IA seria "Ramo da ciência da computação dedicado a desenvolver equivalentes computacionais de processos peculiares à cognição humana" [11].

IA tem demonstrado bastante sucesso em áreas onde é possível se construir abstrações do mundo real. Geralmente, tais abstrações se baseiam em modelos simples que, através de uma abordagem *bottom-up*, buscam modelar sistemas complexos do mundo real, tentando imitar os sistemas naturais em algumas de suas características.

A área de IA é subdividida em diversos ramos, sendo os principais: a aprendizagem de máquina, IA simbólica, as redes neurais artificiais (RNA), dentre outros. O foco deste trabalho é a utilização de técnicas de aprendizagem de máquina, em particular, as técnicas baseadas na regra do vizinho mais próximo.

Técnicas de aprendizagem de máquina têm como objetivo a classificação de padrões. Um algoritmo de aprendizagem de máquina para classificação de padrões tem como meta classificar padrões desconhecidos dentre as várias classes possíveis de um determinado problema.

Diversas aplicações utilizam técnicas de IA para a classificação de imagens por uma máquina. Várias pesquisas tentam aperfeiçoar o desempenho dos classificadores para o reconhecimento de imagens. Dentre os problemas mais relevantes de reconhecimento de imagens podemos citar: o reconhecimento de letras e dígitos manuscritos, a classificação de imagens de satélite, a biometria, dentre outros.

No decorrer da monografia são apresentados e propostos algoritmos de treinamento de máquina baseados na regra do vizinho mais próximo, originalmente chamada de *Nearest Neighbor* (NN) [31], com o objetivo de classificação de padrões para o reconhecimento de imagens. Além disso, apresentamos resultados de simulações usando nossas implementações desses algoritmos.

Os algoritmos baseados na regra do vizinho mais próximo necessitam de recursos computacionais significantes, dentre esses os mais requisitados são recursos de memória e de processamento, como é o caso do algoritmo *K-Nearest Neighbor* (KNN) [31] . Existem muitos algoritmos baseados na regra do vizinho mais próximo que buscam diminuir o tempo computacional e os recursos necessários. Vale citar como exemplo o *Pairwise Opposite*

Class-Nearest Neighbor (POC-NN) [26][27], que é usado como base para os métodos propostos neste trabalho.

Portanto, este trabalho possui os seguintes objetivos:

1. Implementar técnicas baseadas na regra do vizinho mais próximo proposta nas referências [7][27][31];
2. Propor novos algoritmos baseados na técnica do POC-NN;
3. Avaliar os algoritmos propostos, comparando-os com os demais algoritmos baseados no NN, em problemas de classificação de imagens;
4. Comparar os algoritmos baseados em vizinhos mais próximos, considerados neste trabalho, com redes neurais do tipo RBF-DDA-SP, tanto com relação ao desempenho de classificação quanto em relação à complexidade.

Capítulo 2

Aprendizagem de Máquina

Este capítulo mostra uma visão geral sobre aprendizagem de máquina e o contexto de Inteligência Artificial no qual este tema está inserido. São analisadas também as dificuldades que circundam o reconhecimento de imagens por parte de uma máquina. Por fim, são examinados alguns conceitos de redes neurais artificiais e redes neurais artificiais do tipo RBF (*Radial Basis Function*).

2.1 Contexto Histórico

O primeiro trabalho na área de redes neurais artificiais foi o do neurônio artificial de McCulloch e Pitts chamado de MCP (McCulloch *and* Pitts Perceptron). O trabalho foi publicado em 1943 e intitulado *A Logical Calculus of the Ideas Immament in Nervous Activity*. Segundo [9]: "O trabalho de McCulloch e Pitts se concentrava muito mais em descrever um modelo artificial de um neurônio e apresentar suas capacidades computacionais do que em apresentar técnicas de aprendizado".

Em 1958, Rosenblatt criou um modelo chamado *Perceptron* que usava o MCP e permitia a realização de treinamentos para classificação de padrões. No entanto, o *Perceptron* possuía um ponto fraco: só era capaz de classificar padrões linearmente separáveis; ou seja, ele só resolvia problemas cuja solução podia ser obtida dividindo-se o espaço de entrada em duas regiões através de uma reta. Na Figura 1, pode-se observar padrões de duas classes, quadrado ( ) e bola (**O**), de forma que, na Figura 1 (a) esses padrões estão dispostos de forma linearmente separável e na Figura 1 (b) estão dispostos de forma não linearmente separável.

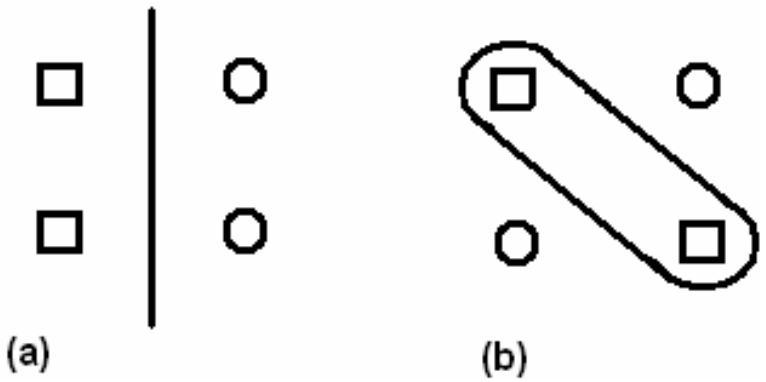


Figura 1. (a) Problema linearmente separável. (b) Problema não linearmente separável.

Em 1966, a regra do vizinho mais próximo, *Nearest Neighbor Rule* (NN), foi proposta por Cover e Hart. Eles mostraram que essa técnica era muito eficiente para problemas de classificação de padrões. Um outro motivo para a utilização da regra do vizinho mais próximo seria que ela é simples e de fácil implementação, mas possui alguns pontos fracos, tais como: (a) requerer uma grande quantidade de memória, pois necessita armazenar todos os padrões de treinamento; (b) o algoritmo requer uma grande quantidade de tempo computacional; (c) é sensível a ruído e falsos padrões [26].

Uma variação bastante antiga e consagrada para o algoritmo NN é *K-Nearest Neighbor*, que, apesar de possuir os mesmos problemas inerentes ao algoritmo NN original, em alguns casos apresenta resultados melhores do que o algoritmo NN original. Uma variação recente e inovadora para o algoritmo NN é o *Pairwise Opposite Class-Nearest Neighbor*. Detalhes sobre esse algoritmo são apresentados no Capítulo 3.

2.2 Reconhecimento de Imagens

Uma boa definição para o reconhecimento de imagens é dada em [24]: "Diversas denominações têm sido dadas a este campo multidisciplinar que aproveita os conhecimentos relacionados com o Processamento de Sinais, Inteligência Computacional, Neurofisiologia e outros [...] De uma forma geral, é o processo de cognição de uma imagem, que pertence à área do Reconhecimento e Análise de Padrões".

A partir da década de 80, com os avanços verificados no campo da eletrônica, as técnicas digitais passaram a ser empregadas com o intuito de se obter um mecanismo para classificação de imagens. Atualmente as técnicas de inteligência artificial vêm sendo empregadas, permitindo novas abordagens para o problema [24].

Nos últimos anos, a área de processamento e reconhecimento de imagens ganhou grande visibilidade devido ao amadurecimento das técnicas de inteligência artificial. Existem diversas aplicações possíveis para o reconhecimento de imagens. Um exemplo bastante popular são as técnicas de OCR [22] (*Optical Character Recognition*), que executam o reconhecimento óptico de caracteres em imagens de texto. Ao final do processo de OCR é gerado um arquivo de texto correspondente à imagem digitalizada.

Mesmo com toda a tecnologia atual, muitas tarefas do cotidiano ainda utilizam escrita manual. Tarefas bastante corriqueiras como o preenchimento de um cheque, a escrita de endereços em envelopes postais e até mesmo notas escritas em um PDA (*Personal Digital Assistants*) utilizam caracteres manuscritos. Dessas tarefas cotidianas surgiu a necessidade de mecanismos que fizessem com que uma máquina fosse capaz de reconhecer caracteres escritos à mão. Com essa finalidade, surgiram as técnicas de reconhecimento de caracteres manuscritos [25].

As técnicas de reconhecimento de imagens geralmente necessitam usar técnicas de processamento de imagens para pré-processar os dados contidos na imagem e apresentá-los de forma que possam ser tratados pelo algoritmo de treinamento. Pode-se ver na Figura 2, retirada de [24], as fases pelas quais uma imagem tem que passar, na maioria dos sistemas, antes que se possa extrair as informações nela contidas. Inicialmente a imagem sofre um pré-processamento, onde ocorre uma série de transformações que retiram da imagem alguns elementos desnecessários, como por exemplo, o fundo da imagem, que não é interessante para o sistema. Após o pré-processamento, a imagem terá suas características extraídas através da utilização de métodos estatísticos. Um fato importante a ser observado é que o número de características varia de acordo com o problema em análise, como será visto no Capítulo 5. Após as características serem extraídas, o classificador será capaz de utilizar diversas técnicas de inteligência computacional para realizar a classificação dos padrões que representam cada

imagem. Um exemplo de imagem que poderia ser usada como entrada do sistema da Figura 2 é o de caracteres escritos à mão, o que pode ser visto na Figura 3, obtida de [25].

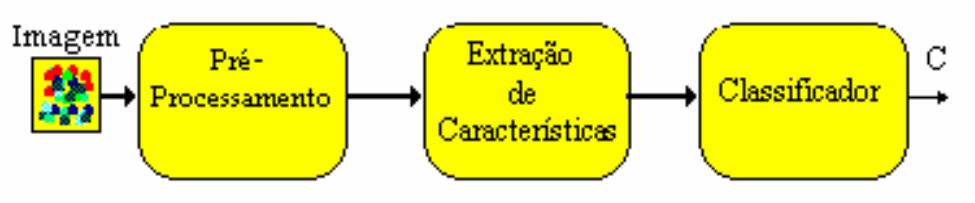


Figura 2. Sistema que realiza pré-processamento, extração de características e classificação de imagens.

No mundo inteiro, diversos sistemas de autenticação tentam aumentar a segurança no acesso à infra-estrutura das empresas. Autenticação é uma forma de evitar o acesso não autorizado a informações e a alguns locais restritos dentro de uma organização. Diversas técnicas são usadas com a finalidade de se obter um maior nível de segurança na autenticação. Uma dessas técnicas é a biometria [12]. Além da autenticação – que compara os dados da pessoa que quer entrar em um sistema com os dados da pessoa que ela diz ser – a biometria pode ser usada em sistemas de criminalística para identificação de pessoas. Para essa finalidade o sistema realiza uma busca, num determinado banco de dados, por informações iguais as da pessoa que se deseja identificar, até que se encontre a correta.

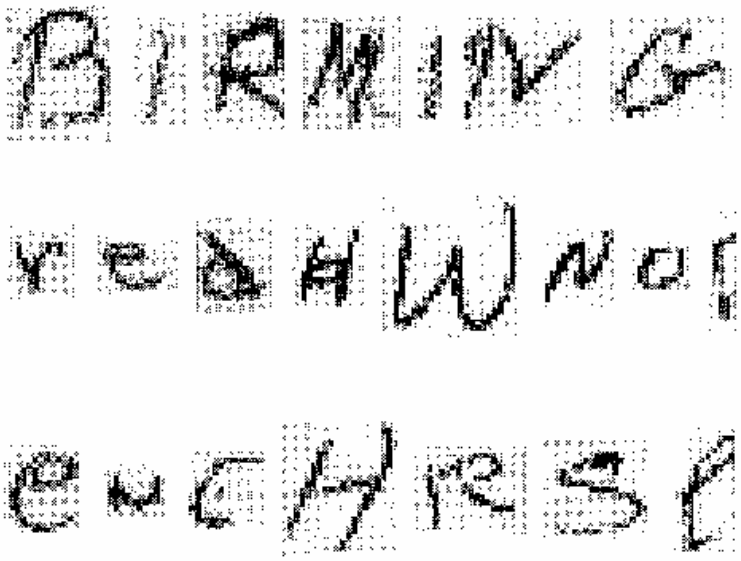


Figura 3. Exemplos de caracteres escritos à mão segmentados de uma imagem.

Existem várias características que podem ser utilizadas para realizar o reconhecimento biométrico automático de pessoas. Impressões digitais, íris, face e outras características podem ser usadas em sistemas que utilizam a biometria como forma de identificação. Porém, cada uma delas possui suas dificuldades e peculiaridades. A identificação de impressões digitais, por exemplo, tem sua identificação dificultada pela forma como a digital é capturada. Existem diversas dificuldades inerentes ao processo de captura, como exemplo, a impressão digital pode ser capturada de forma parcial, em ângulos diferentes (Figura 4) ou até mesmo com distorções advindas da pressão que é feita sobre a superfície de captura [12].

Figura 4. Uma mesma digital capturada em ângulos diferentes em imagens distintas [25].

2.2.1 Classificação de Padrões

Nos últimos anos, novas aplicações para classificação de padrões surgiram, principalmente na área de classificação de imagens. Podem ser citadas como exemplo: a classificação de dígitos manuscritos, a classificação de faces e de impressões digitais (biometria), a classificação de caracteres, classificação de imagens de satélite, dentre outras.

Antes de iniciar o treinamento de um algoritmo, deve-se lançar mão do uso de algumas técnicas de comparação de classificadores, como é o caso da técnica de *holdout* [28], que foi usada neste estudo. A técnica de *holdout* é simples e computacionalmente rápida, consistindo na divisão da base de dados original em dois conjuntos: um de treinamento e outro de teste.

Alguns fatores advindos da divisão do *holdout,* como a quantidade de padrões contidos nos conjuntos de treinamento e teste, a quantidade de padrões de uma mesma classe armazenados no conjunto de treinamento, podem influenciar na avaliação final do algoritmo.

Pode-se fazer a seguinte análise com relação à quantidade de padrões de uma mesma classe armazenados no conjunto de treinamento: caso o conjunto de treinamento possua mais padrões de uma dada classe “X”, durante a fase de teste o algoritmo terá uma maior capacidade de generalização para classificação de padrões dessa determinada classe “X”. Dessa forma, o ideal seria que o conjunto de treinamento possuísse padrões representativos para cada uma das várias classes consideradas [4].

Na classificação de padrões, o algoritmo responsável pela classificação deve primeiramente “aprender” como classificar os padrões do problema ao qual se deseja obter resposta, ou seja, deve-se primeiramente treinar o algoritmo de forma a torná-lo capaz de, após o treinamento, classificar um padrão desconhecido dentre uma das classes existentes. Nessa fase, é usado o conjunto de treinamento advindo do *holdout.*

A fase de treinamento possui um peso muito grande no desempenho do algoritmo como um todo. Ou seja, durante essa fase o algoritmo deve armazenar protótipos (que podem ser padrões de treinamento) que sejam capazes de generalizar ao máximo a classificação feita para os padrões desconhecidos.

Após a fase treinamento o algoritmo deve passar pela fase de teste. Essa fase é responsável por medir o desempenho da classificação feita pelo algoritmo, ou seja, o

desempenho do classificador é avaliado antes de ser usado na prática. Ao final da fase de teste, devem ser apresentadas as porcentagens de padrões incorretamente classificados, para, dessa forma, se analisar o desempenho do classificador para o problema em questão. Durante a fase de teste é usado o conjunto de teste advindo do *holdout*.

2.3 Redes Neurais Artificiais

A humanidade tem aprendido muito através da imitação dos fatos da natureza. Atualmente o homem tenta imitar algumas características da cognição humana e da inteligência do cérebro através de sistemas que utilizam técnicas de inteligência artificial. Dessa forma, as Redes Neurais Artificiais (RNAs) vêm sendo usadas em diversas áreas onde é possível se construir abstrações do mundo real. Diferentemente da programação algorítmica convencional, baseada em regras, as RNAs tentam imitar o funcionamento do cérebro humano. Assim como o cérebro humano é composto por diversos neurônios que trabalham em paralelo, as RNAs são compostas por diversas unidades de processamento simples chamadas de nodos, que também trabalham em paralelo, e que calculam determinadas funções matemáticas.

O cérebro humano possui cerca de 10^{11} neurônios, segundo [4], que são as suas células fundamentais. Os neurônios são divididos em três partes conforme destacado na Figura 5 de [8]: corpo da celular, dendritos e axônios. O neurônio humano transmite informações na forma de sinais elétricos através de seu axônio, e essas informações são então repassadas através das sinapses químicas para os dendritos do neurônio subseqüente. Assim, quando esses sinais chegam aos dendritos da célula seguinte, passam então ao corpo celular e posteriormente para o axônio, iniciando novamente o processo de transmissão de informações [13]. Conjuntamente esses processos de transmissão vão formar as redes neurais biológicas, como pode ser visto na Figura 6 de [8].

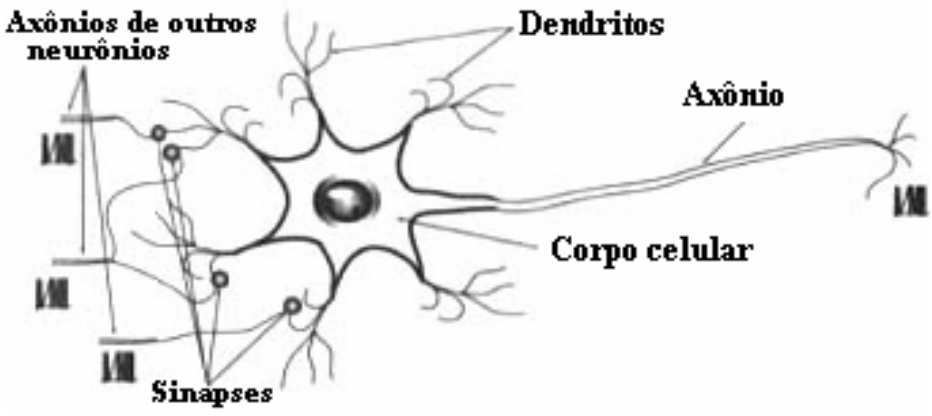


Figura 5. Neurônio humano.

Os nodos de uma RNA são dispostos em uma ou mais camadas, de forma semelhante às redes neurais biológicas. As camadas de uma RNA são interconectadas por um grande de número de conexões, onde cada conexão está associada a um peso que representa o conhecimento armazenado nela. Na Figura 7 é exibida a representação de uma RNA com múltiplos nodos agrupados em três camadas interligadas por conexões unidirecionais.

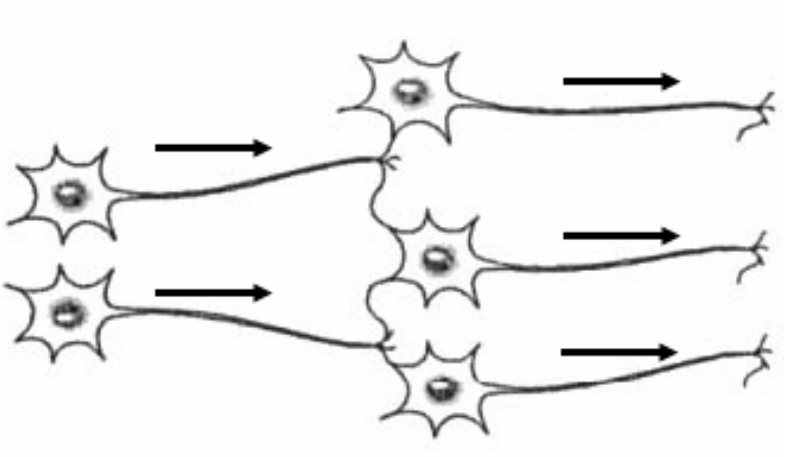


Figura 6. Rede neural biológica transmitindo informações.

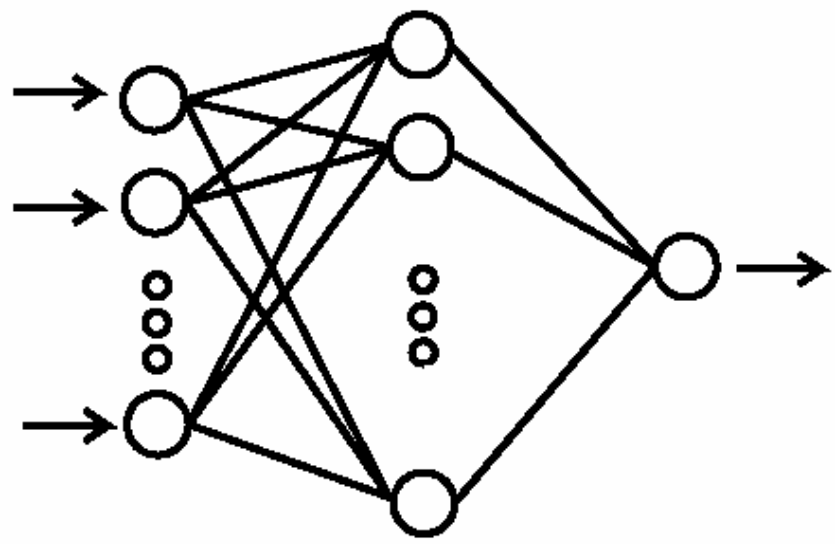


Figura 7. Rede neural artificial.

Para simular o comportamento das sinapses, as entradas de um neurônio artificial possuem pesos, que podem possuir valores positivos ou negativos, sendo estes valores inibitórios ou excitatórios, respectivamente. Na Figura 8 existe um neurônio artificial do tipo MCP com três entradas, que descreve esse modelo em questão.

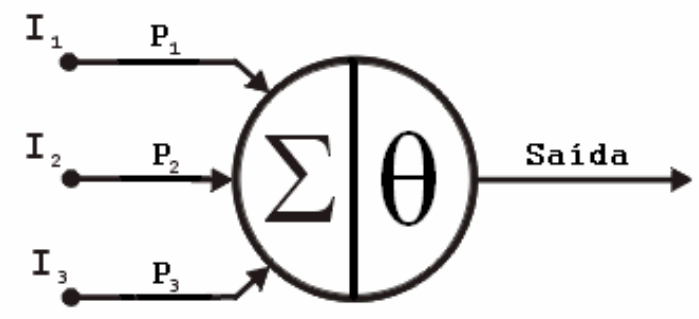


Figura 8. Neurônio artificial com três entradas.

Os nodos de uma RNA possuem uma função de ativação que faz a comparação da soma ponderada das entradas com um valor de limiar, também conhecido como *threshold*. Se a soma das entradas for maior do que o *threshold*, a saída do neurônio é ativada, permanecendo desativada caso contrário. Nos neurônios do tipo MCP a função de ativação é dada pela função limiar, descrita formalmente pela Equação 2.1, onde n é a quantidade de entradas, P_i é o peso associado a cada entrada I_i e θ é o *threshold* do neurônio. Na Figura 24 podem ser vistos quatro tipos de função de ativação [4]: a função linear, a função rampa, a função degrau e a função sigmoidal.

$$y = \begin{cases} 1, \text{ se } \sum_{i=0}^{n} P_i I_i \geq \theta \\ 0, \text{ se } \sum_{i=0}^{n} P_i I_i < \theta \end{cases} \tag{2.1}$$

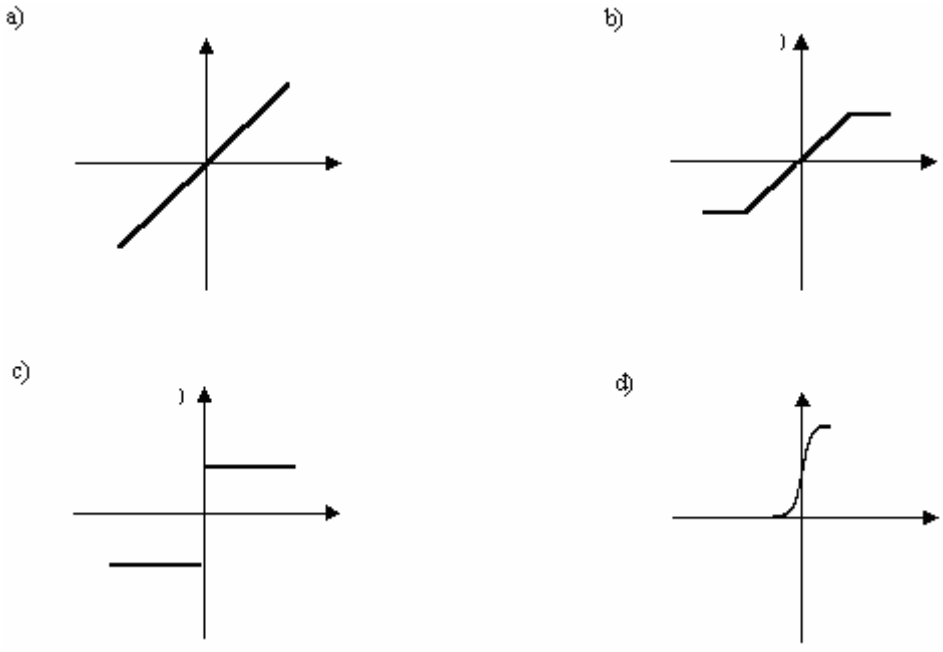


Figura 9. Quatro tipos de funções de ativação: (a) a função linear, (b) a função rampa, (c) a função degrau e (d) a função sigmoidal.

Um fator de suma importância em RNAs é a definição da arquitetura que será utilizada pela rede, pois dependendo do tipo de arquitetura escolhida, uma RNA pode estar habilitada ou não a resolver determinados tipos de problema, como por exemplo, problemas não-linearmente separáveis. Tendo em vista o impacto inerente a escolha da arquitetura de uma RNA, Braga [4] apresenta alguns exemplos de arquitetura na Figura 10 e analisa os quatro parâmetros que definem uma arquitetura: o número de camadas da rede, o número de nodos em cada camada, o tipo de conexão entre os nodos e a topologia da rede. Com relação ao número de camadas, em uma rede com apenas uma camada só existe um nó entre qualquer entrada e qualquer saída (Figura 10 (a), (e)). Já em uma rede com múltiplas camadas, existe mais de um nó entre alguma entrada e alguma saída da rede (Figura 10 (b), (c), (d)). Quanto aos tipos de conexões, elas podem ser acíclicas ou cíclicas. Nas acíclicas, também conhecidas

como *feedforward*, a saída de um nodo na *n*-ésima camada da rede não pode ser usada como entrada de nodos em camadas de índice menor ou igual a *n* (Figura 10 (a), (b), (c)); nas cíclicas, também conhecidas como *feedback*, a saída de um nodo na *n*-ésima camada da rede é usada como entrada de nodos em camadas com índice menor ou igual a *n* (Figura 10 (d), (e)). As redes também podem ser classificadas como fracamente (ou parcialmente) conectada (Figura 10 (b), (c),(d)) e como completamente conectada (Figura 10 (a), (e)).

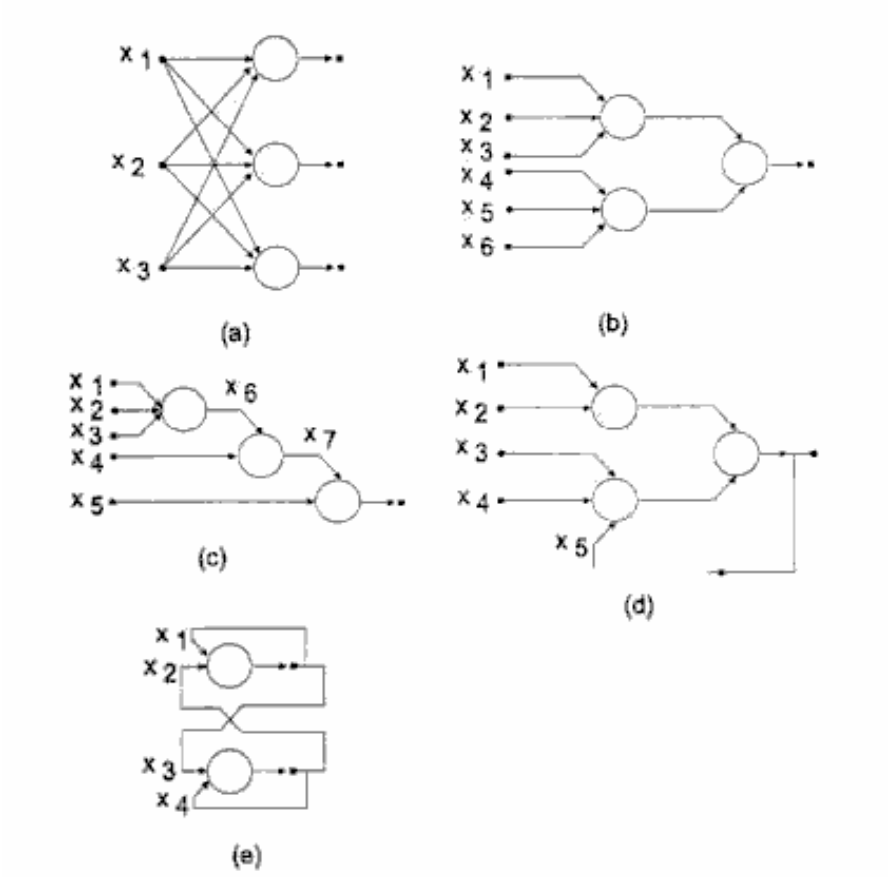


Figura 10. Arquiteturas de RNAs.

2.3.1 Aprendizado de uma Rede Neural Artificial

A solução de problemas utilizando RNAs passa inicialmente por uma fase onde a rede tem que ser treinada. Essa etapa também é conhecida como fase de aprendizagem. No entanto, antes de se iniciar o treinamento, ocorre a preparação dos dados que servirão de entrada, assim como a escolha do modelo e da arquitetura de RNA a serem utilizadas. Um fato importante a ser observado é que o treinamento de uma RNA não possui um tempo determinado de execução, e durante ele é determinada a intensidade das conexões entre os neurônios. Ao final da fase treinamento, o ajuste que foi realizado nos pesos das conexões é usado para criar uma representação própria do problema em questão. Essa representação será usada posteriormente para classificar padrões desconhecidos.

Sobre a resolução de problemas através do uso de RNAs, Braga [4] destaca que: “A capacidade *aprender* através de exemplos e de *generalizar* a informação aprendida é, sem dúvida, o atrativo principal da solução de problemas através de RNAs”. A generalização das RNAs está ligada à sua capacidade de dar respostas coerentes para padrões não conhecidos.

Ou seja, a RNA armazena informações durante fase de treinamento e depois essas informações são usadas para classificar padrões desconhecidos.

Existem diversos algoritmos que são utilizados para o treinamento de RNAs. Esses algoritmos consistem de um processo iterativo de ajuste dos parâmetros da rede, e basicamente eles diferem um do outro pela maneira como é feito o ajuste dos pesos. De uma forma geral os métodos de treinamento podem ser agrupados em dois paradigmas: o aprendizado supervisionado e o aprendizado não supervisionado. Para análise dos dois paradigmas de aprendizado, nas subseções 2.3.1.1 e 2.3.1.2, são utilizadas adaptações de exemplos dados por Braga [4].

2.3.1.1 Aprendizado Supervisionado

O aprendizado supervisionado é o paradigma mais utilizado para treinamento de RNAs. Ele é chamado dessa forma porque a entrada e a saída desejada são fornecidas por um supervisor externo. A Figura 11 ilustra o funcionamento do mecanismo de aprendizagem supervisionada, mostrando que a saída calculada pela RNA para cada padrão é comparada com a saída desejada professor. Após isso ela recebe informações sobre o erro da resposta atual. Esse erro é usado pela RNA para fazer o ajuste dos pesos das conexões de forma a minimizar o erro a cada etapa do treinamento.

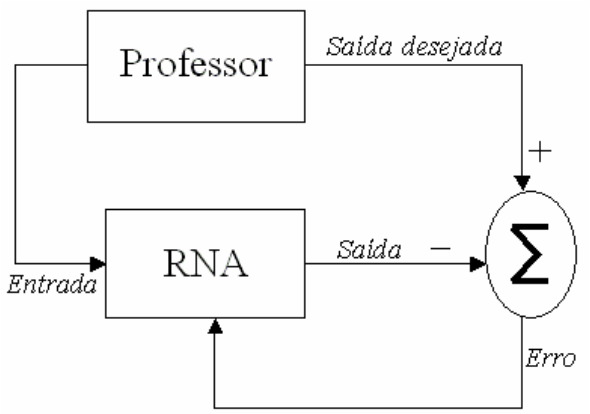


Figura 11. Diagrama do mecanismo de aprendizagem supervisionada.

O treinamento, quando é supervisionado, possui duas formas de implementação: *off-line e on-line*. No modo *off-line*, os padrões de treinamento não mudam e após a obtenção de uma solução para a rede, esta permanece fixa. Se o conjunto de treinamento sofrer adição de novos padrões, um novo treinamento deve ser realizado usando todos os padrões. No modo *on-line*, os padrões de treinamento mudam de forma contínua, o que faz com que a rede seja ajustada sucessivamente.

Como foi visto anteriormente, o ajuste dos pesos das conexões é feito de acordo com o erro advindo da diferença entre a resposta atual e a saída desejada. Esse erro é calculado como descrito na Equação 2.2, onde $e(t)$ é o erro, $d(t)$ é a saída desejada e $y(t)$ é a resposta calculada pela RNA no tempo t. Já o ajuste dos pesos é feito conforme a Equação 2.3, onde $w_i(t+1)$ é o peso ajustado, $w_i(t)$ é o peso atual, η é a taxa de aprendizado, $e(t)$ é o erro e $x_i(t)$ é a entrada do neurônio i no instante de tempo t. A taxa de aprendizado de uma RNA é uma constante de proporcionalidade no intervalo [0,1], onde o processo de ajuste do peso é proporcional a essa variável, que também influência na velocidade do treinamento.

$$e(t) = d(t) - y(t) \tag{2.2}$$

$$w_i(t+1) = w_i(t) + \eta e(t) x_i(t) \tag{2.3}$$

2.3.1.2 Aprendizado Não-supervisionado

No aprendizado não-supervisionado não existe a presença de um *professor* para orientar o processo de aprendizagem. Nesse tipo de aprendizado, os padrões são passados como entrada para rede sem se saber qual a saída desejada para cada um deles. A Rede então irá ajustar seus pesos de acordo com a regularidade estatística de suas entradas. Assim sendo, serão criadas codificações internas para o mapeamento de características e criadas novas classes e grupos automaticamente. Um fato importante é que esse tipo de aprendizagem só é possível quando existe redundância dos dados de entrada, pois sem redundância não seria possível encontrar padrões e características nos dados de entrada. A Figura 12 apresenta um diagrama do mecanismo de aprendizagem não-supervisionada.

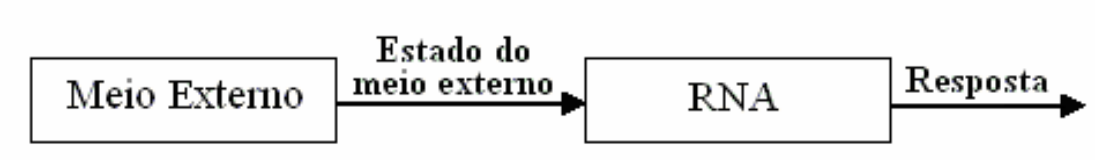


Figura 12. Diagrama do mecanismo de aprendizagem não-supervisionada.

2.3.2 Redes Neurais Artificiais do Tipo RBF

Nos últimos anos, RNAs do tipo RBF vêm sendo objeto de diversas pesquisas e estudos. Como resultado desse esforço, diversos algoritmos e melhorias têm sido propostos para otimizar as técnicas conhecidas até então. As redes RBF utilizam funções de base radial e são uma alternativa às redes multicamadas do tipo *feedforward* [8]. Uma rede RBF é uma função multidimensional não-linear que depende da distância dos vetores de entrada e seu vetor central.

As funções de base radial representam conjunto de funções cujo resultado varia de acordo com a distância em relação ao seu ponto central. Várias funções de base radial podem ser usadas em redes do tipo RBF. Dessa forma, são observados a seguir alguns exemplos dados por Gupta[8]:

(a) Função Gaussiana: $\phi(r) = e^{-(r/c)^2}$ (2.4)

(b) Função *cubic spline*: $\phi(r) = r^3$ (2.5)

(c) Função *thin-plate-spline*: $\phi(r) = r^2 \log(r)$ (2.6)

Dadas as Equações 2.4, 2.5 e 2.6, r é a distância euclidiana entre o vetor de entrada x e o centro da função radial c. A distância Euclidiana calcula a distância entre dois vetores no espaço n-dimensional e é dada pela Equação (2.7). A Figura 20, retirada de [8], mostra a representação gráfica das Equações 2.4, 2.5 e 2.6.

$$\|X - X_k\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2} \tag{2.7}$$

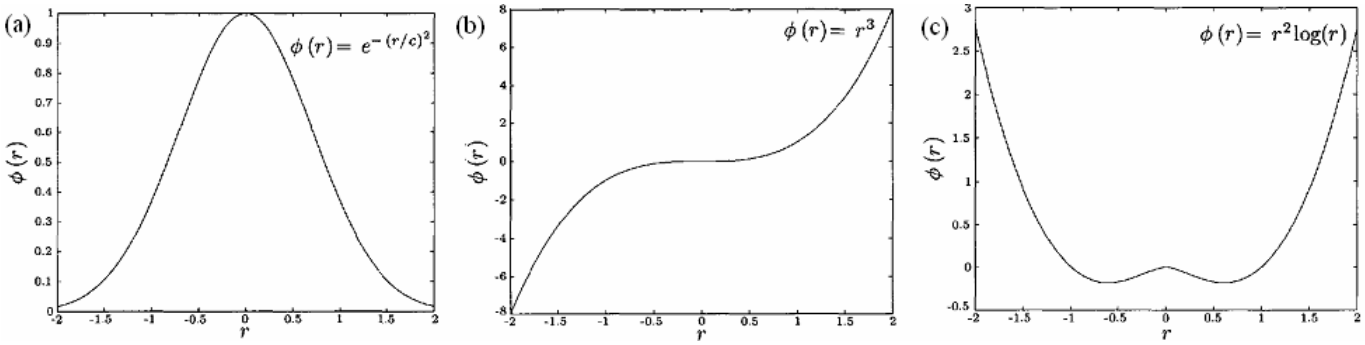


Figura 13. Representação gráfica das funções de base radial: (a) Função Gaussiana, (b) Função *cubic spline* e (c) Função *thin-plate-spline*.

Redes do tipo RBF geralmente possuem apenas uma camada intermediária, embora já tenha sido proposta a utilização de múltiplas camadas. Uma arquitetura típica de uma RNA com apenas uma camada intermediária pode ser vista na Figura 14, onde é importante frisar um detalhe nos nodos da camada intermediária que são funções de base radial. A camada intermediária e de saída desempenham um papel fundamental para uma rede tipo RBF. A camada intermediária recebe da camada de entrada um conjunto de padrões não linearmente separáveis. Esses padrões serão agrupados pela camada intermediária em um conjunto de saída com dados linearmente separáveis. A camada de saída por sua vez irá classificar os dados que foram recebidos da camada anterior.

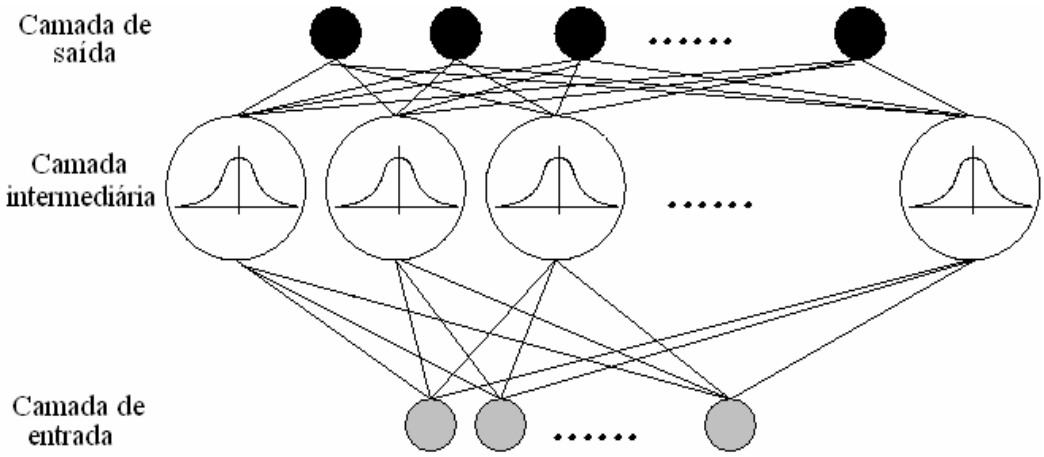


Figura 14. Arquitetura de uma RNA do tipo RBF

Cada nodo da camada intermediária define uma hiperelipsóide no espaço *n*-dimensional, as quais serão responsáveis por fazer a divisão do espaço de entrada, conforme ilustrada na Figura 15. Com a divisão do espaço de entrada, as redes RBF constroem então aproximadores locais, e assim irão classificar apenas padrões que sejam das mesmas classes apresentadas durante o treinamento, o que faz com que a rede não esteja susceptível a padrões espúrios - padrões adulterados ou falsos padrões.

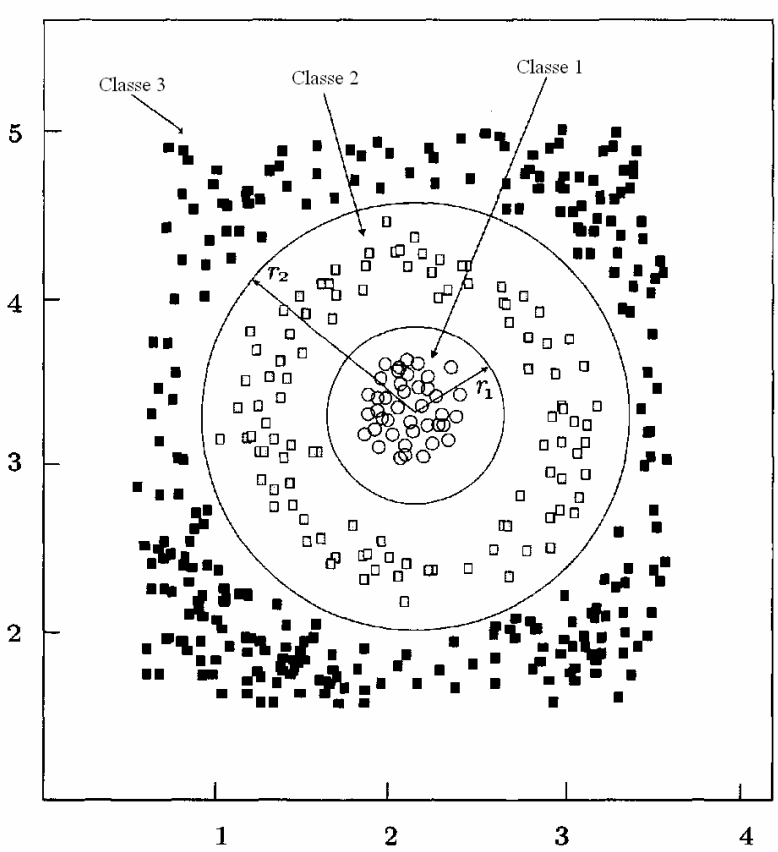


Figura 15. Divisão do espaço de entrada bidimensional feita por unidades RBF.

2.3.2.1 Algoritmo de treinamento DDA

Muitos algoritmos podem ser utilizados para fazer o treinamento de uma rede RBF. Porém, a maioria desses algoritmos possui uma arquitetura fixa que não permite modificação após o início do treinamento. Em redes RBF, pode-se utilizar um algoritmo para treinamento chamado DDA (*Dynamic Decay Adjustment*) [2] [16][20], que não possui uma arquitetura fixa como os outros algoritmos e realiza o treinamento de forma bastante rápida. O DDA foi criado a com intuito de resolver uma deficiência de um algoritmo chamado P-RCE (*Probabilistic Restricted Coulomb Energy*), que não classificava padrões em áreas de conflito. Área de conflito é uma região onde duas ou mais gaussianas se sobrepõem. Se um novo padrão da classe B ficasse em uma região desse tipo, como ilustra a Figura 16, ele não conseguiria ser classificado pelo P-RCE, porém o algoritmo DDA iria classificá-lo como sendo da classe B.

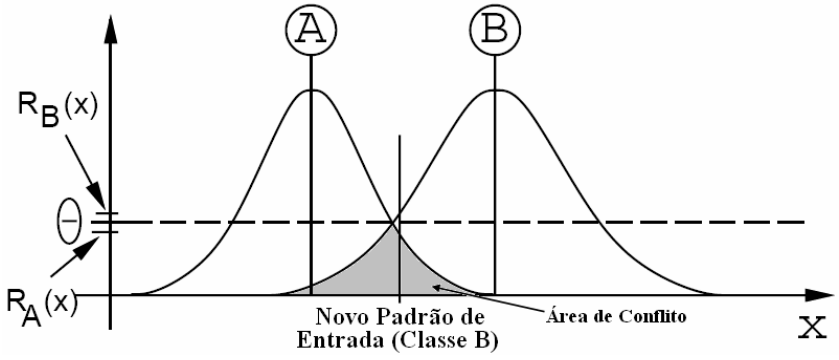


Figura 16. Região de conflito entre duas gaussianas.

O algoritmo DDA é um algoritmo de treinamento construtivo. No início do processo de treinamento a camada escondida não possui nenhum neurônio, e a partir do momento que a fase treinamento se inicia, novos neurônios vão sendo adicionados dentro dessa camada caso haja necessidade. Ou seja, a quantidade de neurônios que devem fazer parte da camada escondida não é determinada inicialmente, ela será determinada automaticamente durante o treinamento pela análise de dois parâmetros, o θ^+ e o θ^-, que serão analisados a seguir. Nas redes RBF-DDA, os nodos da camada de entrada são totalmente conectados aos neurônios da camada intermediária. Dessa forma, se um novo neurônio for adicionado à camada intermediária, ele deverá ser conectado a todas as entradas.

O algoritmo DDA utiliza dois parâmetros para decidir se um novo neurônio deverá ser introduzido na camada intermediária [2] [3][20]. O primeiro parâmetro é o *threshold* positivo, ou θ^+, que possui um valor padrão de 0,4 e que verifica se para um novo padrão, usado durante o treinamento da rede, existe algum protótipo da mesma classe com ativação acima do θ^+. Caso exista, nenhum novo protótipo será adicionado à rede. O segundo parâmetro é *threshold* negativo, ou θ^-, que possui um valor padrão de 0,1 e é utilizado para solucionar problemas de conflito de padrões que possam vir a existir durante o treinamento [2][20].

Na camada de saída para uma rede RBF-DDA existe uma quantidade n de neurônios, de forma que essa quantidade é igual ao número de classes utilizadas pelos padrões de treinamento. Ou seja, se durante o treinamento tivemos padrões de 4 classes diferentes, na camada de saída teremos 4 unidades presentes e cada uma delas representa a saída de uma dessas classes. Para realizar essa classificação, os neurônios da camada saída competem através do método conhecido como *winner-takes-all*, onde o neurônio que contiver o maior valor de ativação será usado para realizar a classificação.

O algoritmo de treinamento do DDA pode possuir várias épocas. Uma época é uma passagem completa por todos os exemplos da base de treinamento. O algoritmo DDA, para uma época, pode ser descrito da seguinte forma: inicialmente ele zera os pesos de todos os protótipos da rede para não acumularem valores duplicados sobre os padrões de treinamento. Após isso ele compara, para cada padrão de treinamento, o valor de ativação com o parâmetro θ^+, onde esse valor de ativação é a intersecção do centro do novo protótipo com a gaussiana de mesma classe. Se o valor de ativação for maior ou igual ao θ^+, o peso (altura) da gaussiana é incrementado em uma unidade. Caso a ativação seja menor do que o θ^+, um novo protótipo será adicionado à rede com centro igual vetor de entrada e o peso igual a 1. O último passo é fazer o ajuste das larguras das gaussianas, por conta das regiões de conflito. Para isso, as gaussianas são ajustadas através da intersecção do θ^- com o centro das gaussianas de classes conflitantes.

A Figura 17 é um exemplo apresentado por Berthold [2], onde este ilustra os passos do funcionamento do DDA:

(a) Um novo padrão classe *A* é apresentado, então uma nova gaussiana é adicionada.
(b) Um novo padrão da classe *B* é apresentado, então uma nova gaussiana é adicionada e suas larguras são ajustadas através da intersecção do θ^- com o centro da gaussiana da classe conflitante.
(c) Um novo padrão da classe *B* é apresentado, então a amplitude da gaussiana *B* é incrementada em uma unidade, pois valor de ativação ficou acima do θ^+; e a largura da gaussiana A é ajusta através da intersecção dela com θ^- e o centro do padrão da classe *B* que foi apresentado como entrada.
(d) Um novo protótipo da classe A é apresentado, então será introduzida uma nova gaussiana, pois o valor de ativação ficou abaixo do θ^+, e a largura dessa nova gaussiana será ajustada de acordo com a intersecção dela o θ^- e o centro da gaussiana de classe conflitante.

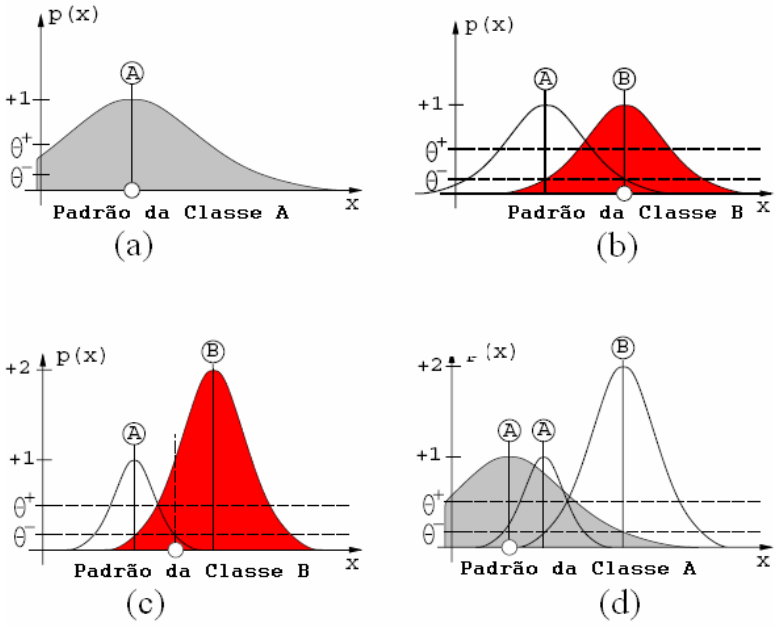


Figura 17. Treinamento do DDA.

2.3.2.2 Variações do algoritmo de treinamento DDA

Desde o surgimento das redes neurais do tipo RBF-DDA, argumenta-se que os parâmetros θ^+ e θ^- têm pouca influência no desempenho das redes RBF. Originalmente, recomendava-se que os parâmetros fossem mantidos com seus valores padrões, ou seja, θ^+=0,4 e θ^-=0,1 [2]. Em contraste com a recomendação original, pesquisas demonstraram que, em alguns casos, a variação do parâmetro θ^- apresenta um resultado considerável no desempenho de classificação do algoritmo [17][20].

O método de seleção do θ^- propõe a utilização de valores menores do que o valor padrão de 0,1. Para selecionar um valor ideal para o θ^-, é usado um método de seleção que tem o objetivo de escolher um valor ótimo para esse parâmetro, ou seja, um valor que proporcionasse a maior capacidade de generalização possível para um determinado problema e que não causasse *overfiting* na rede, devido ao aumento do número de neurônios na camada escondida. O *overfiting* ocorre quando a rede perde sua capacidade de generalização e passa a representar internamente cada padrão como uma classe, devido ao alto número de neurônios armazenados.

Para evitar o *overfiting*, é feita uma divisão do conjunto de treinamento em duas partes, sendo uma parte de teste e outra de validação. O conjunto de treinamento será usado para ajuste dos pesos, e o conjunto de validação será utilizado para estimar a capacidade de generalização da rede durante o processo de treinamento. Nesse caso o treinamento deve ser interrompido quando o erro do conjunto de validação começar a crescer.

Um outro método que propõe modificações para o RBF-DDA é o RBF-DDA-T. Segundo Paetz [21], durante o treinamento das redes RBF-DDA muitos neurônios supérfluos são inseridos na camada intermediária. Essa inserção ocorre por conta de dados com ruído, ou seja, dados nulos e dados com parâmetros preenchidos inadequadamente. O método RBF-DDA-T propõe a utilização de neurônios temporários [21], que são podados em tempo de execução de forma a reduzir o número de neurônios armazenados na camada intermediária. Com este método houve uma redução média de 57,70% do número de neurônios armazenados na camada escondida.

O RBF-DDA-T propõe modificações ao algoritmo de treinamento DDA de forma que, sempre que um novo neurônio é adicionado à camada intermediária ele é imediatamente

marcado como *temporário*. Durante o treinamento os neurônios vão sendo incrementados, conforme foi visto na seção 2.3.2.1, e à medida que isso vai acontecendo, eles vão sendo marcados como *permanentes*. Depois de cada época de treinamento, todos os neurônios não representativos, ou seja, todos os neurônios supérfluos, são excluídos da camada intermediária e não serão usados durante a próxima época do treinamento.

Outra otimização para o método RBF-DDA é o método proposto por Oliveira em [18]. Esse novo método é chamado de RBF-DDA-SP. Assim como o algoritmo DDA, o RBF-DDA-SP é um método para treinamento construtivo de redes neurais do tipo RBF.

O RBF-DDA-SP propõe a junção da técnica de seleção do parâmetro θ^- com a técnica de poda proposta pelo RBF-DDA-T. Como já foi visto, o algoritmo de seleção do θ^- aumenta a capacidade de generalização da rede, porém aumenta o número de neurônios na camada escondida em relação ao algoritmo DDA original. Por conta dessa desvantagem, foi proposta a junção do método de seleção do θ^- com o RBF-DDA-T. Essa junção tem o objetivo de aumentar a capacidade de generalização sem gerar redes muito grandes.

Durante a integração dos dois algoritmos, observou-se que ocorria uma alta taxa de poda dos neurônios quando se usavam valores pequenos para o θ^-. Assim foram colocados muito menos neurônios na camada escondida, e conseqüentemente ocorreu uma queda no desempenho do classificador.

Por conta do grande número de neurônios excluídos da rede, o algoritmo RBF-DDA-SP, ao contrario do que propunha o algoritmo original do RBF-DDA-T, faz a poda dos neurônios somente após a última época de treinamento do algoritmo DDA. Dessa forma, o algoritmo primeiro constrói a rede usando o algoritmo DDA original, e na fase subseqüente, após construída a rede, faz a poda de uma porcentagem P dos neurônios que tiveram a função de ativação (amplitude da gaussiana) igual a 1. A escolha dos neurônios que serão podados é feito de forma randômica. O RBF-DDA-SP possui dois parâmetros que influenciam diretamente no resultado final da rede, o θ^- e a porcentagem P de neurônios que serão podados. O melhor ajuste para esses dois parâmetros é feito via validação cruzada dos dados de entrada.

A validação cruzada [6][23] é uma técnica que melhora a divisão da base de dados feita pelo *holdout*. Essa técnica funciona de forma que o conjunto de treinamento é dividido em K partes com tamanhos aproximadamente iguais. Após essa etapa, o treinamento é feito de forma que, a cada rodada de execução do algoritmo, um subconjunto de K é usado para teste e os outros $K-1$ subconjuntos são usados juntos para treinamento. A cada rodada do treinamento é calculado um erro de classificação, e, no final, será calculada a média do erro de todas as K tentativas.

Os resultados obtidos pelo algoritmo RBF-DDA-SP para bases de reconhecimento de imagens são utilizados no Capítulo 5 para efeito de comparação de desempenho entre os algoritmos baseados no NN, implementados durante este trabalho.

Capítulo 3

Algoritmo NN

O algoritmo *Nearest Neighbor* [7][26][31] foi proposto por Cover e Hart em 1966. Essa técnica é um método de estimação de densidade bastante simples conceitualmente e de fácil implementação. Durante o Capítulo de experimentos são analisados os resultados obtidos para a classificação de imagens usando o algoritmo NN e algumas variantes desse algoritmo que foram implementadas durante o desenvolvimento do trabalho, como o *K-Nearest Neighbor*, o *Pairwise Opposite Class-Nearest Neighbor* (POC-NN) e outras variantes propostas neste trabalho.

Para demonstrar a maneira como os algoritmos apresentados neste capítulo funcionam para a classificação de um padrão desconhecido, são usadas adaptações de exemplos dados por Raicharoen e colaboradores em [27]. Nesses exemplos, é dado um padrão desconhecido x_d e um conjunto de padrões de treinamento, com suas classes previamente conhecidas. Dessa maneira, os padrões da classe 1, são simbolizados por "+", e os padrões da classe 2, são simbolizados por "*".

3.1 Conceito Sobre NN

A classificação de padrões usando o algoritmo *Nearest Neighbor* é dividida em duas fases: a primeira, de treinamento, e a segunda, de teste. Durante a primeira fase de execução do NN, é usada uma base de treinamento obtida através da técnica de *holdout*. Após o término da etapa de treinamento, inicia-se a fase de teste, na qual se usa a base de dados de teste, também obtida pela técnica de *holdout.*

3.1.1 Fase de Treinamento e Fase de Teste

A fase de treinamento do algoritmo NN é bastante simples e consiste em, dada uma base de treinamento $TR^n=\{X_1, \ldots, X_n\}$ onde n é o numero de padrões armazenados em TR, armazenar na memória todos os padrões de TR. Um padrão X qualquer, que pertença à base de treinamento ou à base de teste, possui duas características: um vetor de valores e uma classe. A primeira é um vetor de valores que armazena todas as suas características e a segunda é uma variável que armazena a qual classe o padrão pertence.

Durante a fase de teste do algoritmo é que se avalia o desempenho do classificador para a classificação de novos padrões. Para isso, durante a fase de teste, dada uma base de

teste $TE^n=\{X_1, \ldots, X_n\}$ onde a classe de cada padrão $X_n \in TE$ é conhecida e n é o número de padrões contidos em TE, deve-se calcular a taxa de erro do classificador para o dado conjunto de teste através da Equação 3.1. Para que se calcule a taxa de erro do conjunto de teste, o algoritmo deve encontrar para cada padrão $X_n \in TE$ um padrão $X' \in TR$, de forma que X', dentre todos os padrões de treinamento de TR, seja o que possui a menor distância Euclidiana em relação a X_n. Então X_n é classificado pelo algoritmo como sendo da mesma classe do padrão X'. Como a classe de X_n era previamente conhecida, durante a fase de teste deve-se avaliar se o algoritmo errou na classificação do padrão ou não, e como já foi visto, ao final de todos os padrões de teste, calcular a taxa de erro através da Equação 3.1.

$$\text{Tx. de Erro} = 100 \times \frac{\text{N.º de Padrões Classificados Incorretamente}}{\text{N.º de Padrões do Conjunto de Teste}} \qquad (3.1)$$

De forma resumida, um pseudocódigo para o algoritmo NN pode ser visto através dos passos descritos no Algoritmo 1. Segundo Duda [7], o algoritmo NN cria uma partição no espaço n-dimensional em *células de Voronói*, onde cada célula é rotulada com a classe do padrão que a originou. Um exemplo da divisão do espaço bidimensional em *células de Voronói*, feita pelo algoritmo NN, é dado por Duda [7] e apresentado na Figura 18, onde os pontos pretos representam os padrões da classe 1 e os pontos vermelhos os padrões da classe 2.

Fase de Treinamento:
- Dada uma base de treinamento $TR^n \mid TR^n=\{P_1, \ldots, P_n\}$
  1. Para cada padrão $P_n \in TR^n$, adicionar o padrão P_n a lista de padrões de treinamento [TREINA]

Fase de Classificação:
- Dado um padrão desconhecido P_d
  1. Achar o padrão $P_x \in [\text{TREINA}]$, tal que P_x seja o padrão com menor distância Euclidiana em relação a P_d
  2. Classificar P_d como sendo da mesma classe que P_x

Algoritmo 1. Pseudocódigo NN para a classificação de um padrão desconhecido.

As *células de Voronói*, também conhecidas como *polígonos de Voronói*, são, no caso do espaço bidimensional, regiões V_i criadas por um dado padrão de treinamento x_i. Dessa maneira um padrão desconhecido x_d pertencerá à região V_i se e somente $\| x_i, x_d \| \leq \| x_i, x_j \|$, para todo j diferente de i. Dessa forma, Carvalho [5] afirma que: "Cada uma destas desigualdades representa o semiplano contendo x_i determinado pela mediatriz do segmento $x_i x_j$. Assim, V_i é um polígono convexo, por ser a interseção de $n-1$ semiplanos", como pode ser visto na Figura 19. Os semiplanos de uma célula de *Voronói* servem como limite de decisão para o algoritmo.

Um exemplo de como funciona a classificação feita pelo NN pode ser visto na Figura 20. Nela podemos ver um padrão desconhecido x_d entre padrões da classe 1 e padrões da classe 2. Usando o algoritmo NN, descrito nessa seção, o padrão x_d é classificado como sendo da classe "+", que é a classe do seu vizinho mais próximo.

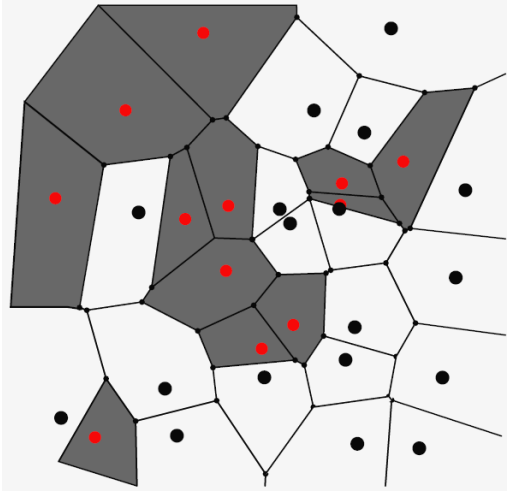


Figura 18. Divisão do espaço em células de Voronói feita pelo NN.

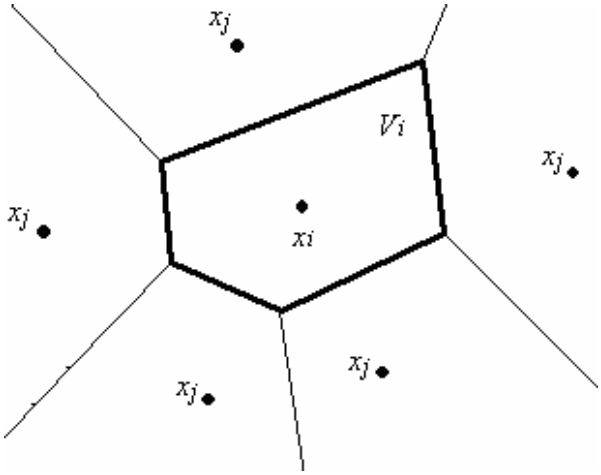


Figura 19. Diagrama de Voronói para pontos do plano.

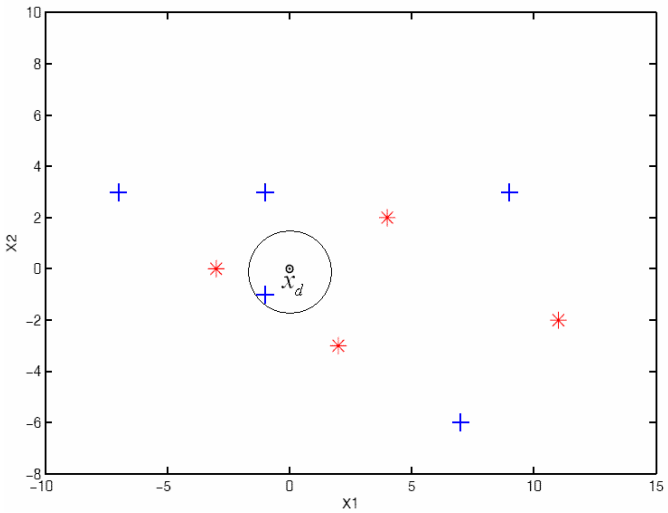


Figura 20. Classificação feita pelo NN para um padrão desconhecido.

3.2 Algoritmos para Aprendizagem de Máquina Baseados no Algoritmo NN

Diversas variações foram propostas para o algoritmo original do NN com o objetivo de otimizar seu desempenho na classificação de padrões, diminuir o esforço computacional e outras características negativas desse algoritmo.

Uma variação bastante simples do algoritmo NN, que surgiu com objetivo de melhorar o desempenho de classificação, é a do algoritmo *K-Nearest Neighbor* [7][31], que propõe modificações durante a fase de teste/classificação do algoritmo NN original. Essa variação do NN é analisada em maiores detalhes na seção 3.2.1. Assim como o KNN, também surgiram outras propostas de melhoria para o algoritmo NN.

O desempenho dos algoritmos de treinamento baseados na regra do vizinho mais próximo pode ser medido pela análise de diversos fatores. Porém, existem três variáveis que influenciam diretamente o desempenho de um algoritmo:

(a) O número de protótipos armazenados durante a fase treinamento;
(b) O tempo computacional necessário para se classificar um padrão desconhecido;
(c) A taxa de erro do conjunto de teste.

Dessa forma, um algoritmo que armazena muitos padrões, como é o caso do NN e do KNN, necessita de um tempo computacional muito grande para classificar um novo padrão. Por outro lado, o fato de armazenar muitos padrões traz alguns benefícios, pois pode ocorrer uma diminuição da taxa de erro do conjunto de teste. Por fim, o desempenho de um algoritmo baseado no NN pode ser medido através de análise custo *versus* benefício dessas 3 variáveis. Assim, se o número de protótipos armazenados aumentar, o tempo computacional aumenta e a taxa de erro do conjunto de teste tende a sofrer uma diminuição. Caso contrário, se o número de protótipos armazenados diminuir, o tempo computacional diminui e a taxa de erro do conjunto de teste tende a sofrer um aumento.

3.2.1 Algoritmo KNN

O algoritmo KNN é uma variação do algoritmo NN. Esse algoritmo propõe uma modificação em relação ao algoritmo original, que se dá durante a fase de teste/classificação, onde o algoritmo faz uso dos K-vizinhos mais próximos, ao invés de usar apenas o vizinho mais próximo, como propunha o algoritmo original.

Para estimar a classe de um novo padrão *X,* o algoritmo KNN calcula os K-vizinhos mais próximos a *X* e classifica-o como sendo da classe que aparece com maior freqüência dentre os seus K-vizinhos. Durante a fase de classificação do KNN, algumas vezes ocorre um problema, onde, dado um padrão de teste *X*, os seus K-vizinhos mais próximos são de uma mesma classe e o algoritmo não consegue decidir com qual classes dos K-vizinhos ele deve comparar o padrão *X*. Para resolver essa situação, o padrão que teve o problema citado anteriormente será rodado de forma recursiva pelo algoritmo, o qual agora usará apenas (K-1) vizinhos para o cálculo, até que uma das classes dos K-vizinhos apareça com maior freqüência em relação às demais.

Apesar de melhorar o desempenho de classificação em relação ao algoritmo original em alguns problemas, o KNN mantém as mesmas deficiências encontradas no NN, pois continua a armazenar todos os padrões de treinamento na memória, como também ainda exige um grande esforço computacional.

Segundo Webb [31]: “O KNN é um método simples de estimação de densidade”. O KNN recebe essa denominação pelo ao fato dele estimar a densidade local de padrões de

treinamento na vizinhança de um padrão desconhecido durante a classificação. O KNN determina um volume *V* que contém os K-vizinhos mais próximos centrados em um padrão *X*, o qual se deseja classificar. Por exemplo, se X_k é o *K*-ésimo vizinho de *X*, então *V* será uma esfera centrada em *X* e com raio igual à distância Euclidiana entre *X* e X_k, ou seja, $\|X- X_k\|$. O volume *V* da esfera, de raio *r* em *n*-dimensões, é dado pela Equação 3.2 [31], onde $\Gamma(x)$ é a função gama [7] dada pela Equação 3.3.

$$V = \frac{2r^n \pi^{n/2}}{n\Gamma(n/2)} \tag{3.2}$$

$$\Gamma(n+1) = \int_0^\infty x^n e^{-x} dx \tag{3.3}$$

O KNN possui um parâmetro chamado *K*, que indica o número de vizinhos que serão usados pelo algoritmo durante a fase de teste. O parâmetro *K* faz com que algoritmo consiga uma classificação mais refinada, porém o valor ótimo de *K* varia de um problema para o outro, o que faz com que, para cada base de dados, sejam testados vários valores diferentes de forma a descobrir qual o melhor valor de *K* para determinado problema.

Um exemplo simples de como funciona a classificação feita pelo algoritmo KNN pode ser visto na Figura 21, onde se pode ver o padrão desconhecido x_d entre padrões da classe 1, e os padrões da classe 2. A tarefa do KNN é classificar o padrão x_d como sendo pertencente a uma das classes exibidas no exemplo. Dependendo do número de K-vizinhos, x_d será classificado da seguinte forma: se *K*=1, x_d será classificado como "+"; se *K*=3, x_d será classificado como "+"; se *K*=5, x_d será classificado como "*".

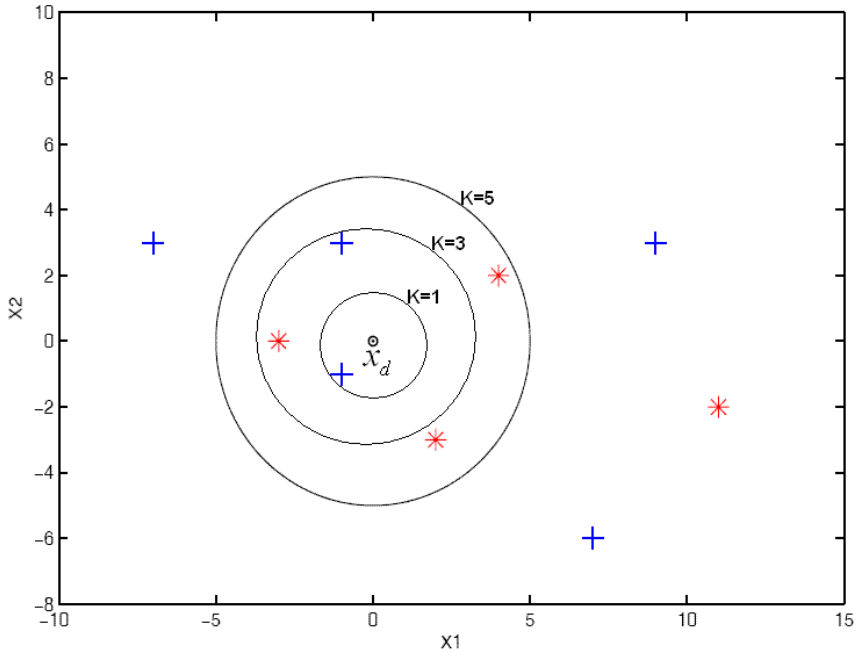


Figura 21. Classificação feita pelo KNN para um padrão desconhecido x_d.

Capítulo 4

Algoritmo POC-NN

No Capítulo 2, foram analisadas as vantagens e desvantagens do algoritmo *Nearest Neighbor*. Também foi visto o algoritmo KNN, uma variação do algoritmo NN que, apesar de trazer um melhor desempenho de classificação em alguns problemas, apresenta as mesmas deficiências do algoritmo original, principalmente no que diz respeito ao tempo computacional para a classificação de novos padrões.

Durante este capítulo, será analisada uma nova variação do algoritmo *Nearest Neighbor*, que foi criada com o objetivo de diminuir o número de padrões armazenados na memória durante a fase treinamento, e, conseqüentemente, diminuir o esforço computacional para se classificar um novo padrão. Este novo algoritmo é chamado de *Pairwise Opposite Class-Nearest Neighbor* (POC-NN).

4.1 Motivação e Conceitos Sobre POC-NN

Visando diminuir o tempo computacional do algoritmo original, o POC-NN [26][27] propõe duas aproximações, chamadas Seleção de Protótipos (*Prototype Selection)* e Troca de Protótipos (*Prototype Replacement*). A primeira deu origem ao algoritmo S-POC-NN, enquanto que a segunda deu origem ao algoritmo R-POC-NN.

O S-POC-NN e o R-POC-NN realizam um pré-processamento sobre o conjunto de padrões de treinamento, armazenando na memória apenas os padrões que possuem uma maior capacidade de generalização. Assim a quantidade de padrões armazenados em memória diminui drasticamente e, conseqüentemente, o uso dos recursos computacionais.

4.1.1 Fase de Treinamento do POC-NN

O objetivo da fase de treinamento do POC-NN é encontrar um subconjunto de padrões, dentre todos os padrões do conjunto de treinamento, que formará o conjunto de protótipos a serem usados na fase classificação/teste. O subconjunto encontrado define os limites de decisão e a maneira como o algoritmo deve classificar quem está dentro ou fora desse limite. A fase de treinamento do POC-NN é dividida em etapas, que variam de acordo com a aproximação utilizada. São elas [26][27]:

(a) *Finding-POC-NN*;

(b) *Selecting-POC-NN*;

(c) Cálculo do intervalo de aceitação;
(d) *Replacing-POC-NN*.

Durante as subseções relacionadas ao treinamento do POC-NN, são usados exemplos dados por Raicharoen em [26][27]. Esses exemplos possuem duas classes de padrões, onde os padrões da Classe 1 são representados pelo símbolo "+" e os padrões da Classe 2 são representados por "*", assim como também o padrão médio (x_m) é simbolizado por "◊", enquanto que os *protótipos POC-NN*, x_{p1} e x_{p2}, são os padrões que estão circulados.

4.1.1.1 Finding-POC-NN

Dado um conjunto de treinamento S, composto por duas classes de padrões distintas S^1 e S^2, de forma que $S^1 \cap S^2 = \varnothing$ e onde n_1 é a quantidade de padrões de S^1 e n_2 é quantidade padrões de S^2. A função *Finding-POC-NN* pode ser vista no Algoritmo 2:

Função *Finding-POC-NN*(S)
1. Dados que S^1 e S^2 são duas classes de padrões distintas e que n_1 e n_2 são seus respectivos tamanhos
2. **SE** $n_1 \geq n_2$
   ENTÃO
3. x_m = Protótipo médio de S^1
   3.1. x_{p2} será o padrão pertencente a S^2 mais próximo de x_m
   3.2. x_{p1} será o padrão pertencente a S^1 mais próximo de x_{p2}
   SENÃO
4. x_m = Protótipo médio de S^2
   4.1. x_{p1} será o padrão pertencente a S^1 mais próximo de x_m
   4.2. x_{p2} será o padrão pertencente a S^2 mais próximo de x_{p1}
   Fim do SE
5. Retorne o padrão POC-NN (x_{p1}, x_{p2})

Algoritmo 2. Função *Finding-POC-NN*.

A Figura 22 ilustra o funcionamento do algoritmo *Finding-POC-NN*. De forma resumida, o algoritmo *Finding-POC-NN* tem o objetivo de encontrar os *protótipos POC-NN* de duas classes distintas. Um detalhe importante a ser observado é que os *protótipos POC-NN*, x_{p1} e x_{p2}, entre duas classes serão sempre os mesmos, independentemente da ordem dos padrões do conjunto de treinamento.

Após os protótipos do POC-NN terem sido encontrados (Passo 5 do Algoritmo 2), é gerado um hiperplano que será usado como limite de decisão pelo algoritmo. Assim, cada região de decisão será limitada por um hiperplano. O hiperplano encontrado será ortogonalmente colocado na distância média entre os protótipos x_{p1} e x_{p2}. Este hiperplano servirá como limite de decisão para o algoritmo, de forma semelhante aos semiplanos que servem de limite para as células de um diagrama de *Voronói*. Os limites de decisão para cada uma das regiões são definidos pelos correspondentes hiperplanos de separação, que foram gerados pelos *protótipos POC-NN* encontrados em cada uma das regiões do espaço de entrada.

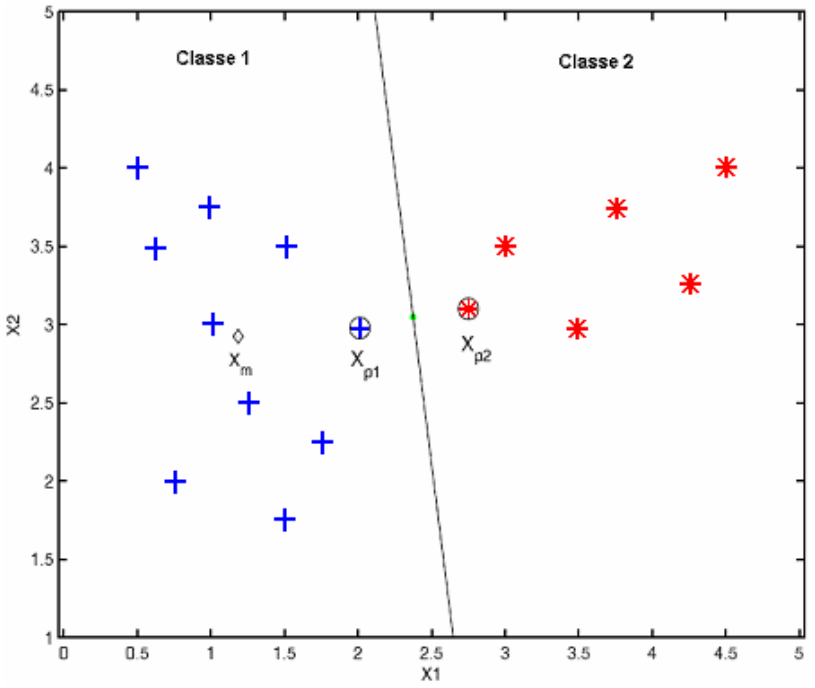


Figura 22. Função Finding-POC-NN.

4.1.1.2 Selecting-POC-NN

O função *Selecting-POC-NN* é um método inovador para seleção de protótipos, de forma que o algoritmo não tenha que armazenar na memória todos os padrões de treinamento. O *Selecting-POC-NN* é um algoritmo recursivo que realiza a classificação de padrões entre duas classes, porém pode ser estendido para classificação de múltiplas classes.

O *Selecting-POC-NN* recebe como entrada um conjunto de treinamento S composto por n padrões de duas classes distintas, o retorno do algoritmo é um conjunto chamado *POC-NN-SET*, que armazena na memória os protótipos de treinamento e os seus respectivos hiperplanos. Vale salientar que, quando o *Selecting-POC-NN* inicia, o *POC-NN-SET* está vazio e que ele segue as regras de um conjunto, onde cada objeto adicionado não deve se repetir. O funcionamento do *Selecting-POC-NN* pode ser visto no Algoritmo 3.

O exemplo da Figura 23 mostra como é feita a separação de padrões de duas classes não linearmente separáveis através do *Selecting-POC-NN*. Inicialmente pode-se ver que os padrões x_{p1} e x_{p2} criaram o hiperplano H_1, Figura 23(a), que gerou alguns padrões classificados incorretamente do lado direito. Todos os padrões do lado direito de H_1 devem ser considerados como sendo um novo conjunto de padrões de treinamento, o novo conjunto será passado de forma recursiva para a função *Selecting-POC-NN*, até que não existam mais padrões classificados erroneamente, como na Figura 23 (b).

O algoritmo POC-NN possui duas abordagens, a primeira é do algoritmo *S-POC-NN* que utiliza o método *Selecting-POC-NN*, descrito nesta seção para a redução do número de padrões armazenados na memória. A segunda abordagem do POC-NN é o algoritmo R-POC-NN que utiliza o método *Replacing-POC-NN*. Este é um método de troca que reduz ainda mais o número de padrões armazenados na memória; será descrito na seção 4.1.1.4.

Função Selecting-POC-NN(S **)**

1. Encontrar os protótipos POC-NN usando (x_{p1} , x_{p2})=**Finding-POC-NN(** S **)**
2. Determinar o *Center Point* usando $c = \frac{x_{p1} + x_{p2}}{2}$
3. Criar um Hiperplano para separação H, de forma que H:{x | $w.x$-b=0}, onde

   $w = \frac{x_{p1} - x_{p2}}{\left\| x_{p1} - x_{p2} \right\|}$ e b=$w.c$
4. Salvar no *POC-NN-SET* os protótipos (x_{p1} , x_{p2}) e o hiperplano H
5. Dividir todos os padrões de S em duas regiões, chamadas $R1$ e $R2$, onde

   $R1 = \{x_i \in S \mid w.x_i - b \geq 0\}$ e $R2 = \{x_i \in S \mid w.x_i - b < 0\}$, $\forall i$ com i variando de 1...n
6. Encontre algum padrão classificado incorretamente em ambas as regiões, $R1$ e $R2$
7. **SE** existe algum padrão classificado incorretamente em $R1$
   ENTÃO
8. Considerar todos os padrões contidos em $R1$ como sendo um novo conjunto de dados e fazer: **Selecting-POC-NN(***R*1)
   FIM DO SE
9. **SE** existe algum padrão classificado incorretamente em $R2$
   ENTÃO
10. Considerar todos os padrões contidos em $R2$ como sendo um novo conjunto de dados e fazer: **Selecting-POC-NN(***R*2)
    FIM DO SE
11. **SE** não existe mais padrões classificados incorretamente
12. **RETORNE** o *POC-NN-SET*

Algoritmo 3. Função *Selecting-POC-NN.*

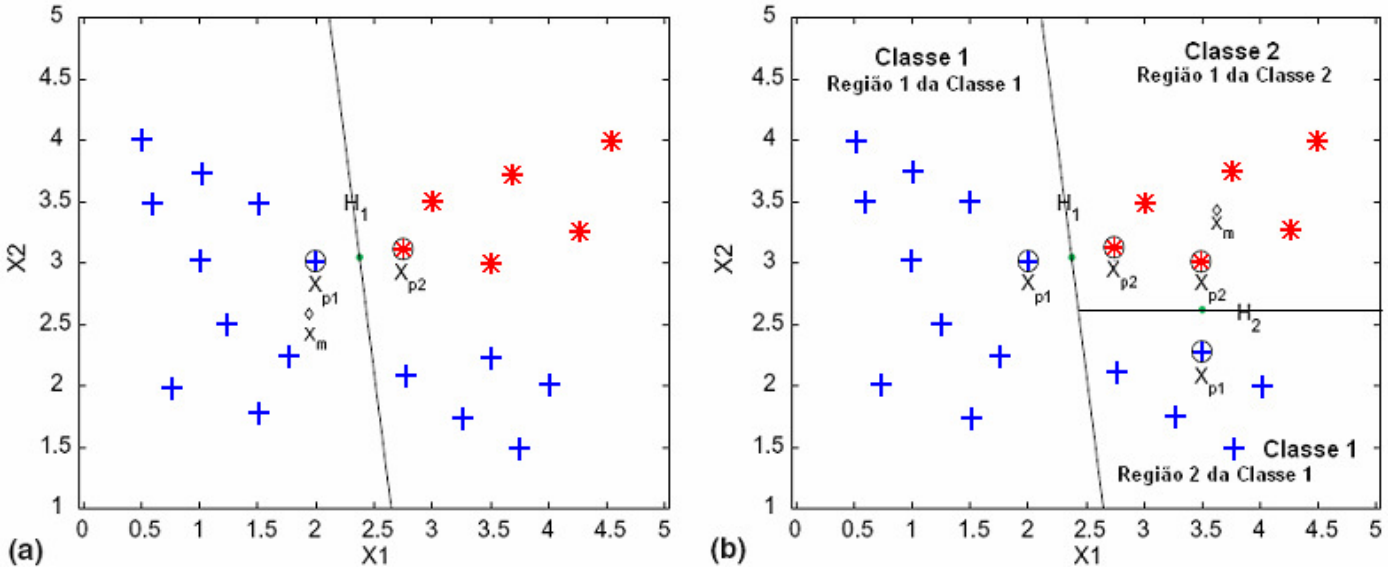


Figura 23. Funcionamento do *Selecting-POC-NN*. (a) Separação feita pelo hiperplano inicial H_1; (b) segunda separação feita pelo hiperplano H_2.

4.1.1.3 Intervalo de aceitação

O POC-NN utiliza um intervalo de aceitação durante a busca por padrões classificados incorretamente (passos 7 e 9 do Algoritmo 3). Este intervalo faz com que haja uma redução da complexidade e da sensibilidade a ruídos no algoritmo POC-NN. Para reduzir a complexidade do algoritmo, um hiperplano de separação pode ser considerado como uma fatia, chamada de intervalo de aceitação, onde esta fatia possui uma largura α. A largura do intervalo de aceitação é proporcional à distância *d* entre os dois *protótipos POC-NN*. Dessa forma, a largura α pode ser calculada como: $\alpha = \alpha\text{-}ratio \times d$, para *α-ratio* variando entre $0 < \alpha\text{-}ratio < 1$.

Com o uso intervalo de aceitação, os padrões que estiverem dentro deste intervalo, do hiperplano de separação, serão considerados como classificados corretamente. Porém os padrões devem ser descartados, pois também são considerados como ruídos ou *outliers*.

Dado um padrão x_i, um hiperplano de separação $H : w.x\text{-}b = 0$ e um intervalo de aceitação com $\alpha > 0$, se $|w.x_i\text{-}b = 0| < \alpha$, então o padrão x_i estará dentro da fatia do intervalo de aceitação e deverá ser descartado, pois será considerado como um ruído/*outlier*.

Na Figura 24, pode ser visto um exemplo onde o *α-ratio* foi definido como 0,2. Após uma iteração do algoritmo *Selecting-POC-NN,* serão encontrados dois protótipos e o espaço será dividido em duas regiões pelo hiperplano *H* . Neste exemplo, 3 padrões classificados incorretamente são ignorados por estarem dentro do intervalo de aceitação. As Figuras 25 e 26 fazem referência ao problema da Figura 24, porém nessas o *α-ratio* foi definido como 0,1. Com isso apenas dois padrões ficaram dentro do intervalo de aceitação e um padrão foi considerado como classificado incorretamente (Figura 25). Logo após, o algoritmo *Selecting-POC-NN* é rodado recursivamente e novos protótipos e um hiperplano são encontrados (Figura 26), com objetivo de classificar o padrão classificado incorretamente. Assim o espaço fica dividido em três regiões.

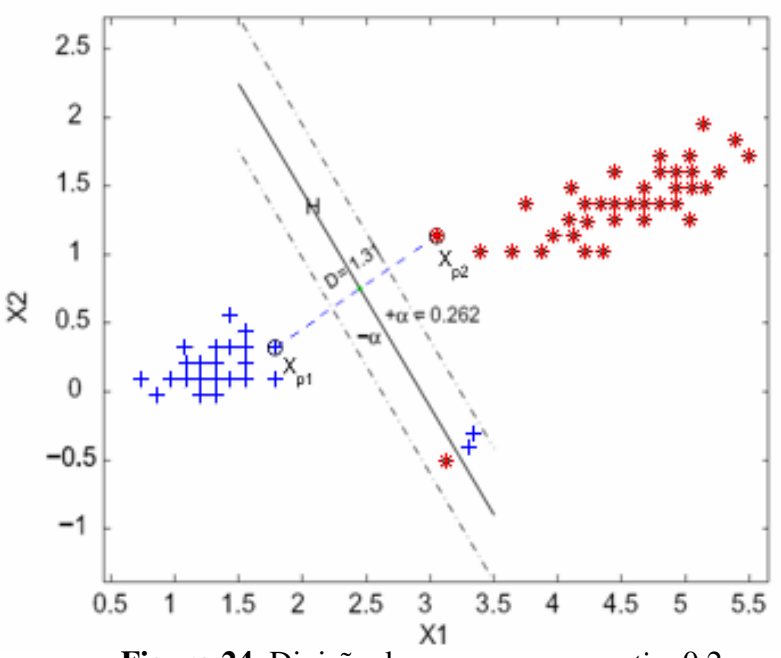


Figura 24. Divisão do espaço com *α-ratio*=0,2.

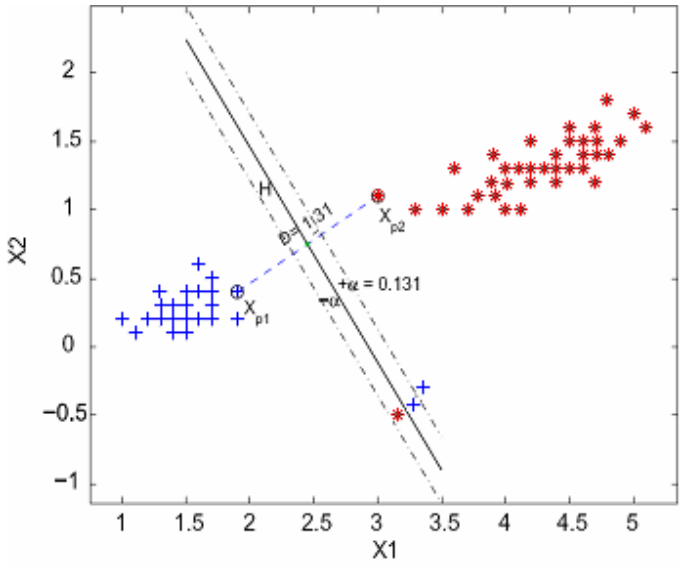


Figura 25. Primeira fase da divisão do espaço com *α-ratio*=0,1.

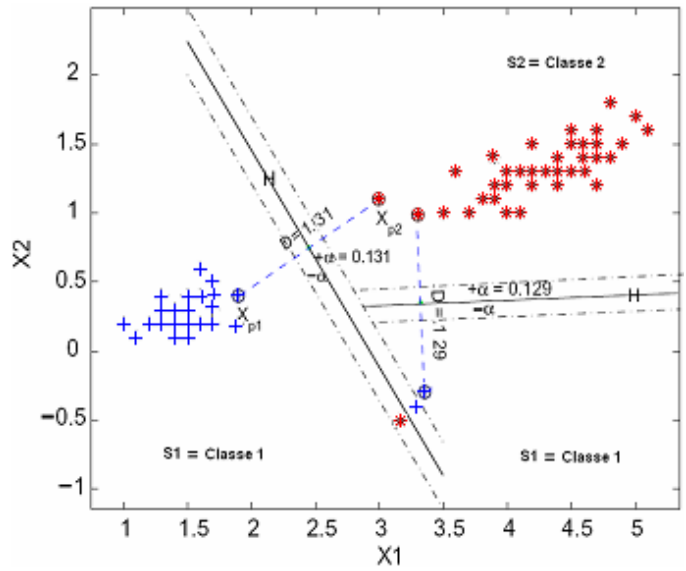


Figura 26. Segunda fase da divisão do espaço com *α-ratio* =0,1.

4.1.1.4 Replacing-POC-NN

Uma segunda abordagem da solução proposta pelo POC-NN é a do R-POC-NN, onde a função usada é a *Replacing-POC-NN* [26][27]. Essa função apresenta, em alguns casos, resultados melhores do que a função *Selecting-POC-NN*, e, geralmente, armazena menos protótipos na memória. A função *Selecting-POC-NN* divide o espaço em diversas regiões, cada região possui protótipos de seleção que a representam. Por sua vez, o algoritmo *Replacing-POC-NN* armazena na memória *protótipos de troca* que não necessariamente coincidem com algum padrão do conjunto de treinamento original. Assim sendo, para cada região encontrada, o *Replacing-POC-NN* realiza uma troca de todos os protótipos de seleção de cada região por um padrão médio, chamado *protótipo de troca*.

Para a obtenção da função *Replacing-POC-NN* é necessária uma modificação na função *Selecting-POC-NN* (Algoritmo 3). O retorno da função também é modificado, ou seja, o conjunto de saída *POC-NN-SET* é substituído por um *MOR-NN-SET*. Os detalhes da implementação estão no algoritmo a seguir (Algoritmo 4), que recebe como entrada um conjunto de treinamento *S*:

Função Replacing-POC-NN(*S* **)**
1. Encontrar os *protótipos POC-NN* usando (x_{p1} , x_{p2})=**Finding-POC-NN**(*S*)
2. Determinar o *Center Point* usando $c = \frac{x_{p1} + x_{p2}}{2}$
3. Criar um Hiperplano para separação *H*, de forma que $H:\{x \mid w.x\text{-}b=0\}$, onde
   $w = \frac{x_{p1} - x_{p2}}{\left\| x_{p1} - x_{p2} \right\|}$ e $b=w.c$
4. Salvar no *POC-NN-SET* protótipos (x_{p1} , x_{p2}) e o hiperplano *H*
5. Dividir todos os padrões de *S* em duas regiões, chamadas *R1* e *R2*, onde
   $R1 = \{x_i \in S \mid w.x_i - b \geq 0\}$ e $R2 = \{x_i \in S \mid w.x_i - b < 0\}$, $\forall i$ com i variando de 1...n
6. Encontre algum padrão classificado incorretamente em ambas as regiões, *R1* e *R2*
7. **SE** existe algum padrão classificado incorretamente em *R1*
   ENTÃO
8. Considerar todos os padrões contidos em *R1* como sendo um novo conjunto de dados e fazer: **Replacing-POC-NN**(*R1*)
   SENÃO
9. Salve x_{mor}, que é o padrão médio de *R1* no *MOR-NN-SET*
   FIM DO SE
10. **SE** existe algum padrão classificado incorretamente em *R2*
    ENTÃO
11. Considerar todos os padrões contidos em *R2* como sendo um novo conjunto de dados e fazer: **Replacing-POC-NN**(*R2*)
    SENÃO
12. Salve x_{mor}, que é o padrão médio de *R2* no *MOR-NN-SET*
    FIM DO SE
13. **SE** não existem mais padrões classificados incorretamente
    ENTÃO
14. Salve x_{mor}, que é o padrão médio de *R1* no *MOR-NN-SET*
    Salve x_{mor}, que é o padrão médio de *R2* no *MOR-NN-SET*
15. **Retorne** o *MOR-NN-SET*

Algoritmo 4. Função *Replacing-POC-NN* [26][27].

Para demonstrar o funcionamento da função *Replacing-POC-NN* será usado, por motivo de comparação, o mesmo exemplo usadona Figura 23. Na Figura 27 pode-se ver que quando o algoritmo converge, o espaço é dividido em três regiões de padrões corretamente classificados. É importante frisar que o número de protótipos armazenados pelo *Replacing-POC-NN* é menor que no *Selecting-POC-NN*. Este fato pode ser observado se compararmos o exemplo da Figura 23 com o da Figura 27, o primeiro armazena quatro protótipos no POC-NN-SET, enquanto que o segundo armazena apenas três protótipos no *MOR-NN-SET*.

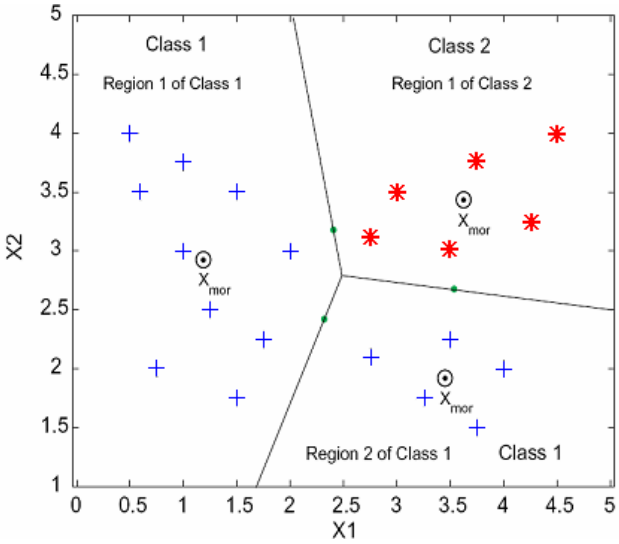


Figura 27. Funcionamento do *Replacing-POC-NN.*

4.1.2 Classificação Para Múltiplas Classes

As funções *Selecting-POC-NN* e *Replacing-POC-NN* do POC-NN foram construídas para fazer classificação binária, ou seja, classificação de problemas que possuem apenas duas classes. Para estender os limites do POC-NN para classificação de múltiplas classes é usada uma técnica chamada *one-against-one (1-n-1)* [26][27][31], que realiza a combinação de diversos classificadores binários para a criação de soluções para problemas de múltiplas classes.

A técnica *one-against-one* faz a combinação de diversos classificadores binários. O número de combinações irá variar de acordo com o número de classes existentes no problema em análise. O número de combinações feitas pelo *one-against-one* é dado pela Equação 4.1.

Na Figura 28, é possível ver o funcionamento do *one-against-one* para um problema de múltiplas classes, onde existem quatro classes distintas. O algoritmo realiza o treinamento entre todas as combinações binárias das classes existentes. Cada combinação entre duas classes irá retornar padrões de treinamento que serão armazenados no POC-NN-SET. É importante frisar que o uso dessa técnica no POC-NN merece alguns cuidados, pois, pode fazer com que um mesmo padrão de treinamento possa ser armazenado mais de uma vez dentro do POC-NN-SET, fato este que deve ser evitado.

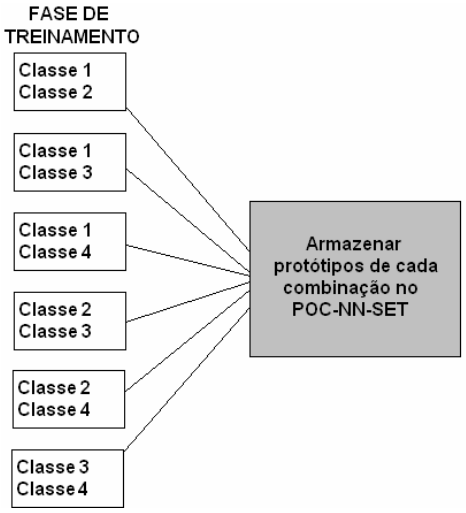


Figura 28. *One-against-one* para problemas com quatro classes distintas.

$$\text{Combinações do one-against-one} = \frac{K(K-1)}{2} \tag{4.1}$$

4.1.3 Fase de Teste POC-NN

A fase de teste do POC-NN é idêntica à fase de teste do NN. Dessa forma, durante a fase teste do algoritmo se avalia o desempenho do classificador para a classificação de novos padrões. Para isso, durante a fase de teste, dada uma base de teste $TE^n=\{X_1, \ldots, X_n\}$, onde a classe de cada padrão $X_n \in TE$ é conhecida e n é o número de padrões contidos em TE, deve-se calcular a taxa de erro do classificador para o dado conjunto de teste através da Equação 3.1. Se estivermos usando o S-POC-NN, para que se calcule a taxa de erro do conjunto de teste, o algoritmo deve encontrar para cada padrão $X_n \in TE$ um padrão $X' \in$ POC-NN-SET de forma que X', dentre todos os protótipos de POC-NN-SET, seja o que possui a menor distância Euclidiana em relação a X_n. Então X_n será classificado pelo algoritmo como sendo da mesma classe do protótipo X'. Como a classe de X_n era previamente conhecida, deve-se avaliar se o algoritmo errou na classificação do padrão ou não.

4.2 Vantagens do POC-NN

O algoritmo POC-NN traz muitas vantagens em relação ao algoritmo original *Nearest Neighbor* e suas variantes, como o *K-Nearest Neighbor.* Cada uma das duas abordagens do POC-NN, S-POC-NN e R-POC-NN, possuem vantagens que são peculiares a cada uma delas, porém existem algumas vantagens que são independentes da abordagem. A primeira é que as duas abordagens reduzem significativamente o número de padrões de treinamento armazenados na memória e a segunda é uma conseqüência da primeira, pois, como o número de padrões diminui, o tempo de classificação de novos padrões também diminui.

Um fato importante a ser frisado é que as taxas de erro obtidas pelo POC-NN para o conjunto de teste são piores do que as taxas obtidas pelos algoritmos NN e KNN. Porém, se for analisado o parâmetro custo *versus* benefício entre os algoritmos, será observado que os resultados obtidos pelo POC-NN são bastante significativos no que diz respeito ao número de padrões armazenados e ao tempo de classificação de novos padrões. A análise custo *versus* benefício, para esse tipo de algoritmo, pode ser feita analisando-se as variáveis descritas na seção 3.2.

O algoritmo de seleção de protótipos ou S-POC-NN possui algumas características que valem a pena serem observadas. A primeira delas é que a função para seleção de protótipos apresenta o mesmo resultado, independente da ordem que os padrões de treinamento são apresentados durante a fase de treinamento. Outro fator interessante e favorável ao S-POC-NN é que o algoritmo sempre converge depois de $n-1$ iterações, onde n é o número padrões POC-NN.

Em relação à abordagem R-POC-NN, a sua principal vantagem é a quantidade de protótipos reduzidos que ele armazena durante a fase de treinamento. Além disso, algumas vezes o R-POC-NN apresenta uma taxa de erro menor que o S-POC-NN. O R-POC-NN, assim como o S-POC-NN, apresenta o mesmo resultado independente da ordem que os padrões de treinamento são apresentados durante a fase de treinamento.

4.3 Algoritmos Propostos

O estudo do *Nearest Neighbor* e de duas de suas variantes, KNN e POC-NN, permitiu a criação de uma ferramenta de treinamento de máquina utilizando esses algoritmos, de forma que fosse possível gerar uma análise dos mesmos. Através desta análise foi possível propor melhorias aos algoritmos existentes até então. Dessa forma surgiram o S-POC-KNN e o S-POC-KNN com núcleo RBF.

A metodologia usada para a criação dos novos algoritmos foi a de combinar duas técnicas distintas, de modo que fosse possível resolver um dado problema usando as vantagens das técnicas em questão.

4.3.1 S-POC-KNN

O algoritmo S-POC-KNN propõe uma modificação simples para o algoritmo S-POC-NN e que traz bons resultados. A fase de teste das duas abordagens do POC-NN (4.1.3), S-POC-NN e R-POC-NN, é idêntica à fase teste do algoritmo *Nearest Neighbor* (3.1.1). Com isso cada padrão *X*, seja ele do conjunto de teste ou um padrão desconhecido, é classificado pelo algoritmo como sendo da mesma classe de um protótipo pertencente ao *POC-NN-SET* que possua a menor distância Euclidiana em relação a *X*.

Para a criação do S-POC-KNN uniram-se as vantagens do algoritmo KNN com as do S-POC-NN. Dessa maneira foi alcançado o objetivo de melhorar o desempenho do algoritmo S-POC-NN original. O S-POC-KNN utiliza a abordagem seleção de protótipos do POC-NN. Assim, o S-POC-KNN armazena em memória a mesma quantidade de padrões que o S-POC-NN, porém a sua fase teste foi modificada de forma a assimilar os benefícios advindos da fase de teste do KNN (3.2.1). Por conseguinte a fase de teste passa a não levar em consideração somente o vizinho mais próximo, mas também os K-vizinhos mais próximos, da mesma forma que ocorre no algoritmo KNN. Os resultados obtidos com o S-POC-KNN podem ser observados no Capítulo 5.

4.3.2 S-POC-KNN-RBF

Outra variação criada para o S-POC-NN que apresentou bons resultados é o algoritmo S-POC-NN com núcleo RBF (S-POC-NN-RBF). Este algoritmo une as vantagens do algoritmo S-POC-NN com as do núcleo RBF. Tanto o algoritmo POC-NN original quanto o *Nearest Neighbor*, utilizam a distância Euclidiana como maneira de medir a distância entre padrões n-dimensionais. A proposta do algoritmo S-POC-NN-RBF é substituir a maneira como o algoritmo calcula a distância entre dois padrões no espaço n-dimensional.

Uma série de trabalhos propõe a utilização do NN em espaços não euclidianos. Esses trabalhos utilizam diversas maneiras de se calcular a distância entre dois padrões no espaço *n*-dimensional. O NNSRM (*Nearest Neighbor Structural Risk Minimization*) [14] é um método de classificação de padrões que utiliza o NN junto com outras técnicas de classificação e utiliza a distância Euclidiana para calcular a distância entre dois padrões no espaço. Em Karaçali [15] é proposta uma modificação para do método NNSRM através de uma modificação do espaço usando operadores de *kernel* e utilizando-se do núcleo RBF, também chamado de *kernel* RBF, para calcular a distância entre padrões. Após essa modificação, observou-se que a técnica demonstrou ser mais robusta para condições onde se usa *kernel*.

Nestas condições foi observado um baixo custo computacional em relação à técnica convencional.

Diante dos resultados obtidos por essas pesquisas, foi proposto que a distância Euclidiana fosse substituída pelo cálculo da distância através do núcleo RBF, também conhecida como distância gaussiana. A função para o cálculo da distância utilizando núcleo RBF é dada pela Equação 4.2, onde τ é a variância, e representa um novo parâmetro que pode ser configurado pelo usuário para ajustes no desempenho do algoritmo.

$$\text{distancia}(X_{p1}, X_{p2}) = 2 - 2 * \exp\left(-\frac{\left\| X_{p1} - X_{p2} \right\|^2}{2 * (\tau^2)}\right) \tag{4.2}$$

Após a implementação do algoritmo S-POC-NN-RBF, o mesmo foi estendido de forma a realizar a junção das características obtidas até então com as vantagens do KNN, para melhorar o desempenho do S-POC-NN-RBF. Esse algoritmo é chamado de S-POC-KNN-RBF; esse armazena em memória a mesma quantidade de padrões que o S-POC-NN-RBF, porém sua fase teste foi modificada, de forma assimilar os benefícios advindos da fase de teste do KNN (3.2.1). Assim sendo, a fase de teste passa a não levar em consideração somente o vizinho mais próximo, mas leva em consideração os K-vizinhos mais próximos, da mesma forma que ocorre no algoritmo KNN.

Para o algoritmo S-POC-NN-RBF são possíveis a configuração de três parâmetros: (a) o número de K-Vizinhos; (b) o *α-ratio*; (c) o valor de τ. Durante os experimentos realizados com S-POC-KNN-RBF não foi possível medir o grau de influência do parâmetro *α-ratio*, pois aparentemente esse valor apresenta uma grandeza diferente das distâncias obtidas pelo núcleo RBF. Dessa forma, a análise do grau de influência e a grandeza desse parâmetro devem ser feitos em estudo posterior. Ainda por conta desse fato, durante os experimentos realizados com S-POC-NN-RBF o parâmetro *α-ratio* foi mantido com o valor padrão, ou seja, *α-ratio*=0, pois mesmo pequenas variações deste parâmetro causavam uma queda considerável no desempenho final do classificador. Os resultados obtidos com o S-POC-KNN-RBF podem ser observados no Capítulo 5.

Capítulo 5

Experimentos

Durante este trabalho, foi desenvolvida uma ferramenta para treinamento e avaliação de técnicas de aprendizagem de máquina, de forma que fosse possível a realização de uma análise dos algoritmos implementados. Na Figura 29 é possível observar a interface gráfica do software desenvolvido. Nesta interface gráfica é possível ao usuário escolher qual tipo de algoritmo de aprendizagem de máquina ele deseja utilizar, como também a configuração dos parâmetros de cada um dos algoritmos. No Apêndice podem ser vistos os códigos-fonte para as principais classes do aplicativo.

Figura 29. Interface gráfica do software.

5.1 Bases de Dados Para Reconhecimento de Imagens

São usadas algumas bases do *UCI Machine Learning Repository* [30] para se fazer o estudo comparativo de desempenho entre cada algoritmo. Essas bases são públicas e são usadas para validar e avaliar algoritmos de aprendizado de máquina. São usadas cinco bases de dados: (a) *Iris Plant Database*, base de dados com 3 classes de classificação possíveis, onde cada classe representa uma variedade de uma planta chamada Íris; (b) *Letter Recognition Database*, que tem o objetivo de identificar cada *pixel* de um visor preto-e-branco e classificar a saída como sendo uma das 26 letras do alfabeto inglês; (c) *Optical Recognition of Handwritten Digits*, base de dados usada para reconhecimento óptico de dígitos escritos à mão; (d) *Pen-Based Recognition of Handwritten Digits*, base de dados obtida de 44 escritores através de uma mesa digitalizadora (*Pressure Sensitive Tablet*); (e) *Statlog Project Databases*, base de dados usada na classificação de imagens multi-espectrais de satélite; (f) *Wisconsin Breast Cancer Databases*, apesar de não ser uma base de dados para reconhecimento de imagens, ela foi usada durante a implementação dos algoritmos para validar se os resultados obtidos têm uma qualidade de desempenho equivalente ao algoritmo original, a base *Cancer*, como é popularmente conhecida, foi criada pela *University of Wisconsin Hospitals* para reconhecer se um tumor é maligno ou benigno.

Durante os experimentos, visando à validação dos nossos resultados em relação aos dos algoritmos originais, os conjuntos de treinamento e teste foram mantidos da mesma forma que estão dispostos em sua fonte [30]. Na Tabela 1, podem ser vistas as características de cada base dados. Podem ser observados também o número de padrões de treinamento e de teste, assim como o número de características e o número de classes de cada base.

Tabela 1. Propriedade das bases de dados.

Base de Dados	Número de padrões de treinamento	Número de padrões de teste	Número de parâmetros	Número de classes
Cancer	500	199	9	2
Iris	100	50	4	3
Letter	15000	5000	16	26
Optdigits	3823	1797	64	10
Pendigits	7494	3498	16	10
Satimage	4435	2000	36	6

5.2 Resultados obtidos pelo KNN

Nesta seção são analisados os resultados obtidos pelo algoritmo KNN. A Tabela 2 mostra os resultados obtidos pelo KNN, onde o valor do parâmetro K indica a quantidade de vizinhos que será utilizada pelo algoritmo. A quantidade de K-vizinhos foi variada de forma a se encontrar o melhor ajuste dessa variável, para cada base de dados. Os melhores resultados de cada base estão destacados em negrito.

Ao final desses experimentos podemos verificar que não existe um valor ideal para o número de *K* vizinhos. O melhor valor de *K* varia de acordo com o problema em análise. Algumas vezes as variações do erro são pequenas, porém devem ser levadas em consideração quando se vai fazer a comparação desses resultados com os obtidos por outro algoritmo. É de grande importância se observar também que nem sempre o KNN apresenta resultados

melhores do que o NN. Na Tabela 2, por exemplo, as bases *Letter* e *Optdigits* obtiveram a menor taxa de erro quando o K=1, ou seja, para o algoritmo NN tradicional.

Tabela 2. Erros de classificação do conjunto de teste obtidos com o algoritmo KNN para vários valores de *K*.

Base de Dados	Taxa de Erro do Conjunto de Teste para K-Vizinhos			
	K=1	K=3	K=5	K=7
Cancer	2,51%	**1,00%**	1,50%	2,01%
Iris	6,00%	6,00%	**4,00%**	4,00%
Letter	**4,56%**	4,64%	5,08%	4,92%
Optdigits	**2,00%**	2,11%	2,11%	2,28%
Pendigits	2,25%	**2,17%**	2,37%	2,48%
Satimage	10,65%	**9,35%**	9,60%	10,05%

5.3 Resultados Obtidos Pelo S-POC-NN

Esta seção analisa os resultados obtidos pelo S-POC-NN. Na Tabela 3, é possível observar os resultados obtidos através do S-POC-NN, os melhores resultados estão em negrito e foram obtidos através da variação do parâmetro *α-ratio* num intervalo que vai de 0 a 1, como ser visto nos Gráficos de 1 a 6. Uma análise dessa tabela mostra que as taxas de erro obtidas pelo S-POC-NN, geralmente, são maiores do que as taxas de erro obtidas pelo KNN. Se forem levados em consideração os resultados obtidos pelo S-POC-NN (Tabela 3) e pelo KNN (Tabela 2) para a base *Iris*, será visto que os resultados da taxa de erro de classificação tiveram o mesmo desempenho (4,00%), no melhor dos casos. Porém, a quantidade de protótipos armazenados pelo S-POC-NN é muito menor, o que acarreta uma diminuição do uso de recursos computacionais e do tempo necessário para a classificação de padrões desconhecidos. Enquanto que o KNN armazena 100 padrões de treinamento, o que equivale a 100% dos padrões da base treinamento *Iris*, o S-POC-NN, no melhor dos casos, onde o *α-ratio*, descrito na tabela como α_r, é igual 0,51, armazena apenas 7 padrões, o que equivale a apenas 7% dos padrões da base de treinamento *Iris*.

Uma variável que desempenha um papel importante no desempenho do S-POC-NN é a α_r, que determina o tamanho do intervalo de aceitação (4.1.1.3), que será usado pelo algoritmo durante a fase de treinamento. O intervalo de aceitação é usado durante a busca por padrões classificados incorretamente, passos 7 e 9 do Algoritmo 3. Este intervalo faz com que haja uma redução da complexidade e da sensibilidade a ruídos no algoritmo POC-NN.

A Tabela 3 analisa os resultados obtidos pelo S-POC-NN para os valores de *α-ratio*=0 e para alguns valores de *α-ratio* $\neq 0$. Mas, através dessa, não é possível analisar a influência da variação do parâmetro *α-ratio* (α_r) sobre o desempenho do S-POC-NN. Para este fim foram construídos gráficos (Gráficos de 1 a 6) que mostram o impacto da variação do parâmetro alfa-rate (α_r) sobre cada umas das bases.

Se analisarmos a variação do *α-ratio* nos gráficos, podemos ver o impacto causado pela variação deste parâmetro, como também podemos observar alguns fatos interessantes. Por exemplo, vemos que no gráfico da base *Iris* (Gráfico 2) ocorrem regiões de descontinuidade tanto da taxa de erro, quanto no número de padrões armazenados. Esse fato pode ser justificado por conta da base *Iris* possuir poucos padrões para treinamento e teste. Para os gráficos de bases maiores, como por exemplo, os da base *Letter* (Gráfico 3) e os das base *Pendigits* (Gráfico 5) veremos que esse fato não ocorre.

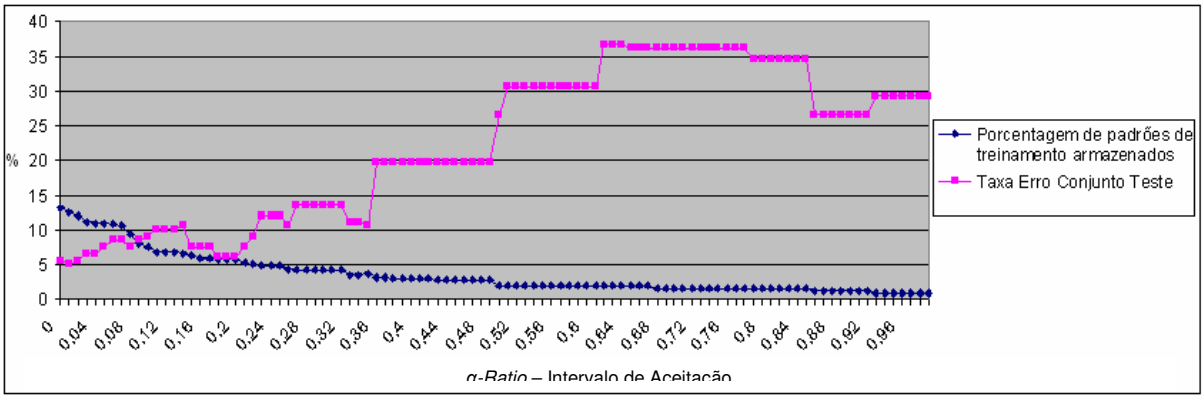


Gráfico 1. Resultado do S-POC-NN para a variação do parâmetro *α-ratio* sobre a base *Cancer*.

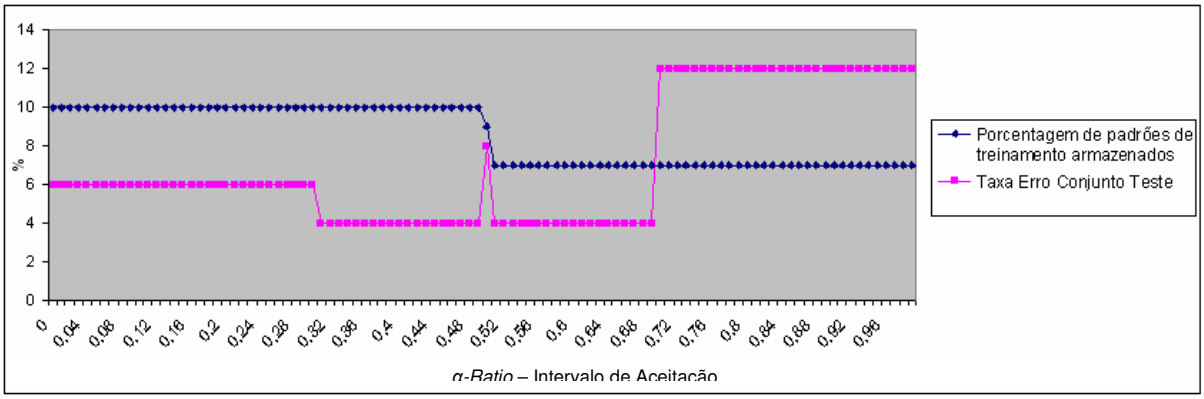


Gráfico 2. Resultado do S-POC-NN para a variação do parâmetro *α-ratio* sobre a base *Iris*.

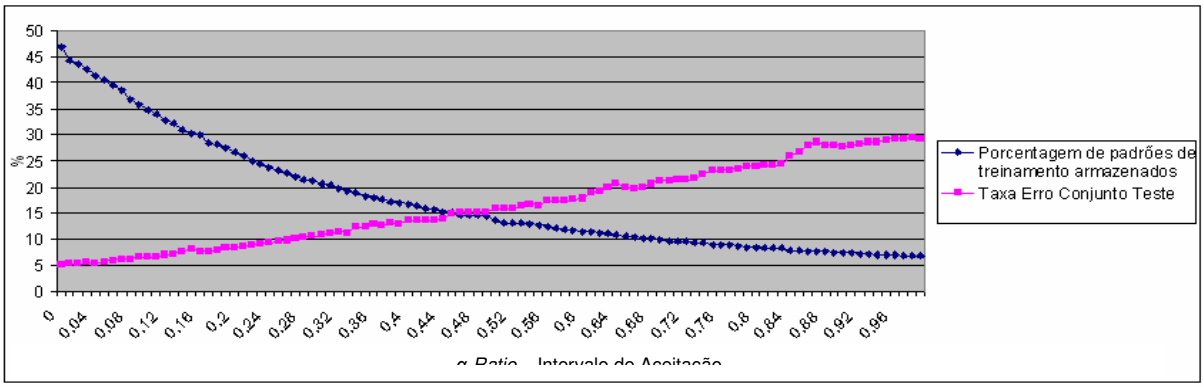


Gráfico 3. Resultado do S-POC-NN para a variação do parâmetro *α-ratio* sobre a base Letter.

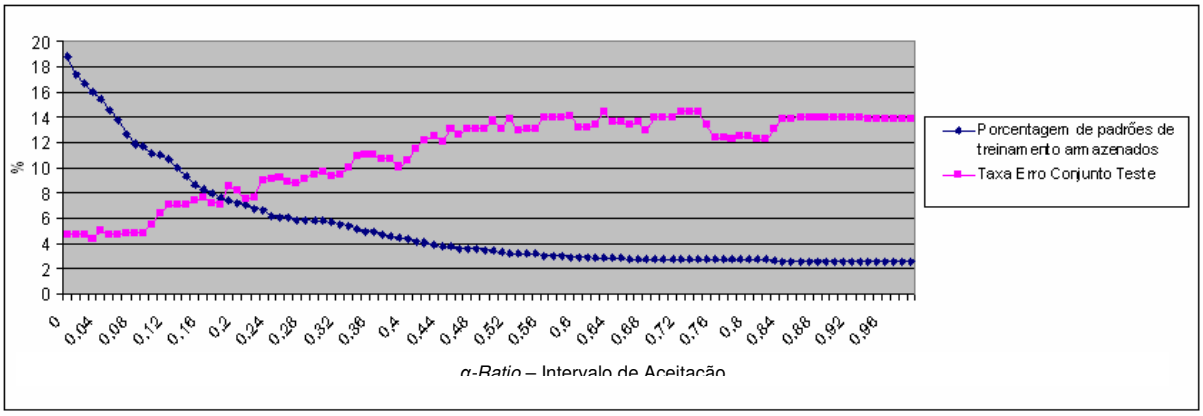


Gráfico 4. Resultado do S-POC-NN para a variação do parâmetro *α-ratio* sobre a base Optdigits.

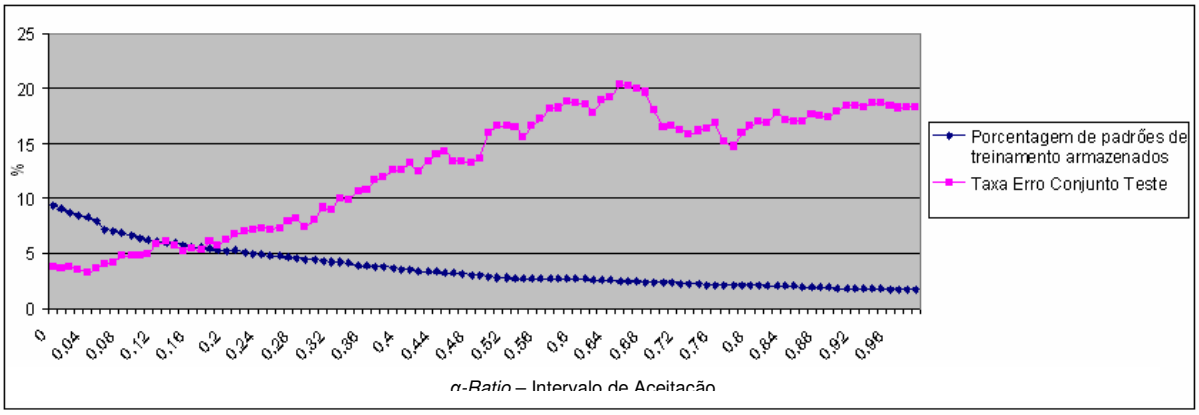


Gráfico 5. Resultado do S-POC-NN para a variação do parâmetro *α-ratio* sobre a base Pendigits.

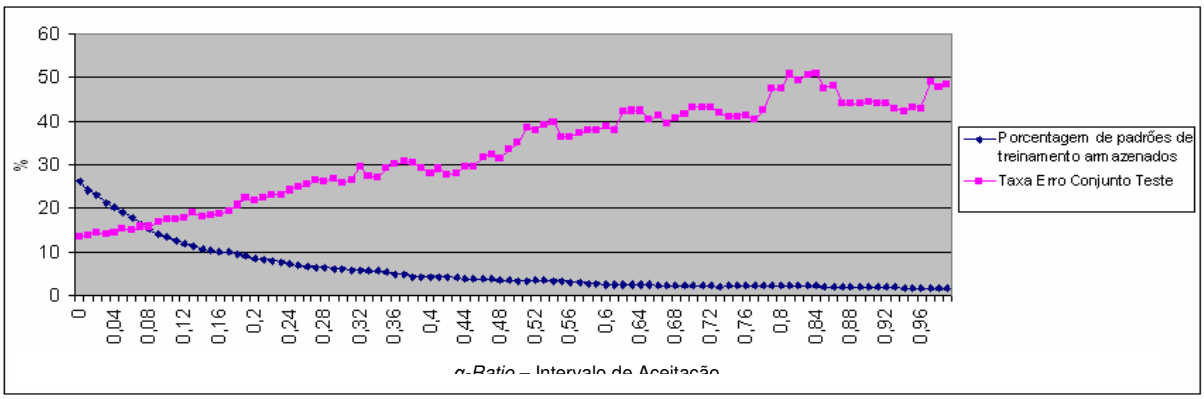


Gráfico 6. Resultado do S-POC-NN para a variação do parâmetro *α-ratio* sobre a base Satimage.

Tabela 3. Erros de classificação do conjunto de teste obtidos com o algoritmo S-POC-NN.

Base de Dados	$\alpha_r = 0$			$\alpha_r \neq 0$			
	Protótipos Treinamento		Taxa de Erro Conj. Teste	α_r	Protótipos Treinamento		Taxa de Erro Conj. Teste
	Total	Arm.			Total	Arm.	
Cancer	500	66	5,53%	0,01	500	63	**5,03%**
Iris	100	10	6,00%	0,51	100	7	**4,00%**
Letter	15000	7018	**5,16%**	0,05	15000	6057	5,62%
Optdigits	3823	722	4,73%	0,03	3823	613	**4,34%**
Pendigits	7494	700	3,86%	0,04	7494	622	**3,32%**
Satimage	4435	1154	**13,25%**	0,03	4435	937	13,80%

5.4 Resultados Obtidos pelo S-POC-KNN

O S-POC-KNN foi proposto com intuito de melhorar o desempenho do S-POC-NN. Para criação desse novo algoritmo uniram-se as vantagens do KNN com as do S-POC-NN. Desta maneira o desempenho do algoritmo S-POC-NN foi melhorado consideravelmente para algumas bases de dados.

A Tabela 4 analisa os resultados obtidos pelo S-POC-KNN para k=1, K=3, k=5 e K=7. Os valor do *alfa-ratio* (α_r) são advindos dos melhores resultados obtidos para cada base pelo S-POC-NN, encontrados na Tabela 3. Os resultados do S-POC-NN estão dispostos na coluna onde K=1 (Tabela 4). Se forem comparados como valores obtidos pelo S-POC-KNN, verifica-se uma diminuição significativa na taxa de erro de classificação para três bases. Na Tabela 4 os valores em negrito representam os melhores resultados obtidos pelo S-POC-KNN para cada base. Para o S-POC-KNN, a quantidade de protótipos armazenados para cada base de dados é a mesma, tendo em vista que a modificação proposta ocorre durante a fase de teste, onde apenas o valor do parâmetro *K* é variado. Ou seja, os valores contidos na coluna *quantidade de protótipos* é independente do valor de *K*.

Tabela 4. Desempenho do S-POC-KNN para K=3, k=5 e K=7.

Base de Dados	α_r	Quantidade de protótipos	Taxa de erro de classificação para conjunto de teste do S-POC-KNN			
			K=1	K=3	K=5	K=7
Cancer	0,01	63	5,03%	3,02%	3,02%	**2,51%**
Iris	0,51	7	**4,00%**	14,00%	68,00%	70,00%
Optdigits	0,03	613	4,34%	**3,23%**	3,84%	3,78%
Pendigits	0,04	622	**3,32%**	3,66%	3,46%	3,80%
Letter	0,00	7018	**5,16%**	5,22%	5,62%	6,10%
Satimage	0,00	1154	13,25%	12,05%	11,35%	**10,95%**

5.5 Resultados obtidos pelo S-POC-KNN-RBF

Uma outra variação para o algoritmo S-POC-NN e para o S-POC-KNN, proposto neste trabalho, é o algoritmo S-POC-KNN-RBF. Esse algoritmo une as vantagens do algoritmo S-POC-NN com as do núcleo RBF e do KNN. Dessa maneira ele consegue melhorar o desempenho em relação ao S-POC-NN original.

Durante os experimentos o parâmetro *α-ratio* foi mantido com o valor padrão, ou seja, *α-ratio*=0. Os resultados obtidos pelo POC-NN-RBF (k=1) podem ser observados nos gráficos de 7 a 10. Os melhores resultados do parâmetro τ obtidos dos gráficos de 7 a 10, para cada base de dados, foram agrupados na Tabela 5 e estendidos pelo algoritmo S-POC-KNN-RBF, para que fossemos usados os K-vizinhos mais próximos. Na Tabela 5 os valores em negrito representam os melhores resultados obtidos para cada base de dados.

Ainda na Tabela 5, a coluna com a quantidade de protótipos armazenados para cada base de dados possui os mesmos valores, independentemente do valor de *K*, ou seja, os valores da coluna *quantidade de protótipos* não depende dos valores de *K*.

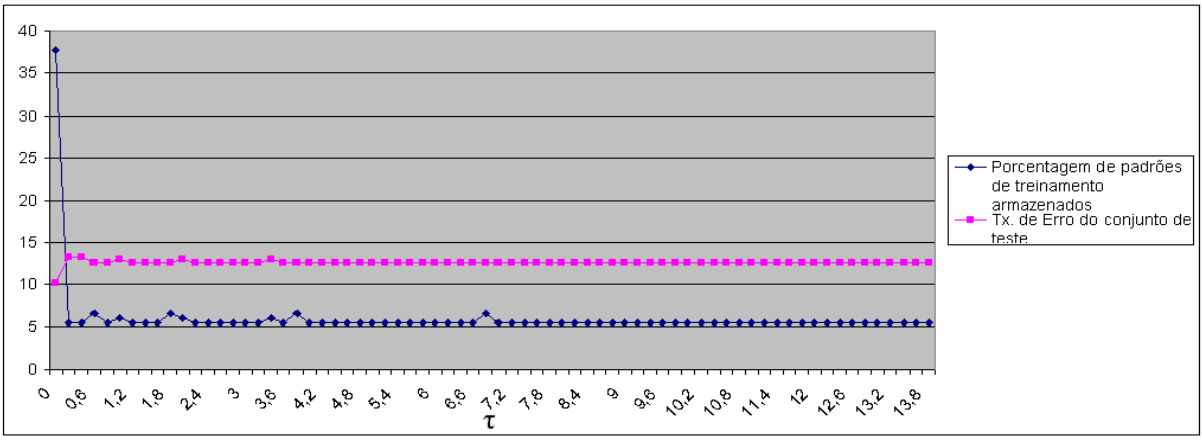


Gráfico 7. Resultado do POC-NN-RBF para a variação do parâmetro τ sobre a base *Cancer*.

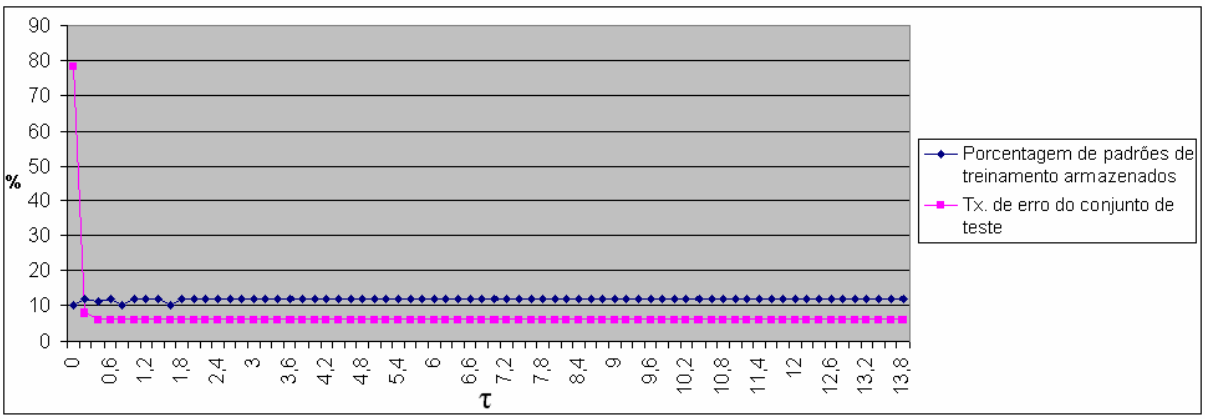


Gráfico 8. Resultado do POC-NN-RBF para a variação do parâmetro τ sobre a base *Iris*.

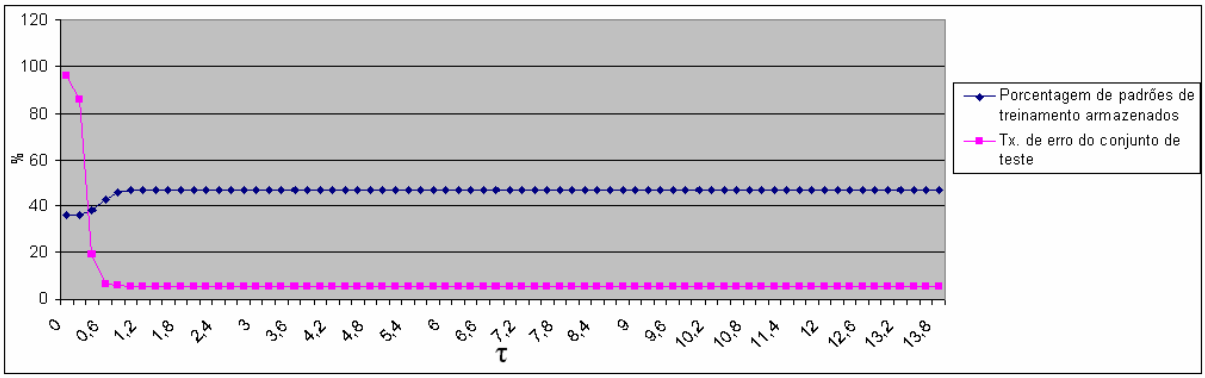


Gráfico 9. Resultado do POC-NN-RBF para a variação do parâmetro τ sobre a base Letter.

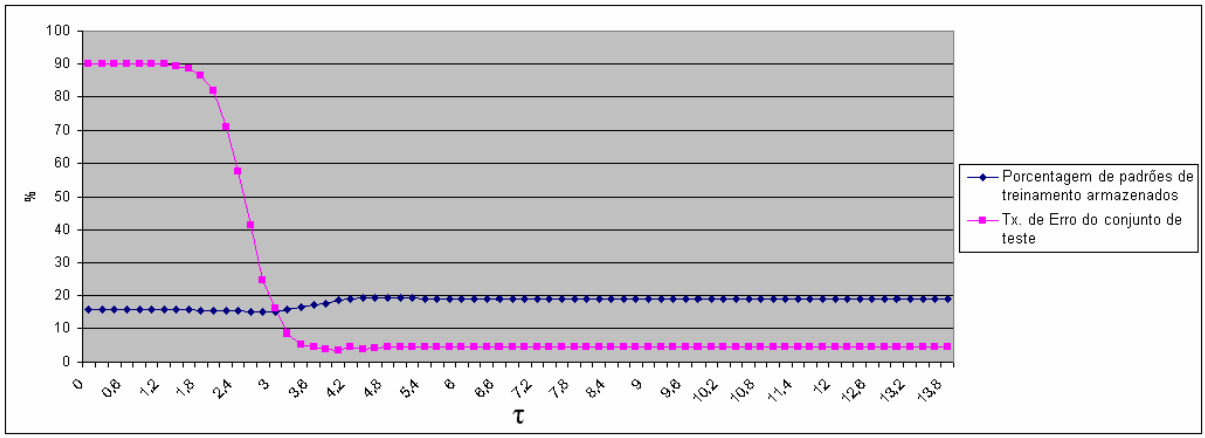


Gráfico 10. Resultado do POC-NN-RBF para a variação do parâmetro τ sobre a base Optdigits.

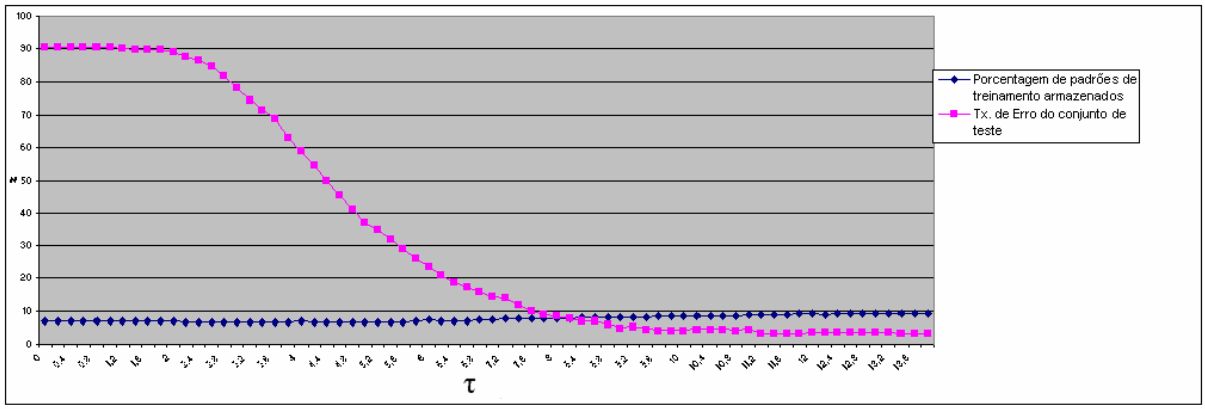


Gráfico 11. Resultado do POC-NN-RBF para a variação do parâmetro τ sobre a base Pendigits.

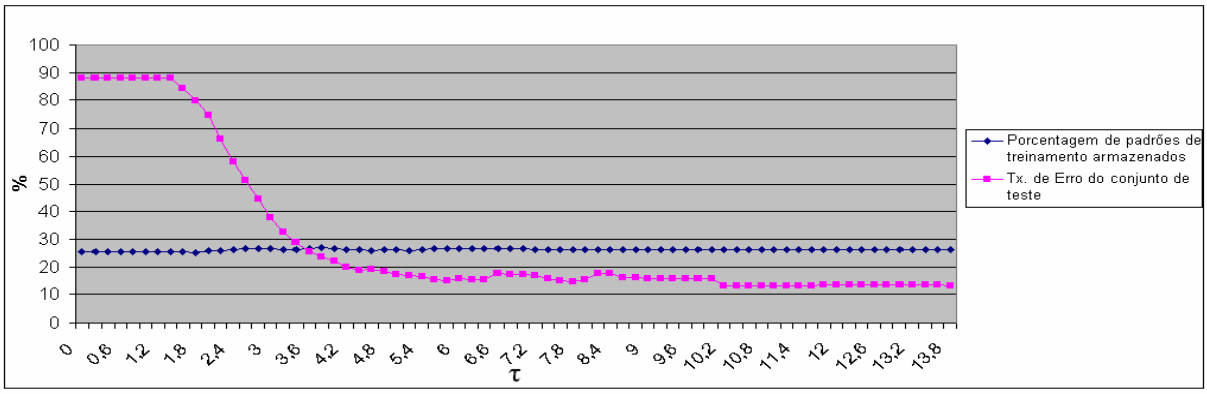


Gráfico 12. Resultado do POC-NN-RBF para a variação do parâmetro τ sobre a base Satimage.

Tabela 5. Desempenho do S-POC-KNN-RBF para K=3, k=5 e K=7.

Base de Dados	τ	α_r	Quantidade de protótipos	Taxa de erro de classificação para conjunto de teste do S-POC-KNN-RBF			
				K=1	K=3	K=5	K=7
Cancer	0,80	0,00	63	5,53%	4,02%	3,02%	**2,51%**
Iris	0,80	0,00	10	**6,00%**	14,00%	34,00%	34,00%
Optdigits	4,00	0,00	707	3,67%	**3,34%**	3,51%	3,67%
Pendigits	13,80	0,00	684	3,37%	**3,34%**	3,49%	3,54%
Letter	1,60	0,00	7013	**5,24%**	5,28%	5,48%	6,20%
Satimage	10,20	0,00	1161	**13,1%**	14,05%	13,55%	15,10%

5.6 Comparação Entre Resultados

Até o momento, já foram analisados os desempenhos individuais de vários algoritmos de treinamento de máquina como o KNN, o S-POC-NN, o S-POC-KNN e o S-POC-KNN-RBF, para o problema de classificação de imagens. Nessa seção, o desempenho dos algoritmos é comparado com os resultados obtidos em [18] por um algoritmo de treinamento de redes neurais, chamado RBF-DDA-SP. Os desempenhos de cada algoritmo também são comparados entre si.

A Tabela 6 analisa o desempenho do S-POC-NN em relação ao algoritmo RBF-DDA-SP. Os valores de α_r da Tabela 6 foram configurados de acordo com os melhores resultados obtidos por cada base na Tabela 3. É importante observar que as taxas de erro obtidas pelo RBF-DDA-SP foram menores do que as do S-POC-NN, porém o S-POC-NN armazena uma quantidade muito menor de padrões em memória.

A Tabela 7 analisa o desempenho do S-POC-KNN em relação ao algoritmo RBF-DDA-SP. Os valores de α_r da Tabela 7 foram configurados de acordo com os melhores resultados obtidos por cada base na Tabela 3. É importante observar que a quantidade de padrões armazenados pelo S-POC-KNN é igual as do S-POC-NN. Porém, muitas vezes, o desempenho é melhorado pela variação do número dos *K* vizinhos, como por exemplo na base

Optdigits e *Satimage*. Os resultados obtidos pelo S-POC-KNN se aproximam dos obtidos pelo RBF-DDA-SP, mas com a vantagem de armazenar muito menos padrões.

Tabela 6. Erros de classificação do conjunto de teste para os algoritmos S-POC-NN e RBF-DDA-SP.

Base de Dados	**S-POC-NN**		**RBF-DDA-SP**		
	α_r	Taxa de Erro	**Taxa de erro**		
			(30%, θ^- sel.)	(40%, θ^- sel.)	(50%, θ^- sel.)
Optdigits	0,03	4,34% [613]	3,13% [2672]	3,30% [2292]	3,57% [1912]
Pendigits	0,04	3,32% [622]	3,04% [4344]	3,17% [3884]	3,29% [3424]
Letter	0,00	5,16% [7018]	6,54% [9358]	7,10% [8191]	8,00% [7023]
Satimage	0,00	13,25% [1154]	9,18% [2934]	9,59% [2546]	10,05% [2157]

Tabela 7. Erros de classificação do conjunto de teste para os algoritmos S-POC-KNN e RBF-DDA-SP.

Base de Dados	**S-POC-KNN**				**RBF-DDA-SP**		
	α_r	Taxa de Erro			Taxa de Erro		
		K=3	K=5	K=7	(30%, θ^- sel.)	(40%, θ^- sel.)	(50%, θ^- sel.)
Optdigits	0,03	3,23% [613]	3,84% [613]	3,78% [613]	3,13% [2672]	3,30% [2292]	3,57% [1912]
Pendigits	0,04	3,66% [622]	3,46% [622]	3,80% [622]	3,04% [4344]	3,17% [3884]	3,29% [3424]
Letter	0,00	5,22% [7018]	5,62% [7018]	6,10% [7018]	6,54% [9358]	7,10% [8191]	8,00% [7023]
Satimage	0,00	12,05% [1154]	11,35% [1154]	10,95% [1154]	9,18% [2934]	9,59% [2546]	10,05% [2157]

Na Tabela 8, é analisado o desempenho do S-POC-KNN-RBF em relação ao algoritmo RBF-DDA-SP. Os valores de τ e K da Tabela 8 foram configurados de acordo com os melhores resultados obtidos por cada base na Tabela 5. Os valores do α_r foram omitidos da Tabela 8, tendo em vista que esse parâmetro foi configurado com o valor padrão, onde α_r=0. Os resultados conseguidos pelo S-POC-KNN-RBF são próximos aos obtidos pelo RBF-DDA-SP, porém armazenam uma quantidade muito menor de padrões e para a base *Letter* obteve melhores resultados.

Tabela 8. Erros de classificação do conjunto de teste para os algoritmos S-POC-KNN-RBF e RBF-DDA-SP.

Base de Dados	**S-POC-KNN-RBF**			**RBF-DDA-SP**		
	τ	**K**	**Taxa Erro**	**Taxa Erro**		
				(30%, θ^- sel.)	(40%, θ^- sel.)	(50%, θ^- sel.)
Optdigits	4	3	3,34% [707]	3,13% [2672]	3,30% [2292]	3,57% [1912]
Pendigits	13,8	3	3,34% [684]	3,04% [4344]	3,17% [3884]	3,29% [3424]
Letter	1,6	1	5,24% [7013]	6,54% [9358]	7,10% [8191]	8,00% [7023]
Satimage	10,2	1	13,1% [1161]	9,18% [2934]	9,59% [2546]	10,05% [2157]

Na Tabela 9, é analisado o desempenho do KNN em relação ao algoritmo RBF-DDA-SP. Os valores do parâmetro K da Tabela 9 foram configurados de acordo com os melhores resultados obtidos por cada base na Tabela 2. Como pode ser visto na Tabela 9, os resultados das taxas de erro do KNN foram melhores do que os resultados do RBF-DDA-SP e dos outros algoritmos baseados na regra do vizinho mais próximo. O inconveniente é que o KNN armazena em memória todos os padrões de treinamento.

Tabela 9. Erros de classificação do conjunto de teste para os algoritmos KNN e RBF-DDA-SP.

Base de Dados	**KNN**		**RBF-DDA-SP**		
	K	**Taxa Erro**	**Taxa Erro**		
			(30%, θ^- sel.)	(40%, θ^- sel.)	(50%, θ^- sel.)
Optdigits	1	2,00% [3823]	3,13% [2672]	3,30% [2292]	3,57% [1912]
Pendigits	3	2,17% [7494]	3,04% [4344]	3,17% [3884]	3,29% [3424]
Letter	1	4,56% [15000]	6,54% [9358]	7,10% [8191]	8,00% [7023]
Satimage	3	9,35% [4435]	9,18% [2934]	9,59% [2546]	10,05% [2157]

A Tabela 10 foi construída com o objetivo de comparar os resultados dos algoritmos propostos neste trabalho, S-POC-KNN e S-POC-KNN-RBF, com os resultados S-POC-NN. Através dessa tabela é possível uma análise dos melhores resultados de cada algoritmo e uma comparação mais aprofundada entre eles.

Tabela 10. Melhores resultados do S-POC-KNN-RBF, S-POC-KNN e S-POC-NN.

Base de Dados	**S-POC-KNN-RBF**			**S-POC-KNN**			**S-POC-NN**	
	τ	**K**	**Taxa de Erro**	α_r	**K**	**Taxa de Erro**	α_r	**Taxa de Erro**
Optdigits	4	3	3,34% [707]	0,03	3	3,23% [613]	0,03	4,34 % [613]
Pendigits	13,8	3	3,34% [684]	0,04	5	3,46% [622]	0,04	3,32 % [622]
Letter	1,6	1	5,24% [7013]	0,00	1	5,16% [7018]	0,00	5,16 % [7018]
Satimage	10,2	1	13,1% [1161]	0,00	7	10,95% [1154]	0,00	13,25 % [1154]

Capítulo 6

Conclusão e Trabalhos Futuros

Existem diversos métodos de treinamento de máquina baseados na regra do vizinho mais próximo. Porém, o algoritmo POC-NN é um algoritmo recente e inovador. O POC-NN, em relação a outros algoritmos baseados na regra do vizinho mais próximo, como KNN, consegue diminuir o tempo computacional e exige menor quantidade de padrões armazenados na memória. A implementação e a comparação dos dois algoritmos tornou possível ver claramente a inovação trazida pelo POC-NN, suas vantagens e também tornou possível que fossem sugeridas melhorias para o algoritmo POC-NN original.

6.1 Contribuições

Uma das inovações deste trabalho, e que vale a pena ser destacada, é a de que o criador do POC-NN testou o algoritmo em bases variadas [26][27], enquanto que neste estudo foi realizada a implementação do POC-NN juntamente com uma análise do seu desempenho em bases maiores e de reconhecimento de imagens. Os resultados advindos da análise de desempenho do POC-NN para bases de reconhecimento de imagens foram comparados com os resultados obtidos nestas bases por outra pesquisa [18], que utiliza o algoritmo RBF-DDA-SP.

Durante a monografia foi realizada uma análise de desempenho dos algoritmos para o reconhecimento de imagens, envolvendo um significativo grau de complexidade. Sugestões para a solução de problemas relacionados ao tema também foram propostas. Dessa forma, surgiram o S-POC-KNN e o S-POC-KNN-RBF, propostos neste trabalho. Ao final, temos uma visão mais profunda sobre os problemas de aprendizado de máquina e classificação/reconhecimento de imagens.

6.2 Sugestões para trabalhos futuros

Como sugestão de continuidade deste trabalho, está a implementação do algoritmo R-POC-NN. Dessa forma será possível analisar profundamente o impacto trazido por essa abordagem do POC-NN. Através da implementação do R-POC-NN será possível realizar sugestões de melhorias, como, por exemplo, a junção do algoritmo original com o KNN e com o núcleo RBF. Outra sugestão muito importante para continuidade deste trabalho é a de estender a análise feita sobre a influência do parâmetro *alfa-ratio* para o algoritmo S-POC-KNN-RBF.

Bibliografia

[1] BARONE, D. et al. **Sociedades Artificiais – A Nova Fronteira da Inteligência nas Máquinas**. 1. ed. São Paulo: Bookman, 2002.

[2] BERTHOLD, M.R.; DIAMOND, J. Boosting the performance of RBF networks with dynamic decay adjustment. **Advances in Neural Information Processing**, Vol. 7, MIT Press, 1995, p. 521-528.

[3] BERTHOLD, M.; DIAMOND, J. Constructive training of probabilistic neural networks. **Neurocomputing**, Vol.19, 1998, p. 167–183.

[4] BRAGA, A.P.; CARVALHO, A.P.L.F.; LUDEMIR, T.B. **Redes Neurais Artificiais: Teoria e Aplicações**. 1. ed. Rio de Janeiro: LTC, 2000.

[5] CARVALHO, P. C. P.; FIGUEIREDO, L. H. Introdução à Geometria Computational. In: 18° COLÓQUIO BRASILEIRO DE MATEMÁTICA. IMPA, Rio de Janeiro, RJ, 1991. Disponível em: <http://w3.impa.br/~lhf/cursos/gc/livro-gc.zip>. Acessado em: 24 dez. 2005.

[6] DELEN, D.; WALKER, G.; KADAM, A. Predicting Breast Cancer Survivability: A Comparison of Three Data Mining Methods. **Artificial Intelligence in Medicine**, Vol. 34, 2005, p. 113-127.

[7] DUDA, R.O.; HART, P.E.; STORK, D.G. **Pattern Classification**. 2. ed. New York: Wiley-Interscience, 2000.

[8] GUPTA, M.M.; JIN, L.; HOMMA, N. **Static and Dynamic Neural Networks From Fundamentals to Advanced Theory**. 1. ed. New Jersey: Wiley-Interscience, 2003.

[9] HART, P.E. The condensed nearest neighbor rule. **IEEE Trans. Inform. Theory**, Vol. 14, Num. 5, 1968, p. 515-516.

[10] HAYKIN, S. **Redes Neurais, Princípios e Prática**. 2. ed. Porto Alegre: Bookman, 2001.

[11] HOLANDA, A. B. **Novo Dicionário Eletrônico Aurélio**. 3. ed. Editora Positivo, 2004. 1 Cd-Rom. Produzido por: Positivo Informática LTDA.

[12] JAIN, A.K.; HONG, L.; PANKANTI, S.; BOLLE R. An Identity Authentication System Using Fingerprints. **Proceedings of the IEEE**, Vol. 85, Num. 9, 1997, p. 1365-1388.

[13] KANDEL, E. R. As Células Nervosas e o Comportamento. In: KANDEL, E.R.; SCHWARTZ, J. H.; JESSEL, T.M. **Princípios da Neurociência**. 4. ed. São Paulo: Manole, 2003.

[14] KARAÇALI, B.; KRIM, H. Fast minimization of structural risk by nearest neighbor rule. **IEEE Transactions on Neural Networks**, Vol. 14, 2003, p. 127-137.

[15] KARAÇALI, B.; RAMANATH, R.; SNYDER, W.E. A Comparative Analysis of Structural Risk Minimization by Support Vector Machines and Nearest Neighbor Rule. **Pattern Recognition Letters**, Vol. 25, 2004, p. 63-71.

[16] OLIVEIRA, A.L.I. **Neural Networks Forecasting and Classification-Based Techniques for Novelty Detection in Time Series**, Tese (Doutorado em Ciência da Computação). Universidade de Pernambuco, Recife, Brasil, 2004.

[17] OLIVEIRA, A.L.I.; MEDEIROS, E.A.; ROCHA, T.A.; BEZERRA, M.E.R.; VERAS, R.C. A Study on the Influence of Parameter q- on Performance of RBF Neural Networks Trained with the Dynamic Decay Adjustment Algorithm. In: INTERNATIONAL CONFERENCE ON HYBRID INTELLIGENT SYSTEMS, 5., 2005, Rio de Janeiro. **Proceedings**... Rio de Janeiro: His, 2005. (Aceito).

[18] OLIVEIRA, A.L.I.; MELO, B.J.M.; MEIRA, S.R.L. Improving constructive training of RBF networks through selective pruning and model selection. **Neurocomputing**, Vol. 64, 2005, p. 537-541.

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

[20] OLIVEIRA, A.L.I.; NETO, F.B.L.; MEIRA, S.R.L. Improving RBF-DDA performance on optical character recognition through parameter selection. In: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 17., 2004, Cambridge. **Proceedings**... Cambridge: Icpr, 2004. p. 625–628.

[21] PAETZ, J. Reducing the number of neurons in radial basis function networks with dynamic decay adjustment. **Neurocomputing**, Vol. 62, 2004, p. 79–91.

[22] PAGUREK, B.; DAWES, N.; BOURASSA, G.; EVANS, G.; SMITHER, P. Letter pattern recognition. In: CONFERENCE ON ARTIFICIAL INTELLIGENCE APPLICATIONS, 6., 1990, Santa Barbara. **Proceedings**… Santa Barbara: Caia, 1990. p. 312-319.

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

[24] PERELMUTER, G.; CARRERA, E.V.C.E.; VELLASCO, M.; PACHECO, M.A. Reconhecimento de Imagens Bidimensionais Utilizando Redes Neurais Artificiais. In: BRAZILIAN SYMPOSIUM ON COMPUTER GRAPHICS AND IMAGE PROCESSING, 8., 1995, São Carlos. **Proceedings**… São Carlos: Sibgrapi, 1995. p. 197-203.

[25] PLAMONDON, R.; SRIHARI. S.N. On-line and off-line handwriting recognition: a comprehensive survey. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. 22, Num. 1, 2000, p. 63-84.

[26] RAICHAROEN, T.; LURSINSAP, C. A Divide-And-Conquer Approach To The Pairwise Opposite Class-Nearest Neighbor (POC-NN) Algorithm. **Pattern Recognition Letters**, Vol. 26, Num. 10, 2005, p. 1554-1567.

[27] RAICHAROEN, T.; LURSINSAP, C. **A Divide-And-Conquer Approach To The Pairwise Opposite Class-Nearest Neighbor (POC-NN) Algorithm For Classification And Regression Problems**, Tese (Doutorado em Matemática). Chulalongkorn University, Bangkok, Thailandia, 2003.

[28] THEODORIDIS, S.; KOUTROUMBAS, K. **Pattern Recognition**. 2. ed. USA: Elsevier, 2003.

[29] TOMEK, I. Two modifications of CNN. **IEEE Transactions Systems Man Cybernetics**, Vol. SMC-6, 1976, p. 769-772.

[30] UCI Machine Learning Repository. Disponível em: <http://www.ics.uci.edu/~mlearn/MLRepository.html>. Acessado em: 2 out. 2005.

[31] WEBB, A. R. **Statistical Pattern Recognition**. 2. ed. Chichester: John Wiley & Sons, 2002.

[32] WIKIPÉDIA – A Enciclopédia Livre. Disponível em: <http://pt.wikipedia.org>. Acessado em: 2 out. 2005.

Apêndice

Códigos Fonte

Classes Básicas

```
/**
 * @author Miguel Bezerra
 *
 */
public class Pattern implements Comparable{
	private double valoresDoPadrao[];
	private String classeDoPadrao;

	public Pattern(int numParametros){
		valoresDoPadrao=new double[numParametros];
	}
	/**Retorna a classe a qual o padrao pertence
	 * @return Classe do padrao
	 */
	public String getClasseDoPadrao() {
		return classeDoPadrao;
	}
	/**Retorna os valores de um determinado padrao
	 * @return Array de todos os valores de um padrao
	 */
	public double[] getValoresDoPadrao() {
		return valoresDoPadrao;
	}
	/**
	 * Retorna o valor de uma posição qualquer do padrao
	 * @param pos Posição do valor que será lido
	 * @return Valor da posição "pos"
	 */
	public double getValorDoPadrao(int pos) {
		return valoresDoPadrao[pos];
	}
	/**Define a classe a qual o padrao pertence
```

```
 * @param classeDoPadrao Classe a qual o padrao pertence
 */
public void setClasseDoPadrao(String classeDoPadrao) {
	this.classeDoPadrao = classeDoPadrao;
}


/**
 * Define o valor de uma posição qualquer dentro do Array[] de valores do padrao
 * @param pos Posição do Array
 * @param value Valor que será inserido na posição
 */
public void setValorDoPadrao(int pos,double value) {
	this.valoresDoPadrao[pos] = value;
}
/**
 * Define todos os valores dentro do Array[] de valores do padrao
 * @param valores Array com os valores que serão inseridos
 */
public void setValoresDoPadrao(double valores[]){
      // valoresDoPadrao=(double[])valores.clone();
       valoresDoPadrao=valores;
}
/**
 * Permite que os objetos PatternComDistancia possam ser comparados, o que permite
 * que eles sejam ordenados
 */
public int compareTo(Object obj){
		int compara=0;
		int count=0;
		double valores[] = ((Pattern)obj).getValoresDoPadrao();

		while (count<valoresDoPadrao.length){
			if(this.valoresDoPadrao[count] != valores[count] ){
				if (this.valoresDoPadrao[count] < valores[count])
					compara=-1;
				else compara=1;

			}//fim do if

			count++;
		}//fim do while

	return compara;
}//fim do compareTo



/**
 * Retorna o Padrao em formato de String
```

```
     * @return String String que representa o conteudo do padrao
     */
    public String getPadraoComoString() {
      int count=0;
      String padrao="";
      while (count < valoresDoPadrao.length){
        padrao+=valoresDoPadrao[count]+" ";
        count++;
      }
      padrao+=this.getClasseDoPadrao();

      return padrao;
    }

}//fim da Classe Pattern


/**
 * @author Miguel Bezerra
 *
 */

public class PatternComDistancia extends Pattern implements Comparable{
    private double distancia;

    /**
     * Construtor de PatternComDistancia recebendo um pattern e inicializando distancia
com valor -1
     * @param padrao Pattern que será especializado para um PatternComDistancia
     */
    public PatternComDistancia(Pattern padrao){
        super(padrao.getValoresDoPadrao().length);
        super.setValoresDoPadrao(padrao.getValoresDoPadrao());
        super.setClasseDoPadrao(padrao.getClasseDoPadrao());
        this.distancia=-1;
    }

    /**
     * Construtor de PatternComDistancia
     * @param numParametros Numero de parametros do padrao
     * @param distancia Distancia entre o padrao de teste e de treinamento para ordenação
     */
    public PatternComDistancia(int numParametros, double distancia){
        super(numParametros);
        this.distancia=distancia;
    }

    /**
     * Permite que os objetos PatternComDistancia possam ser comparados, o que permite
     * que eles sejam ordenados
```

```
     */
    public int compareTo(Object obj){
        if( this.getDistancia() < ((PatternComDistancia)obj).getDistancia() ) return -1;
        if( this.getDistancia() == ((PatternComDistancia)obj).getDistancia() ) return 0;
        if( this.getDistancia() > ((PatternComDistancia)obj).getDistancia() ) return 1;
        else return 0;
    }

    /**
     * Retorna a distancia entre o Padrao de teste e o de treinamento
     * @return Distancia entre os Padroes
     */
    public double getDistancia() {
        return distancia;
    }

    /**
     * Seta a distancia entre o Padrao de teste e o de treinamento
     * @param distancia Distancia entre os Padroes
     */
    public void setDistancia(double distancia) {
        this.distancia = distancia;
    }

}// fim da classe PatternComDistancia
```

Classes para implementação do NN e do KNN

```
/**
 * @author Miguel Bezerra
 *
*/
public class KNN {

	/**
	 * Os metodos contaPadroes(), contaEntradas(), contaSaidas() foram colocados
	 * como private pois só devem ser chamados internamente na classe NN, isso
	 * evitara problemas de leitura aos arquivos, pois os mesmo devem ser lidos
	 * numa ordem especifica
	 */

	private BufferedReader leitorTreinamento, leitorTeste;
	private int numPadroes, numEntradas, numSaidas;
	private ArrayList padroesTreinamento=new ArrayList();
	private ArrayList padroesTeste=new ArrayList();
	private ArrayList padroesComDistancia=new ArrayList();
	private int numVizinhos;

	public KNN(){
	}

	/**
	 * Função que realiza a fase de treinamento do algoritmo NN
	 * @param file Arquivo de treinamento
	 */
	public void treinamento(File file){

		try{
			leitorTreinamento=new BufferedReader(new FileReader(file));
		}//fim do try
		catch(IOException e){
		  JOptionPane.showMessageDialog(null,
		     "O arquivo de treinamento não pôde ser lido",
					"IOException ",
					JOptionPane.ERROR_MESSAGE);
		}//fim do catch

		numPadroes=contaPadroes(leitorTreinamento);
		numEntradas=contaEntradas(leitorTreinamento);
		numSaidas=contaSaidas(leitorTreinamento);
		//Lê os padroes de treinamento
		lerPadroes(leitorTreinamento,padroesTreinamento,numEntradas,numSaidas);

	}//fim do metodo treinamento
```

```
/**
 * Função que realiza a fase de teste do algoritmo NN
 * @param file Arquivo com base de teste
 * @param k Numero de vizinhos mais próximos
 */
 public void teste(File file, int k){


        DecimalFormat formatar = new DecimalFormat("0.00000");
        int numPadroes, numEntradas, numSaidas;
        double erroTotal, taxaDeErro;
        numVizinhos=k;

        try{
              leitorTeste=new BufferedReader(new FileReader(file));
        }//fim do try
        catch(IOException e){
          JOptionPane.showMessageDialog(null,
                                "O arquivo de teste não pôde ser lido",
                                "IOException ",
                                JOptionPane.ERROR_MESSAGE);
        }//fim do catch

        /**
         * Le o cabeçalho do arquivo de teste
         */
        numPadroes=contaPadroes(leitorTeste);
        numEntradas=contaEntradas(leitorTeste);
        numSaidas=contaSaidas(leitorTeste);

        /**
         * Lê os padrões de teste
         */
        lerPadroes(leitorTeste,padroesTeste,numEntradas,numSaidas);

        colocaDistanciaNoPattern(padroesTreinamento,padroesComDistancia);

        erroTotal=0;

        for (int contaTeste=0; contaTeste<padroesTeste.size(); contaTeste++){
              Pattern padraoTeste, padraoTreinamento;
              ArrayList kVizinhos=new ArrayList();
              ArrayList arrayClasse=new ArrayList();
              padraoTeste=(Pattern)padroesTeste.get(contaTeste);

              /**
               * Irá colocar as distancia nos padroes do ArrayList padroesComDistancia
               * extender de Pattern-> PatternComDistancia
               */
```

```
                for (int
contaTreina=0;contaTreina<padroesComDistancia.size();contaTreina++){
                    double distancia;

padraoTreinamento=(PatternComDistancia)padroesComDistancia.get(contaTreina);
                    distancia=distanciaEuclidiana(padraoTeste,(Pattern)padraoTreinamento);

((PatternComDistancia)padroesComDistancia.get(contaTreina)).setDistancia(distancia);

                }//fim do for

                /**
                 * Irá ordenar o ArrayList padroesComDistancia de acordo com ditancia
                 * até o padrão de teste
                 */
                Collections.sort(padroesComDistancia);

                /**
                 * Irá colocar no ArrayList kVizinhos os K-vizinhos mais próximos
                 */
                kVizinhos=vizinhosMaisProximos(padroesComDistancia, numVizinhos);

                /**
                 * arrayClasse irá guardar as quantas vezes cada classe aparece em kVizinhos
                 */
                arrayClasse = contaClasses(kVizinhos);

                /**
                 * Vai procurar em arrayClasse qual classe aparece mais vezes
                 */
                Classe classeEscolhida=selectClasse(arrayClasse, numVizinhos);

                if (!padraoTeste.getClasseDoPadrao().equals(classeEscolhida.getClasse())){
                    erroTotal++;
                }//fim do if
            }//fim do for

            taxaDeErro=(erroTotal/numPadroes);

            String texto =      "O número de padroes de teste foi de: " + (int)numPadroes
+"\n"+
                                 "O número de padroes classificados incorretamente foi de: "+
                                (int)erroTotal +"\n\n" + "A taxa de erro foi de: " +
                                formatar.format(taxaDeErro*100) + "%";

            JOptionPane.showMessageDialog(null, texto, "Resultado",
                                JOptionPane.INFORMATION_MESSAGE);

        }//fim de teste
```

```
	/**
	 * Copia o conteudo de um ArrayList de Pattern para um ArrayList de
PatternComDistancia
	 * @param orig ArrayList de Pattern
	 * @param dest ArrayList de PatternComDistancia
	 */
	private void colocaDistanciaNoPattern(ArrayList orig, ArrayList dest){

		int count=0;
		while(count<orig.size()){
			dest.add(new PatternComDistancia( (Pattern)orig.get(count) ));
			count++;
		}//fim do while

	}// fim de colocaDistanciaNoPattern

	/**
	 * Irá retornar um ArrayList com K-vizinhos mais próximos
	 * @param array ArrayList com todos os PatternComDistancia
	 * @param numVizinhosK Numero de K-vizinhos
	 * @return ArrayList com K-vizinhos mais próximos
	 */
	private ArrayList vizinhosMaisProximos (ArrayList array, int numVizinhosK){

		ArrayList vizinhos = new ArrayList();
		int count=0;
	while (count < numVizinhosK){
		 vizinhos.add(array.get(count));
		 count++;
	}//fim do while

	return vizinhos;

	}//fim do metodo vizinhosMaisProximos

	/**
	 * Irá retornar um ArrayList que informa quantas vezes cada classe aparece no ArrayList
de entrada
	 * @param array ArrayList no qual serão contadas as classes
	 * @return ArrayList de Classes
	 */
	private ArrayList contaClasses(ArrayList array){
	int count=0;
	ArrayList arrayClasse = new ArrayList();
	while(count < array.size()){

		if(arrayClasse.isEmpty()){
			arrayClasse.add( new
Classe(((PatternComDistancia)array.get(count)).getClasseDoPadrao()));
			count++;
```

```
            }else{
                Classe classe = new
Classe(((PatternComDistancia)array.get(count)).getClasseDoPadrao());
                int cnt=0;
                int size=arrayClasse.size();

                while(cnt<size){
                    if (classe.compareTo(arrayClasse.get(cnt))==1){
                        ((Classe)arrayClasse.get(cnt)).incrementaQuantidade();

                    }else if(cnt==size-1) //indica que esta no último elemento
                        arrayClasse.add(classe);
                    cnt++;
                }//fim do while
             count++;
            }// fim do else
        }//fim do while

        return arrayClasse;
        }//fim do metodo contaClasses

        /**
         * Retorna a Classe que será comparada com o padrao de teste
         * @param array ArrayList que guardar as quantas vezes cada classe aparece em
kVizinhos
         * @return Classe que será comparada com o padrao de teste
         */
        private Classe selectClasse(ArrayList array, int numK){
        Classe classe=null;
        int repeticoesDaClasse=-1;
        int vezes[]=new int[array.size()];
        int count;
        int vizinhos=numK;
        Stack pilha =  new Stack();

        count=0;
        while(count<array.size()){
            vezes[count]=((Classe)array.get(count)).getQuantidade();
            count++;
        }//fim do while

        Arrays.sort(vezes);


        /**
         * Inverte o array usando pilha
         */
        count=0;
        while(count<vezes.length){
           pilha.push(new Integer(vezes[count]));
```

```
        count++;
    }//fim do while
    count=0;
    while(count<vezes.length){
      vezes[count]=((Integer)pilha.pop()).intValue();
      count++;
    }//fim do while

    if (vezes.length>1){
          if (vezes[0]==vezes[1]){
                vizinhos-=1;
                //KNN-Regressivo
                      return
       selectClasse(contaClasses(vizinhosMaisProximos(padroesComDistancia, vizinhos)),
                                              vizinhos);
          }// fim do if
    }//fim do if

    count=0;
    while(count<array.size()){

          int quantidade=((Classe)array.get(count)).getQuantidade();


          if (count == 0 || quantidade > repeticoesDaClasse) {
            classe = ( (Classe) array.get(count));
            repeticoesDaClasse = quantidade;
          } //fim do if


          count++;
    }//fim do while

    return classe;
}//fim do metodo selectClasse

    /**
     * Calcula a distancia entre dois padroes
     * Para calcular a distancia usamos a formula:
     * (d(x,y))^2 := (x1-y1)^2 + (x2-y2)^2 + ... + (xn-yn)^2
     * @param p1 Padrao 1
     * @param p2 Padrao 2
     * @return Distancia entre os Padroes
     */
    public double distanciaEuclidiana(Pattern p1, Pattern p2){

          double p1Valores[]=p1.getValoresDoPadrao();
          double p2Valores[]=p2.getValoresDoPadrao();
          double distancia=0;
```

```
        //testa se os dois arrays são do mesmo tamanho
        if (p1Valores.length==p2Valores.length){
            for (int count=0; count<p1Valores.length; count++){
                distancia+=Math.pow((p1Valores[count]-p2Valores[count]),2);
                                        //distancia+=(p1Valores[count]-
                  p2Valores[count])*(p1Valores[count]-p2Valores[count]);
            }//fim do for
        }//fim do if

        return (Math.sqrt(distancia));
}//fim de distanciaEuclidiano
/**
 * Lê cada padrao de treinamento, cria o objeto Pattern e armazena no ArrayList
 * @param in Leitor do arquivo
 */
private void lerPadroes(BufferedReader in, ArrayList patterns, int entradas, int saidas){
        String line=" ";
        StringTokenizer token=new StringTokenizer(line);

        //patterns=new ArrayList();

        /**
         * while conta até ele ler o primeiro padrao que é composto
         * por entradas+saidas tokens
         */
        while (token.countTokens()!= entradas+saidas){
            try {
                line=in.readLine();
            }//fim do try
            catch (IOException iOException) {
              JOptionPane.showMessageDialog(null,
                 "Não conegue ler a linha com Nº de Padroes",
                                    "IOException",
                                    JOptionPane.ERROR_MESSAGE);
            } //fim do catch

            try{
            token=new StringTokenizer(line);
            }catch (NullPointerException nullPointerException) {
               JOptionPane.showMessageDialog(null,
                     "Erro de leitura - Verifique o cabeçalho do seu arquivo",
                                        "NullPointerException",
                                        JOptionPane.ERROR_MESSAGE);
            } //fim do catch
        }//fim do while

        //Neste ponto line já contem os dados do 1º padrao do arquivo
        do{
            System.out.println(line);
            Pattern padrao = new Pattern(numEntradas);
```

```
                int count=0;
                token=new StringTokenizer(line);
                //Irá preencher os valores do padrao
                do{

padrao.setValorDoPadrao(count,Double.parseDouble(token.nextToken()));
                    count++;
                }while(count!=entradas);
                //Irá preencher os valores do padrao

                if (saidas==1){
                  padrao.setClasseDoPadrao(token.nextToken());
                }else {
                  padrao.setClasseDoPadrao(line.substring( line.length()-((2 *numSaidas)-
1),line.length() ));
                }// fim do else

                //Adiciona o novo padrao no vetor
                patterns.add(padrao);
                //Le a proxima linha
                    try {
                        line=in.readLine();
                    }//fim do try
                catch (IOException iOException) {
                  JOptionPane.showMessageDialog(null,
                     "Não conegue ler a linha com Nº de Padroes",
                                    "IOException",
                                    JOptionPane.ERROR_MESSAGE);
                }//fim do catch
            }while (line!=null);

        }//fim de lerPadroes
        /**
         * Conta a quantidade de padroes contidos no arquivo de treinamento
         * @param in Leitor do arquivo
         * @return Quantidade de padroes de entrada
         */
        private int contaPadroes(BufferedReader in){
            String line = " ";
            String valor = "";
            StringTokenizer token;
            try {
                line=in.readLine();
                System.out.println(line);
               }//fim do try
            catch (IOException iOException) {
              JOptionPane.showMessageDialog(null,
                                "Não conegue ler a linha com Nº de Padroes",
                                "IOException",
                                JOptionPane.ERROR_MESSAGE);
```

```
            }//fim do catch
        token = new StringTokenizer(line);
while (token.hasMoreElements())
        valor=token.nextToken();
return Integer.parseInt(valor);
}//fim de contaPadroes

/**
 * Conta a quantidade de entradas contidas no arquivo de treinamento
 * @param in Leitor do arquivo
 * @return Quantidade de entradas
 */
private int contaEntradas(BufferedReader in){
        String line = " ";
        String valor = "";
        StringTokenizer token;
        try {
             line=in.readLine();
             System.out.println(line);
        }//fim do try
        catch (IOException iOException) {
          JOptionPane.showMessageDialog(null,
            "Não conegue ler a linha com Nº de Entradas",
                                "IOException ",
                                JOptionPane.ERROR_MESSAGE);
            }//fim do catch
        token = new StringTokenizer(line);
while (token.hasMoreElements())
        valor=token.nextToken();
return Integer.parseInt(valor);
}//fim de contaEntradas

/**
 * Conta a quantidade de saidas contidas no arquivo de treinamento
 * @param in Leitor do arquivo
 * @return Quantidade de saidas
 */
private int contaSaidas(BufferedReader in){
        String line = " ";
        String valor = "";
        StringTokenizer token;
        try {
             line=in.readLine();
             System.out.println(line);
            }//fim do try
        catch (IOException iOException) {
          JOptionPane.showMessageDialog(null,
                                "Não conegue ler a linha com Nº de Saidas",
                                "IOException",
                                JOptionPane.ERROR_MESSAGE);
```

```
                }//fim do catch
            token = new StringTokenizer(line);
        while (token.hasMoreElements())
            valor=token.nextToken();
        return Integer.parseInt(valor);
        }//fim de contaSaidas

        /**
         * @return Retorna o valor de numEntradas.
         */
        public int getNumEntradas() {
            return numEntradas;
        }
        /**
         * @return Retorna o valor de numPadroes.
         */
        public int getNumPadroes() {
            return numPadroes;
        }
}//fim da Classe NN
```

Classe principal para implementação do POC-NN e do POC-KNN

```
/**
 * @author Miguel Bezerra
 *
 */
public class PocNN {

	/**
	 * Os metodos contaPadroes(), contaEntradas(), contaSaidas() foram colocados
	 * como private pois só devem ser chamados internamente na classe PocNN, isso
	 * evitara problemas de leitura aos arquivos, pois os mesmo devem ser lidos
	 * numa ordem específica
	 */

	private BufferedReader leitorTreinamento, leitorTeste;
	private int numPadroes, numEntradas, numSaidas;
	private ArrayList padroesTreinamento=new ArrayList();
	private ArrayList padroesTeste=new ArrayList();
	private ArrayList padroesComDistancia=new ArrayList();
	private PocNNSet pocNNSet = new PocNNSet();
	private double alfaRatio;
	private int numPadroesTreinamento;


	public PocNN(double intervalo){
		this.alfaRatio=intervalo;
	}

	/**
	 * Metodo que realiza o treinamento do algoritmo
	 * @param file Arquivo de treinamento
	 */
	public void treinamento(File file){
		ArrayList arrayDeSubSets = null;

		try{
			leitorTreinamento=new BufferedReader(new FileReader(file));
		}//fim do try
		catch(IOException e){
			JOptionPane.showMessageDialog(null, "IOException ",
					"O arquivo de treinamento não pôde ser lido",
					JOptionPane.ERROR_MESSAGE);
		}//fim do catch

		numPadroes=contaPadroes(leitorTreinamento);
		numEntradas=contaEntradas(leitorTreinamento);
		numSaidas=contaSaidas(leitorTreinamento);

		numPadroesTreinamento=numPadroes;
```

```
//Lê os padroes de treinamento
lerPadroes(leitorTreinamento,padroesTreinamento,numEntradas,numSaidas);

arrayDeSubSets=separaClasses(padroesTreinamento);


/**
 * Faz com que o algoritmo funcione para multiplas classes
 */

int vezes=0;
int vezesAlgoritmo=0;
int dupla=0;
int count=0;
while(count<arrayDeSubSets.size()-1){
        int countInterno=count+1;
        while(countInterno<arrayDeSubSets.size()){
                PocNNSet poc = new PocNNSet();
                ArrayList sq = new ArrayList();
                sq.addAll((ArrayList)arrayDeSubSets.get(count));
                sq.addAll((ArrayList)arrayDeSubSets.get(countInterno));

                poc=selectingPocNN(sq);

                System.out.println("Quantidade prototipo Classe1 + Classe2: "+
sq.size() + "\n" +
                                        "Classe 1: " +
((Pattern)((ArrayList)arrayDeSubSets.get(count)).
                                                get(0)).getClasseDoPadrao() + "\n"
+ "\t\t\t\tX" + "\n" +
                                                "Classe 2: " +
((Pattern)((ArrayList)arrayDeSubSets.

        get(countInterno)).get(0)).getClasseDoPadrao() + "\n" +
                                                        "Nº. Prototipos
Armazenados: " +

        selectingPocNN(sq).getPrototipos().size()+ "\n"+
                                                        "Combinação
número: "+ vezesAlgoritmo + "\n" );

                vezesAlgoritmo++;

                if (poc.getPrototipos().size() >= sq.size())
                        vezes++;

                pocNNSet.appendPocNN(poc);
                //pocNNSet.appendPocNN(selectingPocNN(sq));
```

```
                        dupla++;
                        countInterno++;
                    }//fim do while

                    count++;
                }//fim do while

                System.out.println(vezes);
            }//fim do metodo treinamento


            /**
             * Metodo que realiza o teste do algoritmo
             * @param file Arquivo de teste
             */
            public void teste (File file, int numVizinhos){
                DecimalFormat formatar = new DecimalFormat("0.00");
                padroesTreinamento=pocNNSet.getPrototipos();
                double taxaDeErro;
                int erroTotal;

                try{
                    leitorTeste=new BufferedReader(new FileReader(file));
                }//fim do try
                catch(IOException e){
                    JOptionPane.showMessageDialog(null, "IOException ",
                                "O arquivo de teste não pôde ser lido",
                                JOptionPane.ERROR_MESSAGE);
                }//fim do catch

                numPadroes=contaPadroes(leitorTeste);
                numEntradas=contaEntradas(leitorTeste);
                numSaidas=contaSaidas(leitorTeste);

                //Lê os padroes de teste
                lerPadroes(leitorTeste,padroesTeste,numEntradas,numSaidas);

                colocaDistanciaNoPattern(padroesTreinamento,padroesComDistancia);

                erroTotal=0;

                for (int contaTeste=0; contaTeste<padroesTeste.size(); contaTeste++){
                    Pattern padraoTeste, padraoTreinamento;
                    ArrayList kVizinhos=new ArrayList();
                    ArrayList arrayClasse=new ArrayList();
                    padraoTeste=(Pattern)padroesTeste.get(contaTeste);
                    System.out.println(contaTeste + "\n");
                    /**
                     * Irá colocar as distancia nos padroes do ArrayList
padroesComDistancia
```

```
                        * extender de Pattern-> PatternComDistancia
                        */
                       for (int
contaTreina=0;contaTreina<padroesComDistancia.size();contaTreina++){
                                double distancia;

        padraoTreinamento=(PatternComDistancia)padroesComDistancia.get(contaTreina);

        distancia=distanciaEuclidiana(padraoTeste,(Pattern)padraoTreinamento);

        ((PatternComDistancia)padroesComDistancia.get(contaTreina)).setDistancia(distancia
);

                       }//fim do for

                       /**
                        * Irá ordenar o ArrayList padroesComDistancia de acordo com ditancia
                        * até o padrão de teste
                        */
                       Collections.sort(padroesComDistancia);

                       /**
                        * Irá colocar no ArrayList kVizinhos os K-vizinhos mais próximos
                        */
                       kVizinhos=vizinhosMaisProximos(padroesComDistancia,
numVizinhos);

                       /**
                        * arrayClasse irá guardar as quantas vezes cada classe aparece em
kVizinhos
                        */
                       arrayClasse = contaClasses(kVizinhos);

                       /**
                        * Vai procurar em arrayClasse qual classe aparece mais vezes
                        */
                       Classe classeEscolhida=selectClasse(arrayClasse, numVizinhos);

                       if
(!padraoTeste.getClasseDoPadrao().equals(classeEscolhida.getClasse())){
                                erroTotal++;
                       }//fim do if
               }//fim do for

               taxaDeErro=((double)erroTotal/numPadroes);

               String texto =  "\nO Alfa-Ratio foi de: "+ alfaRatio + "\n" +
               "O número de padrões de treinamento foi de: "+ numPadroesTreinamento
+"\n"+
               "O número de padrões de teste foi de: " + (int)numPadroes +"\n"+
```

```
                "O número de padrões classificados incorretamente foi de: "+ (int)erroTotal +
                "\nO número de protótipos armazenados foi de:
"+padroesTreinamento.size()+"\n\n"+
                "A taxa de erro foi de: " + formatar.format(taxaDeErro*100) + "%";

                JOptionPane.showMessageDialog(null, texto, "Resultado",
                                JOptionPane.INFORMATION_MESSAGE);

                System.out.println(texto);

        }//fim de teste

        /**
         * Retorna a Classe que será comparada com o padrao de teste
         * @param array ArrayList que guardar as quantas vezes cada classe aparece em
kVizinhos
         * @return Classe que será comparada com o padrao de teste
         */
        private Classe selectClasse(ArrayList array, int numK){
                Classe classe=null;
                int repeticoesDaClasse=-1;
                int vezes[]=new int[array.size()];
                int count;
                int vizinhos=numK;
                Stack pilha =  new Stack();

                count=0;
                while(count<array.size()){
                        vezes[count]=((Classe)array.get(count)).getQuantidade();
                        count++;
                }//fim do while

                Arrays.sort(vezes);

                /**
                 * Inverte o array usando pilha
                 */
                count=0;
                while(count<vezes.length){
                        pilha.push(new Integer(vezes[count]));
                        count++;
                }//fim do while
                count=0;
                while(count<vezes.length){
                        vezes[count]=((Integer)pilha.pop()).intValue();
                        count++;
                }//fim do while

                if (vezes.length>1){
                        if (vezes[0]==vezes[1]){
```

```
                                vizinhos-=1;
                                //KNN-Regressivo
                                return
selectClasse(contaClasses(vizinhosMaisProximos(padroesComDistancia, vizinhos)),
                                                vizinhos);
                        }// fim do if
                }//fim do if

                count=0;
                while(count<array.size()){

                        int quantidade=((Classe)array.get(count)).getQuantidade();


                        if (count == 0 || quantidade > repeticoesDaClasse) {
                                classe = ( (Classe) array.get(count));
                                repeticoesDaClasse = quantidade;
                        } //fim do if


                        count++;
                }//fim do while

                return classe;
        }//fim do metodo selectClasse

        /**
         * Irá retornar um ArrayList que informa quantas vezes cada classe aparece no
ArrayList de entrada
         * @param array ArrayList no qual serão contadas as classes
         * @return ArrayList de Classes
         */
        private ArrayList contaClasses(ArrayList array){
                int count=0;
                ArrayList arrayClasse = new ArrayList();
                while(count < array.size()){

                        if(arrayClasse.isEmpty()){
                                arrayClasse.add( new
Classe(((PatternComDistancia)array.get(count)).getClasseDoPadrao()));
                                count++;
                        }else{
                                Classe classe = new
Classe(((PatternComDistancia)array.get(count)).getClasseDoPadrao());
                                int cnt=0;
                                int size=arrayClasse.size();

                                while(cnt<size){
                                        if (classe.compareTo(arrayClasse.get(cnt))==1){
```

```
((Classe)arrayClasse.get(cnt)).incrementaQuantidade();

                                        }else if(cnt==size-1) //indica que esta no último
elemento
                                                arrayClasse.add(classe);
                                        cnt++;
                                    }//fim do while
                                    count++;
                                }// fim do else
                        }//fim do while

                        return arrayClasse;
                }//fim do metodo contaClasses

                /**
                 * Irá retornar um ArrayList com K-vizinhos mais próximos
                 * @param array ArrayList com todos os PatternComDistancia
                 * @param numVizinhosK Numero de K-vizinhos
                 * @return ArrayList com K-vizinhos mais próximos
                 */
                private ArrayList vizinhosMaisProximos (ArrayList array, int numVizinhosK){

                        ArrayList vizinhos = new ArrayList();
                        int count=0;
                        while (count < numVizinhosK){
                                vizinhos.add(array.get(count));
                                count++;
                        }//fim do while

                        return vizinhos;

                }//fim do metodo vizinhosMaisProximos

                /**
                 * Funcão que Seleciona o PocNNSet
                 * @param dataSet Base da dados de treinamento
                 */
                public PocNNSet selectingPocNN(ArrayList dataSet){
                        PocNNSet pocnnSet = new PocNNSet();
                        ArrayList findedPocNN, duasRegioes;
                        double centerPoint[];

                        /**
                         * PASSO 1
                         */
                        findedPocNN=findingPocNN(dataSet);

                        /**
                         * PASSO 2
```

```
                 */
                centerPoint=findCenterPoint(findedPocNN);

                /**
                 * PASSO 3
                 */
                HiperPlano w = new HiperPlano(centerPoint,findedPocNN,alfaRatio);

                /**
                 * PASSO 4
                 */
                pocnnSet.setHiperPlano(w);
                pocnnSet.setPrototipos(findedPocNN);

                /**
                 * PASSO 5
                 */
                duasRegioes=divideEmDuasRegioes(w,dataSet);

                /**
                 * PASSO 6 e 7
                 */
                if(hasMisclassified((ArrayList)duasRegioes.get(0))){
                        /**
                         * PASSO 8
                         */
                        pocnnSet.appendPocNN(
selectingPocNN((ArrayList)duasRegioes.get(0)) );
                }//fim do if
                /**
                 * PASSO 9
                 */
                if (hasMisclassified((ArrayList)duasRegioes.get(1))){
                        /**
                         * PASSO 10
                         */
                        pocnnSet.appendPocNN(
selectingPocNN((ArrayList)duasRegioes.get(1)) );
                }// fim do if

                /**
                 * PASSO 11 e 12
                 */
                return pocnnSet;

        }//fim de selectingPOCNN

        /**
         *Testa se tem algum padrão mal classificado em uma região
         * @param regiao Região que será verificada
```

```
         * @return Boleano true (caso tenha algum padrao mal classificado) ou false
         */
        private boolean hasMisclassified(ArrayList regiao){
                Pattern padrao = (Pattern)regiao.get(0);
                boolean retorno = false;
                int count=0;

                while(count < regiao.size()){

                        if (!padrao.getClasseDoPadrao().equals(
((Pattern)regiao.get(count)).getClasseDoPadrao() ) ){
                                retorno = true;
                                break;
                        }//fim do if

                        count++;

                }//fim do while

                return retorno;
        }//fim de hasMisclassified

        /**
         * Usará o Hiperplano w para separar os padroes em duas regiões
         * @param w Hiperplano que irá separas os padroes em duas regiões
         * @param dataSet Padroes a serem separados
         * @return ArrayList que contém os padroes separados em duas regiões
         */
        private ArrayList divideEmDuasRegioes(HiperPlano w, ArrayList dataSet){
                ArrayList duasRegioes = new ArrayList();
                ArrayList R1 = new ArrayList();
                ArrayList R2 = new ArrayList();
                double resultado;
                int count=0;

                while(count<dataSet.size()){

                        resultado=w.divideInTwoRegions( (
(Pattern)dataSet.get(count)).getValoresDoPadrao() );

                        if (resultado>=w.getIntervaloAceitacao())
                                R1.add(dataSet.get(count));
                        else if (resultado<=-w.getIntervaloAceitacao())
                                R2.add(dataSet.get(count));

                        count++;
                }//fim do while

                duasRegioes.add(R1);
                duasRegioes.add(R2);
```

```
            return duasRegioes;
        }//fim de divideEmDuasRegioes


        /**
         * Acho o valor do centerPoint c
         * @param dataSet Base da dados de treinamento
         * @return Vetor com o valor do centerPoint
         */
        private double[] findCenterPoint(ArrayList dataSet){
            Pattern c = new Pattern(numEntradas);
            int contaEntradas=0;

            while (contaEntradas < numEntradas){
                int contaPadroes=0;
                double media=0;

                while(contaPadroes < dataSet.size()){
                    media+=((Pattern)
dataSet.get(contaPadroes)).getValorDoPadrao(contaEntradas);
                    contaPadroes++;
                }//fim do while

                media = media/2;

                c.setValorDoPadrao(contaEntradas,media);

                contaEntradas++;
            }//fim do while

            return c.getValoresDoPadrao();

        }//fim de findCenterPoint

        /**
         * Seleciona os dois prototipos xp1 e xp2
         * @param dataSet Base da dados de treinamento
         * @return ArrayList com os dois prototipos xp1 e xp2
         */
        private ArrayList findingPocNN(ArrayList dataSet){
            ArrayList arrayDeSubSets = null;
            ArrayList findedPOCNN = new ArrayList();
            Pattern xp1=null;
            Pattern xp2=null;

            /**
             * Irá separar os padroes em classes
             */
            arrayDeSubSets=separaClasses(dataSet);
```

```
                if ( ((ArrayList)arrayDeSubSets.get(0)).size() >
                ((ArrayList)arrayDeSubSets.get(1)).size()){

                        Pattern xm = padraoMedio((ArrayList)arrayDeSubSets.get(0));

                        /**
                         * xp1 e xp2 irão receber o vizinho mais proximo
                         */
                        PatternComDistancia vizinhoMaisProx;

                        vizinhoMaisProx = calculaVizinhoMaisProx(xm,
(ArrayList)arrayDeSubSets.get(1));
                        xp2=new Pattern(vizinhoMaisProx.getValoresDoPadrao().length);
                        xp2.setValoresDoPadrao(vizinhoMaisProx.getValoresDoPadrao());
                        xp2.setClasseDoPadrao(vizinhoMaisProx.getClasseDoPadrao());

                        vizinhoMaisProx=calculaVizinhoMaisProx(xp2,
(ArrayList)arrayDeSubSets.get(0));
                        xp1=new Pattern(vizinhoMaisProx.getValoresDoPadrao().length);
                        xp1.setValoresDoPadrao(vizinhoMaisProx.getValoresDoPadrao());
                        xp1.setClasseDoPadrao(vizinhoMaisProx.getClasseDoPadrao());

                }else{

                        Pattern xm = padraoMedio((ArrayList)arrayDeSubSets.get(1));

                        /**
                         * xp1 e xp2 irão receber o vizinho mais proximo
                         */
                        PatternComDistancia vizinhoMaisProx;

                        vizinhoMaisProx=calculaVizinhoMaisProx(xm,
(ArrayList)arrayDeSubSets.get(0));
                        xp1=new Pattern(vizinhoMaisProx.getValoresDoPadrao().length);
                        xp1.setValoresDoPadrao(vizinhoMaisProx.getValoresDoPadrao());
                        xp1.setClasseDoPadrao(vizinhoMaisProx.getClasseDoPadrao());

                        vizinhoMaisProx=calculaVizinhoMaisProx(xp1,
(ArrayList)arrayDeSubSets.get(1));
                        xp2=new Pattern(vizinhoMaisProx.getValoresDoPadrao().length);
                        xp2.setValoresDoPadrao(vizinhoMaisProx.getValoresDoPadrao());
                        xp2.setClasseDoPadrao(vizinhoMaisProx.getClasseDoPadrao());

                }//fim do else

                findedPOCNN.add(xp1);
                findedPOCNN.add(xp2);

                return findedPOCNN;
```

```
}// fim de findingPOCNN

/**
 * Retorna o vizinho (Pattern) mais proximo de um determinado padrao
 * @param pat Padrao do qual se deseja descobrir o vizinho mais próximo
 * @param dataSet Base da dados de treinamento
 * @return Vizinho (Pattern) mais proximo
 */
private PatternComDistancia calculaVizinhoMaisProx(Pattern pat, ArrayList dataSet){
        ArrayList padroesComDistancia=new ArrayList();

        colocaDistanciaNoPattern(dataSet,padroesComDistancia);

        /**
         * Irá colocar as distancia nos padroes do ArrayList padroesComDistancia
         * em relação xm
         * extender de Pattern-> PatternComDistancia
         */
        for (int contador=0;contador<padroesComDistancia.size();contador++){
                double distancia;

distancia=distanciaEuclidiana(pat,(Pattern)padroesComDistancia.get(contador));

((PatternComDistancia)padroesComDistancia.get(contador)).setDistancia(distancia);
        }//fim do for

        /**
         * Irá ordenar o ArrayList padroesComDistancia de acordo com ditancia
         * até o padrão de teste
         */
        Collections.sort(padroesComDistancia);

        return (PatternComDistancia)padroesComDistancia.get(0);

}//fim de calculaVizinhoMaisProx


/**
 * Separa os padroes em classes
 * @param dataSet Base da dados de treinamento
 * @return Arrays lista com a base da dados de treinamento separa em classes
 */
private ArrayList separaClasses(ArrayList dataSet){
        int contaPadrao=0;
        int contador=0;

        ArrayList copiaDataSet = new ArrayList();
        copiaDataSet.addAll(dataSet);
        ArrayList subConjuntos=new ArrayList();
```

```
            while (!copiaDataSet.isEmpty()){
                int count=0;
                int tamDataSet=copiaDataSet.size();
                Pattern padraoN = (Pattern)copiaDataSet.get(contaPadrao);
                ArrayList arrayN = new ArrayList();

                /**
                 * Coloca em arrayN todos os padroes da mesma classe que
                 * padraoN
                 */
                while (count<tamDataSet){

                    if(padraoN.getClasseDoPadrao().

        equals(((Pattern)copiaDataSet.get(count)).getClasseDoPadrao())){
                        arrayN.add(((Pattern)copiaDataSet.get(count)));
                        copiaDataSet.remove(count);
                        tamDataSet=copiaDataSet.size();
                        continue;
                    }//fim do if
                    count++;
                }//fim do while


                contador++;
                subConjuntos.add(arrayN);
            }//fim do while

            return subConjuntos;
        }//fim de separaClasses


        /**
         *Retorna o padrao médio (Xm) de uma classe
         * @param dataSet ArrayList que contem todos os padroes de uma classe
         * @return Retorna o padrao médio (Xm)
         */
        private Pattern padraoMedio(ArrayList dataSet){
            Pattern xm = new Pattern(numEntradas);
            int contaEntradas=0;

            while (contaEntradas < numEntradas){
                int contaPadroes=0;
                double media=0;

                while(contaPadroes < dataSet.size()){
                    media+=((Pattern)
dataSet.get(contaPadroes)).getValorDoPadrao(contaEntradas);
                    contaPadroes++;
```

```
                }//fim do while

                media = media/dataSet.size();

                xm.setValorDoPadrao(contaEntradas,media);

                contaEntradas++;
            }//fim do while

            xm.setClasseDoPadrao(((Pattern) dataSet.get(0)).getClasseDoPadrao());

            return xm;
        }//fim de padraoMedio


        /**
         * Copia o conteudo de um ArrayList de Pattern para um ArrayList de
PatternComDistancia
         * @param orig ArrayList de Pattern
         * @param dest ArrayList de PatternComDistancia
         */
        private void colocaDistanciaNoPattern(ArrayList orig, ArrayList dest){

            int count=0;
            while(count<orig.size()){
                dest.add(new PatternComDistancia( (Pattern)orig.get(count) ));
                count++;
            }//fim do while

        }// fim de colocaDistanciaNoPattern


        /**
         * Calcula a distancia entre dois padroes
         * Para calcular a distancia usamos a formula:
         * (d(x,y))^2 := (x1-y1)^2 + (x2-y2)^2 + ... + (xn-yn)^2
         * @param p1 Padrao 1
         * @param p2 Padrao 2
         * @return Distancia entre os Padroes
         */
        public static double distanciaEuclidiana(Pattern p1, Pattern p2){

            double p1Valores[]=p1.getValoresDoPadrao();
            double p2Valores[]=p2.getValoresDoPadrao();
            double distancia=0;

            //testa se os dois arrays são do mesmo tamanho
            if (p1Valores.length==p2Valores.length){
                for (int count=0; count<p1Valores.length; count++){
                    distancia+=Math.pow((p1Valores[count]-p2Valores[count]),2);
```

```
                        //distancia+=(p1Valores[count]-
p2Valores[count])*(p1Valores[count]-p2Valores[count]);
                    }//fim do for
                }//fim do if
                return (Math.sqrt(distancia));
        }//fim de distanciaEuclidiano
        /**
         * Lê cada padrao de treinamento, cria o objeto Pattern e armazena no ArrayList
         * @param in Leitor do arquivo
         */
        private void lerPadroes(BufferedReader in, ArrayList patterns, int entradas, int
saidas){
                String line=" ";
                StringTokenizer token=new StringTokenizer(line);

                //patterns=new ArrayList();

                /**
                 * while conta até ele ler o primeiro padrao que é composto
                 * por entradas+saidas tokens
                 */
                while (token.countTokens()!= entradas+saidas){
                        try {
                                line=in.readLine();
                        }//fim do try
                        catch (IOException iOException) {
                                JOptionPane.showMessageDialog(null,
                                                "Não conegue ler a linha com Nº de Padroes",
                                                "IOException",
                                                JOptionPane.ERROR_MESSAGE);
                        } //fim do catch

                        try{
                                token=new StringTokenizer(line);
                        }catch (NullPointerException nullPointerException) {
                                JOptionPane.showMessageDialog(null,
                                                "Erro de leitura - Verifique o cabeçalho do seu
                                                arquivo",
                                                "NullPointerException",
                                                JOptionPane.ERROR_MESSAGE);
                        } //fim do catch
                }//fim do while

                //Neste ponto line já contem os dados do 1º padrao do arquivo
                do{
                        System.out.println(line);
                        Pattern padrao = new Pattern(numEntradas);
                        int count=0;
                        token=new StringTokenizer(line);
                        //Irá preencher os valores do padrao
```

```
                    do{

        padrao.setValorDoPadrao(count,Double.parseDouble(token.nextToken()));
                            count++;
                    }while(count!=entradas);
                    //Irá preencher os valores do padrao

                    if (saidas==1){
                            padrao.setClasseDoPadrao(token.nextToken());
                    }else {
                            padrao.setClasseDoPadrao(line.substring( line.length()-((2
*numSaidas)-1),line.length() ));
                    }// fim do else

                    //Adiciona o novo padrao no vetor
                    patterns.add(padrao);
                    //Le a proxima linha
                    try {
                            line=in.readLine();
                    }//fim do try
                    catch (IOException iOException) {
                            JOptionPane.showMessageDialog(null,
                                    "Não conegue ler a linha com Nº de Padroes",
                                    "IOException",
                                    JOptionPane.ERROR_MESSAGE);
                    }//fim do catch
            }while (line!=null);
        }//fim de lerPadroes
        /**
         * Conta a quantidade de padroes contidos no arquivo de treinamento
         * @param in Leitor do arquivo
         * @return Quantidade de padroes de entrada
         */
        private int contaPadroes(BufferedReader in){
                String line = " ";
                String valor = "";
                StringTokenizer token;
                try {
                        line=in.readLine();
                        System.out.println(line);
                }//fim do try
                catch (IOException iOException) {
                        JOptionPane.showMessageDialog(null,
                                "Não conegue ler a linha com Nº de Padroes",
                                "IOException",
                                JOptionPane.ERROR_MESSAGE);
                }//fim do catch
                token = new StringTokenizer(line);
                while (token.hasMoreElements())
                        valor=token.nextToken();
```

```
		return Integer.parseInt(valor);
	}//fim de contaPadroes
	/**
	 * Conta a quantidade de entradas contidas no arquivo de treinamento
	 * @param in Leitor do arquivo
	 * @return Quantidade de entradas
	 */
	private int contaEntradas(BufferedReader in){
		String line = " ";
		String valor = "";
		StringTokenizer token;
		try {	line=in.readLine();
			System.out.println(line);
		}//fim do try
		catch (IOException iOException) {
			JOptionPane.showMessageDialog(null,
					"Não conegue ler a linha com Nº de Entradas",
					"IOException ",
					JOptionPane.ERROR_MESSAGE);
		}//fim do catch
		token = new StringTokenizer(line);
		while (token.hasMoreElements())
			valor=token.nextToken();
		return Integer.parseInt(valor);
	}//fim de contaEntradas
	/**
	 * Conta a quantidade de saidas contidas no arquivo de treinamento
	 * @param in Leitor do arquivo
	 * @return Quantidade de saidas
	 */
	private int contaSaidas(BufferedReader in){
		String line = " ";
		String valor = "";
		StringTokenizer token;
		try {
			line=in.readLine();
			System.out.println(line);
		}//fim do try
		catch (IOException iOException) {
			JOptionPane.showMessageDialog(null,
					"Não conegue ler a linha com Nº de Saidas",
					"IOException",
					JOptionPane.ERROR_MESSAGE);
		}//fim do catch
		token = new StringTokenizer(line);
		while (token.hasMoreElements())
			valor=token.nextToken();
		return Integer.parseInt(valor);
	}//fim de contaSaidas
}//fim da Classe PocNN
```