Uma Metodologia de Inicialização para um Sistema Neuro-*Fuzzy*

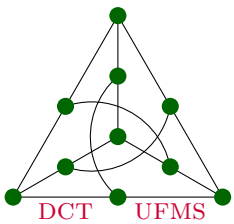
Graciela Lecireth Meza Lovón

Dissertação de Mestrado

Orientação: Profª. Drª. Maria Bernadete Zanusso

Área de Concentração: Ciências da Computação

Durante a elaboração desse trabalho a autora recebeu apoio financeiro da CAPES.



Departamento de Computação e Estatística
Centro de Ciências Exatas e Tecnologia
Universidade Federal de Mato Grosso do Sul
Abril, 2007

Uma Metodologia de Inicialização para um Sistema Neuro-*Fuzzy*

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Graciela Lecireth Meza Lovón e aprovada pela comissão julgadora.

Campo Grande/MS, Abril, 2007.

Banca Examinadora:

- Profª. Drª. Maria Bernadete Zanusso (Orientadora) (DCT-UFMS)
- Profª. Drª. Luciana Cambraia Leite (DEL-UFMS)
- Profª. Drª. Maria Carolina Monard (ICMC-USP)

Aos meus amados pais, Chela e Edward.

Agradecimentos

A Deus, por tudo o que me deu na vida.

À minha orientadora, Profª. Drª. Maria Bernadete Zanusso, pela confiança depositada, pelos muitos conselhos, pela paciência e boa disposição com a qual sempre me atendeu durante estes dois anos. Agradeço-lhe, também, pela exigência de qualidade e rigor na correção deste trabalho. Certamente, sem a sua orientação a realização deste projeto não teria sido possível.

Ao pessoal do DCT-UFMS. Aos docentes, por ter-me dado a oportunidade de aprender com eles no programa de mestrado; e aos funcionários, por ter-me acolhido sempre que precisei.

Aos meus pais, Chela e Edward, que apesar da distância sempre estiveram comigo me apoiando. Agradeço-lhes pelo exemplo, pelo amor incondicional e compreensão que têm-me dado durante toda a minha vida. Aos meus irmãos, Johanna e Giovanni, pelo apoio, incentivo e amor. A minha avó, Zoila, pelo exemplo de vida e de fortaleza.

Aos meus amigos peruanos, que reconfortaram e alentaram-me nos momentos difíceis, em que a resistência parecia acabar. Aos meus amigos brasileiros que presentearam-me com sua sincera amizade desde o primeiro dia que cheguei ao Brasil.

Finalmente, agradeço à CAPES pelo apoio financeiro.

Resumo

A definição da arquitetura de uma rede neural artificial que forneça um desempenho aceitável em problemas de classificação de dados constitui uma tarefa complexa de tentativa e erro. Isto é notório em redes do tipo Perceptron de Múltiplas Camadas (PMC), nas quais a eficiência do aprendizado medida em termos da sua capacidade de generalização num conjunto de dados depende, entre outros parâmetros, do número de camadas, do número de neurônios e das conexões entre seus neurônios. Por outro lado, em redes baseadas em conhecimento simbólico, a arquitetura e os pesos das conexões podem ser determinados por um método de mapeamento de regras. No entanto, encontrar uma boa base de regras para representar conhecimento é um processo custoso e difícil, não só pela necessidade de ter de um especialista no domínio, também, pela aplicação das técnicas de aquisição, ou extração, de seu conhecimento. A teoria de *rough sets* se apresenta como uma alternativa para contornar este problema. Esta se fundamenta na aplicação dos conceitos matemáticos de classes de equivalência e conjunto quociente num ambiente de incertezas, para gerar regras de dependência a partir de um conjunto de dados.

O Sistema Neuro-*Fuzzy* Diferenciável e Interativo (SNFDI), objeto de estudo deste trabalho, tem como núcleo uma rede neural do tipo PMC com 3 camadas cujos neurônios realizam operações lógicas **e** e **ou**, e é treinada com o algoritmo retropropagação. As operações são efetuadas usando t-normas diferenciáveis, que permitem a diferenciação da função de erro quadrático médio, e interativas, que trazem vantagens sobre os operadores de máximo e de mínimo num domínio de dados com diferentes graus de granularidade nos atributos.

Este projeto tem por objetivo aplicar a teoria de *rough sets* para propor uma metodologia de inicialização para o SNFDI, em que o número de neurônios na camada intermediária da rede e os pesos iniciais para algumas das suas conexões entre as três camadas são determinados mapeando regras. O algoritmo genético, aplicado sobre populações de indivíduos formados por regras, é introduzido para determinar a melhor combinação delas que ao ser mapeadas fornecem um SNFDI com uma boa capacidade de generalização. Várias das idéias e procedimentos desta metodologia foram anteriormente propostos por Banerjee et al.[BMP98], mas, neste trabalho, tentou-se contornar algumas das suas limitações, principalmente, com as vantagens oferecidas pelo algoritmo genético. O desempenho da introdução da metodologia no SNFDI foi avaliado na classificação de vogais de um dialeto indiano, da presença ou ausência de diabetes e de uma doença cardíaca. Os resultados se mostraram promissores, pois além de se automatizar em parte, o processo de inicialização, obteve-se melhoras na capacidade de generalização para os conjuntos de dados utilizados.

Palavras-chave: Sistemas Inteligentes Híbridos, Sistemas Neuro-*Fuzzy*, *Rough Sets*, Algoritmos Genéticos.

Abstract

The definition of the architecture of an artificial neural network which provides an acceptable performance in data classification problems constitutes a complex task of attempts and error. It is notorious in the multilayer perceptron network, in which the efficiency of the learning, measured in terms of its generalization capacity in a data set, depends, among other parameters, on the number of layers, the number of neurons and the connections among their neurons. On the other hand, on a symbolic knowledge based neural networks, the architecture and the weights of their connections can be determined by a method of mapping rules. However, to find a good base of rules to represent the knowledge is a difficult process, not only because of the need of having a specialist in data domain, but also by the application of techniques of acquisition, or extraction of his knowledge. In order to solve this problem, the rough sets theory proved to be an alternative. That theory is based on the application of mathematical concepts of equivalency classes and quotient set, in an environment of uncertainty, to generate dependency rules from a data set.

The Interactive and Differentiable Neuro-Fuzzy System (IDNFS), which is the study object of this work, has as nucleus a multilayer perceptron neural network with three layers in which its neurons carry out logical operations *and* and *or*, and is trained with the backpropagation algorithm. The operations are executed by using differentiable t-norms, that allow the differentiation of the mean squared error function; and interactive, which give advantages on the operators of maximum and minimum in a data domain with different degrees of granularities among the attributes.

The present project is aimed to apply the rough sets theory to propose a methodology of initialization to the IDNFS, in which the number of neurons in the intermediate layer of the network and the initial weights for some of its connections among the three layers are determined by mapping rules. The genetic algorithm, applied to populations of individuals formed by rules is introduced to determine the best rule combination that results in a SNFDI with a good generalization capacity. Many of the ideas and procedures of our methodology were previously proposed by Banerjee et al.[BMP98]. However, it was tried to overcome some of the limitations found in their methodology, mainly with the advantages offered by the genetic algorithm. The performance of the introduction of the methodology of initialization in IDNFS was evaluated in the classification of the vowels of an indian dialect, of the presence and the absence of diabetes and of a heart disease. The results were very promising not only because the process of initialization was partially automatized but also because the generalization capacity was improved in the data sets used in our experiments.

Keywords: Hybrid Intelligent Systems, Neuro-Fuzzy Systems, Rough Set Theory, Genetic Algorithms.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

1.1 Considerações Iniciais

Nos últimos anos da Inteligência Artificial (IA) o que está se tornando foco da atenção é o desenvolvimento de Sistemas Inteligentes Híbridos (SIHs), que são o resultado da combinação de duas ou mais técnicas distintas, sendo pelo menos uma delas de IA, para resolver um determinado problema. A motivação do desenvolvimento destes sistemas, deve-se a que algumas técnicas podem ser adequadas para solucionar determinados problemas, embora não podem sê-lo para outros. Combinando várias dessas técnicas é possível aproveitar as suas vantagens e superar as suas limitações na solução de um determinado problema [LCBS05].

Uma das melhores hibridações de SIHs é aquela constituída por sistemas neuro-*fuzzy*. Nesta combinação são capturados os méritos da teoria *fuzzy* e a teoria conexionista, permitindo o desenvolvimento de sistemas com maior tolerância a falhas, maior adaptabilidade e melhor manuseio da incerteza [NKK97] [MA03]. Os trabalhos de Pedrycz [Ped91], Jang [Jan91], Mitra et al. [MP92] e Gupta et al. [GR94], são exemplos bastante conhecidos desta integração.

Os sistemas *fuzzy* também têm sido combinados com diversas técnicas da computação evolutiva, principalmente, com Algoritmos Genéticos (AGs). Um exemplo de aplicação é o dos controladores *fuzzy* que utilizam AGs para a criação automática da suas bases de regras; como é apresentado nos trabalhos de Wu et al. [WL03] e Ishibuchi [INYT95].

Por outra parte, nos inícios dos anos 80' começaram-se as pesquisas em uma nova forma de manusear a ambigüidade. Essas pesquisas conduziram ao surgimento de uma nova teoria conhecida como teoria de *rough sets*. Desde sua aparição, a teoria de *rough sets* tem sido intensamente pesquisada, e atualmente, são várias as áreas em que está sendo aplicada. Dentro de IA, seu estudo está relacionado principalmente com o aprendizado de máquina, a aquisição do conhecimento, a análise de decisões, a descoberta de conhecimento, os sistemas especialistas e o reconhecimento de padrões [PM04] [PJBSZ95].

Em processos de descoberta de conhecimento, por exemplo, aplicando os conceitos desta

teoria, é possível identificar padrões, a partir dos quais podem ser derivadas regras de produção, como visto nas pesquisas de Hassaniena [Has07], Yanga et al. [YLL07], Bose et al. [Bos06] e Cabral et al. [CPC06]. Por exemplo, na dissertação de Cabral et al. [CPC06], as regras de produção que foram geradas descrevem padrões de comportamento de clientes fraudulentos em empresas de distribuição de energia elétrica.

Assim mesmo, integrando esta recente teoria com outras, como a teoria *fuzzy* e conexionista, têm sido desenvolvidos diferentes SIHs. As pesquisas de Lingras [Lin96], Honh et al. [HWW07] e Tsai et al. [TCC06] constituem exemplos desta integração.

1.2 Definição do Problema e Justificativa

Em geral, a definição da arquitetura de uma Rede Neural Artificial (RNA), que forneça um desempenho aceitável na solução de um determinado problema, é um processo complexo de tentativa e erro. Isso é notoriamente visível em RNAs do tipo Perceptron de Múltiplas Camadas (PMC), nas quais a velocidade e a eficiência do processo de aprendizado dependem, entre outros parâmetros (como a taxa de aprendizado e o termo *momentum*), de aspectos da sua arquitetura como o número de camadas, o número de neurônios, e as conexões entre seus neurônios. É conhecido que um número insuficiente de neurônios e conexões podem impedir o aprendizado; enquanto que, neurônios e conexões em excesso, podem produzir *overfitting*[1] [BdCL00] [CPK05]. Desta maneira, torna-se necessário um processo automático que explore diversas arquiteturas, verificando o desempenho de cada uma, para que a escolha seja feita adequadamente [LCBS05].

Por outro lado, há alguns anos, já se mostrou interesse em incorporar conhecimento simbólico dentro de uma RNA. Este fato conduziu à criação de um tipo de RNA chamada RNA *baseada no conhecimento* [Fu95a] [Fu95b]. Em uma RNA deste tipo, alguns aspectos da sua arquitetura (como o número de camadas, o número de neurônios dessas camadas, as conexões entre os neurônios e os pesos iniciais das mesmas) são determinados a partir de algum tipo de conhecimento simbólico; e.g., o conhecimento simbólico representado na forma de regras de produção (*fuzzy* ou não) [Fu95b]. Esse aspecto é destacável, pois como dito no parágrafo anterior, o desempenho de uma RNA depende em grande parte, da configuração dada para esses parâmetros.

Lamentavelmente, uma das tarefas mais difíceis e custosas em sistemas baseados no conhecimento, e portanto numa RNA baseada no conhecimento, é a construção da base de conhecimento do mesmo. Por exemplo, em sistemas baseados em regras, precisa-se, na maioria dos casos, de um especialista que as forneça. Durante a fase de formulação das mesmas e dependendo da complexidade do domínio com que se trabalha, pode ser necessário o manuseio de centenas delas, podendo existir o risco de que as regras formuladas sejam recursivas ou até redundantes. Por esta razão, esta tarefa é, segundo a maioria dos pesquisadores nessa área, uma das principais dificuldades no desenvolvimento destes sistemas [Gal88] [TS94].

[1] O fenômeno de *overfitting* ocorre, quando após um certo ciclo de treinamento, diminui a taxa de acerto para padrões diferentes de aqueles usados no treinamento. Este fenômeno é interpretado como a memorização dos ruídos e peculiaridades dos padrões de treinamento[BdCL00].

As regras de produção encontradas pela aplicação da teoria de *rough sets* derivam de um tipo de regra mais abstrata chamada regra de dependência[2]. Regras de dependência podem ser mapeadas facilmente em RNAs cujos neurônios realizam operações lógicas, e cuja arquitetura é similar à do PMC. Deste modo, depreende-se a idéia de que o problema descrito poderia ser contornado aplicando os conceitos dessa teoria sobre um conjunto de dados de um domínio específico, e assim, seria possível gerar regras de dependência, cujo mapeamento em uma RNA, determinaria alguns aspectos da sua arquitetura, como os citados nos parágrafos anteriores.

Por outro lado, o Sistema Neuro-*Fuzzy* Diferenciável e Interativo (SNFDI)[3], implementado por Oliveira [Oli06], é um sistema neuro-*fuzzy* que utiliza as t-normas diferenciáveis e interativas, propostas por Zanusso [Zan97]. É importante mencionar que no presente trabalho, considera-se a um sistema neuro-*fuzzy* como uma RNA de tipo especial; por essa razão, é possível se referir ao SNFDI em termos de RNA. Este detalhe será esclarecido na Seção 2.4.

O SNFDI atua como um sistema classificador cujo conhecimento é adquirido usando um mecanismo de aprendizado indutivo supervisionado, i.e., o SNFDI é um sistema com a capacidade de classificar um objeto (exemplo, dado) desconhecido, após o término de um processo de aprendizado (treinamento), usando um conjunto de exemplos que possuem uma classificação conhecida. Assim mesmo, o núcleo do SNFDI tem uma arquitetura de RNA de três camadas (entrada, intermediária e saída) similar ao PMC e é treinado com uma versão modificada do algoritmo retropropagação, em que são consideradas as t-normas mencionadas.

No processo de inicialização do SNFDI, como implementado por Oliveira [Oli06], o número de neurônios da camada de entrada e de saída são trivialmente determinados, com base no número de atributos e no número de classes do conjunto de dados utilizado. Os pesos das conexões são inicializados aleatoriamente. O número de neurônios da camada intermediária, a taxa de aprendizagem e de *momentum* são inicializados manualmente. Assim, a determinação da arquitetura adequada para a solução de um determinado problema é feita mediante um processo manual de tentativa e erro. Este empirismo constitui um problema para a utilização do SNFDI, pois faz da determinação da arquitetura adequada uma tarefa cansativa e custosa em termos de tempo.

1.3 Objetivo

O objetivo principal deste projeto é propor uma metodologia de inicialização para o SNFDI. No contexto da metodologia proposta, entenda-se "inicializar" como determinar o número de neurônios da camada intermediária e os pesos iniciais das conexões entre os seus neurônios.

[2] No contexto de *rough sets*, uma regra de dependência indica os atributos dos quais os possíveis valores de outro atributo dependem. Ver Secão 3.6.6.

[3] Vista a possibilidade da existência de sistemas neuro-*fuzzy* que usem outras t-normas diferenciáveis e interativas, a denominação dada a este sistema ainda está em estudo; contudo, neste projeto, optou-se por conservar essa denominação.

Objetivos Específicos:

- Definir e implementar a metodologia proposta.
- Experimentar o SNFDI aplicando a metodologia proposta sobre vários conjuntos de dados.
- Experimentar o SNFDI sem a aplicação da metodologia em questão sobre vários conjuntos de dados.
- Comparar a precisão atingida, usando a metodologia, com aquela atingida por outros classificadores da literatura.
- Comparar a precisão alcançada, usando a metodologia, com aquela alcançada sem a aplicação da mesma.

Espera-se que mediante o uso da metodologia proposta, o problema da inicialização seja contornado em parte, e que o desempenho do SNFDI melhore, no sentido de aumentar a sua capacidade de generalização.

1.4 Metodologia

Em uma RNA do tipo PMC, a inicialização baseada em algum tipo de conhecimento simbólico, por exemplo, regras de dependência, determina uma arquitetura e pesos iniciais, que provavelmente estejam mais próximos aos pesos ideais, i.e., àqueles que minimizem o EQM, e que conseqüentemente, contribuam a melhorar o desempenho da RNA em questão. Conhecendo que a teoria de *rough sets* permite gerar regras de dependência, sendo estas últimas facilmente mapeadas na arquitetura de uma RNA, é válido acreditar que é possível aplicar esta teoria para criar uma metodologia que além de automatizar em parte o processo de inicialização, traga melhoras à precisão da classificação do SNFDI.

A metodologia proposta aborda o problema da inicialização a partir da perspectiva híbrida *rough sets* - AGs. Esta metodologia está dividida em três etapas que serão descritas detalhadamente no Capitulo 4, mas visando dar uma idéia geral das mesmas, serão sucintamente expostas a seguir.

- Na etapa 1, os exemplos fuzzificados de treinamento são preparados para poderem ser utilizados nas etapas posteriores. São realizados processos como binarização, remoção de objetos repetidos, e remoção de objetos pouco freqüentes (ver Seção 4.1).
- Na etapa 2, aplicando a teoria de *rough sets*, é realizada a geração de regras de dependência, podendo estas serem de dois tipos. As regras do primeiro tipo têm o propósito de distingüir as diferentes classes de um domínio. As regras do segundo tipo têm a finalidade de discriminar os vários representantes de uma determinada classe de um domínio em particular (ver Seção 4.2).

- Na etapa 3, integrando uma regra do primeiro tipo com uma do segundo é criada uma única regra que cumpre a função das regras de ambos os tipos, i.e., discerne as classes e discerne os representantes de uma classe.

  Na metodologia proposta foi feita a suposição de que nem todas as regras integradas de uma determinada classe são necessárias; conseqüentemente, nem todas elas devem ser mapeadas no SNFDI. A escolha das regras integradas é feita mediante a aplicação de AGs, isto significa que o AG busca no espaço de regras integradas a combinação delas que ao ser mapeada em um SNFDI determina a arquitetura e pesos que conduzem a uma boa capacidade de generalização. No AG proposto, cada indivíduo da população é uma combinação específica das regras de dependência e dá origem a um SNFDI com uma determinada configuração inicial; ou seja, dá origem a um SNFDI único. Os indivíduos são avaliados mediante a função de aptidão que é baseada na taxa de acertos, no coeficiente *kappa*, e no número de neurônios da camada intermediária (ver Seção 4.3).

Aplicando a metodologia proposta no SNFDI, sobre os conjuntos de dados mencionados, foram realizados vários experimentos, cujos resultados mostraram que a metodologia proposta, além de automatizar parcialmente a inicialização do SNFDI, melhora a capacidade de generalização para os conjuntos de dados testados, constituindo um resultado destacável, pois, como é conhecido, obter um alto nível de generalização é um dos objetivos mais importantes em problemas de classificação.

1.5 Trabalhos Relacionados

Foram revisados dois trabalhos relacionados.

O primeiro, proposto por Yasdi [Yas95], apresenta uma metodologia com 3 etapas. Na primeira etapa, são geradas regras de dependência a partir de um conjunto de treinamento, usando a teoria de *rough sets*. Na segunda etapa, é criado, com base nas regras geradas, um grafo nomeado esquema de regras. Posteriormente, este esquema é projetado numa estrutura de RNA similar ao PMC. Essa projeção determina o número de camadas intermediárias, o número de neurônios por camada, as conexões entre neurônios e os pesos das mesmas. Na terceira etapa, a RNA projetada é treinada, usando uma versão modificada do algoritmo retropropagação. Se a porcentagem de acerto não for o 100% sobre o conjunto de treinamento, o esquema de regras é refinado usando a teoria de *rough sets*. Este processo de treinamento-refinamento é repetido até atingir o 100% de acertos. As regras aprendidas (refinadas) são lidas diretamente do esquema de regras.

Apesar da similaridade, em geral, da metodologia de Yasdi [Yas95] com a metodologia apresentada neste texto, ambos os trabalhos têm objetivos diferentes. O objetivo de Yasdi [Yas95] é o refinamento de regras usando a teoria de *rough set* e RNAs, enquanto que o interesse da metodologia aqui apresentada, é a utilização das regras geradas com a finalidade de inicializar uma RNA.

O segundo trabalho revisado, foi o proposto por Banerjee et al. [BMP98]. Similarmente

ao trabalho de Yasdi [Yas95], Banerjee et al. [BMP98] usam uma RNA de três camadas com uma estrutura semelhante à do PMC; mas, diferentemente, a RNA utilizada funciona com entradas *fuzzy*. Neste segundo trabalho, são propostos dois métodos para a geração de regras de dependência. No primeiro método, assume-se que cada classe do domínio possui apenas um objeto representativo, e no segundo método, que existe mais de um objeto representativo por classe.

A diferença mais importante entre ambos os métodos é a origem das regras geradas. Assim, no método que não considera a multirepresentatividade, as regras são formuladas usando o conjunto de treinamento inteiro. No método que considera a multirepresentatividade, primeiro, os exemplos de cada classe são separados, formando-se subconjuntos de treinamento, e posteriormente, tratando-se cada subconjunto isoladamente, são encontradas as regras para cada um dos representantes das classes de um determinado problema.

Embora haja diferenças em ambos os métodos, de maneira geral, estes podem ser divididos em quatro etapas. Na primeira, os dados são preparados para poderem ser utilizados pela teoria de *rough sets*. Esta etapa inclui a binarização e eliminação de objetos (exemplos) pouco freqüentes. Na segunda, são geradas regras usando a teoria de *rough sets*. Na terceira etapa, as regras geradas são reunidas em diferentes grupos, e posteriormente, cada grupo é mapeado em uma RNA. Esse mapeamento consiste na criação de uma estrutura de RNA, em que são determinados o número de neurônios da camada intermediária, e os pesos iniciais das conexões da RNA. Por último, na quarta etapa, são realizados o treinamento e o teste das RNAs criadas. O algoritmo utilizado é o retropropagação modificado.

É importante ressaltar que várias idéias da metodologia apresentada neste texto, baseiam-se nas idéias de Banerjee et al. [BMP98], a saber: o procedimento de binarização dos dados, a idéia da remoção de objetos pouco freqüentes, a idéia de abordar a multirepresentatividade dividindo o conjunto de treinamento em vários subconjuntos, e o procedimento para realizar o mapeamento.

Não obstante as idéias de Banerjee et al. [BMP98] sejam destacáveis, apresentam limitações que a metodologia proposta pretende contornar. Tais limitações são descritas a seguir:

- Os grupos de regras são formados considerando todas as combinações de regras possíveis. Se o número de regras geradas é pequeno, isso não implicará um problema, mas no caso oposto, sim.
- Não é proposto um método único que considere tanto a monorepresentatividade quanto a multirepresentatividade. Isto supõe a um problema porque muitas vezes não é conhecido antecipadamente se um domínio tem classes multirepresentativas, ou não.

1.6 Organização do Texto

O texto está organizado em 6 capítulos.

No Capítulo 2 faz-se uma breve introdução a SIHs e às técnicas relacionadas com o desenvolvimento da metodologia proposta, e com o sistema neuro-*fuzzy* sobre o qual a metodologia é aplicada. Também, expõe-se detalhadamente o sistema neuro-*fuzzy* em questão.

No Capítulo 3 são apresentados os conceitos básicos da teoria de *rough sets*, bem como, o procedimento de geração de regras de dependência usando esta teoria. Também, é explicada a discretização de atributos e a sua relação com a teoria de *rough sets*.

No Capítulo 4 são descritas detalhadamente as etapas da metodologia proposta para a inicialização do SNFDI.

No Capítulo 5 são apresentados os resultados experimentais da aplicação da metodologia em questão sobre o SNFDI, utilizando diferentes conjuntos de dados. Também, é feita a análise comparativa desses resultados, confrontando-os, tanto com resultados já publicados, quanto com resultados obtidos, sem aplicar a metodologia em questão.

O Capítulo 6 encerra esta dissertação, expondo as conclusões, limitações, contribuições e propostas para trabalhos futuros.

Este documento possui dois apêndices, onde são apresentados conceitos complementares ao entendimento deste trabalho.

No Apêndice A são estudados os conceitos básicos da teoria *fuzzy*, que se relacionam com a pesquisa realizada.

No Apêndice B é dada uma introdução sucinta à computação granular, relacionando-a com a teoria *fuzzy* e a teoria de *rough sets*.

Capítulo 2

Sistemas Inteligentes Híbridos

2.1 Considerações Iniciais

Atualmente, um dos principais objetivos da IA é a integração de várias técnicas para criar sistemas mais robustos. A esses sistemas tem-se dado o nome de Sistemas Inteligentes Híbridos (SIHs). A motivação para a criação de SIHs surge do fato de que diferentes técnicas podem ser adequadas para resolver determinados tipos de problemas, mas podem apresentar deficiências na resolução de outros. Assim, a idéia principal do desenvolvimento de SIHs é combinar várias técnicas com o intuito de superar as desvantagens que cada uma apresenta individualmente na resolução de um problema de interesse [LCBS05].

É importante ressaltar que combinar várias técnicas na criação de um sistema inteligente não significa, necessariamente, melhorar o desempenho de um sistema como um todo; pelo contrário, muitas vezes, o resultado obtido por um SIH é inferior ao resultado obtido, usando apenas uma das técnicas que o compõem. Portanto, os SIHs podem ser eficientes, desde que sejam justificados, desenvolvidos e utilizados corretamente [LCBS05].

Este capítulo tem como objetivos introduzir o sistema neuro-*fuzzy* sobre o qual foi aplicada a metodologia proposta, e apresentar os conceitos de algoritmos genéticos relacionados com a metodologia em questão. A teoria de *rough sets*, que também foi utilizada na sua definição, não será exposta neste capítulo, mas no Capítulo 3, pois para seu entendimento é preciso descrever detalhadamente alguns dos seus conceitos.

Nas seções 2.2, 2.3 e 2.4, dar-se-á uma sucinta descrição sobre sistemas *fuzzy*, redes neurais artificiais, e sistemas neuro-*fuzzy*, respectivamente. Posteriormente, na Seção 2.5 será apresentado o sistema neuro-*fuzzy* para o que foi definida a metodologia proposta, descrevendo aspectos como a fuzzificação das entradas, a sua arquitetura e funcionamento. Além disso, será apresentado um processo complementar ao sistema em questão, a saber, a extração de regras a partir do SNFDI já treinado. Por último, na Seção 2.6, serão expostos os conceitos de algoritmos genéticos.

2.2 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNAs) pertencem à área da IA conhecida como *conexionismo*. Estas estão inspiradas na estrutura física dos neurônios biológicos e do sistema nervoso e são consideradas como sistemas paralelos e distribuídos compostos de unidades de processamento simples (neurônios artificiais), distribuídas em camadas e interligadas por conexões que, na maioria dos casos, têm associados pesos que são usados para ponderar os estímulos de entrada a estas unidades [BdCL00] [Hay98].

Entre suas principais características encontram-se, por uma parte, a sua característica de paralelismo e distribuição própria da sua arquitetura; e por outra, a sua capacidade de generalização a qual lhe permite extrair conhecimento não fornecido explicitamente; i.e., uma RNA é capaz de dar respostas válidas para dados não apresentados na fase de aprendizagem [BdCL00] [Hay98]. Assim mesmo, a sua capacidade de aprender por meio de exemplos implica que não é necessário algum outro tipo de conhecimento prévio, e.g., regras de produção fornecidas por um especialista. Esta é uma característica importante, pois dependendo do domínio, expressar o conhecimento em forma de regras não é sempre simples [NKK97].

Por outro lado, uma RNA é um modelo não linear generalizado, cujos parâmetros são determinados a partir de um conjunto de dados e por meio de algoritmos eficientes [EA05b]. A não linearidade é uma propriedade importante, sobretudo quando o mecanismo responsável da geração de dados de entrada é inerentemente não linear (e.g., sinal de voz). Ainda mais, a maioria dos problemas reais são de natureza não linear [Hay98].

Assim mesmo, uma RNA tem a característica de aprender de seu ambiente num processo iterativo e melhorar seu funcionamento através do aprendizado que é realizado, usando algum algoritmo que ajusta os pesos sinápticos e outros parâmetros (e.g., nível de bias), com a finalidade de obter um objetivo desejado. Está é, sem dúvida, uma das características mais importantes das RNAs [BdCL00] [Hay98].

Por outro lado, a maneira pela qual os neurônios de uma RNA estão interconectados dá lugar a diferentes tipos de arquiteturas. Assim mesmo, não existe um único algoritmo de aprendizado, mas uma variedade deles. O tipo de algoritmo de aprendizado usado depende da arquitetura escolhida e do tipo de aprendizado desejado. Por sua vez, a escolha do aprendizado depende dos dados (exemplos) disponíveis; i.e., se os dados são do tipo entrada-saída, então pode-se usar um algoritmo de *aprendizado supervisionado*; enquanto que, se as saídas para as entradas (exemplos) são desconhecidas, então pode-se usar um algoritmo de *aprendizado não supervisionado*. A existência desta ampla gama de arquiteturas e algoritmos de aprendizado é um aspecto importante das RNAs, pois permite a solução de uma grande variedade de problemas, por meio da aplicação da RNA mais adequada para um problema dado [NKK97].

No entanto, o desempenho de uma RNA depende dos valores de vários parâmetros, por exemplo, em uma RNA do tipo PCM a velocidade e eficiência do processo de aprendizado depende de parâmetros relacionados com a sua arquitetura, como o valor dos pesos, o número de neurônios, e o número de camadas. Também depende dos parâmetros relacionados com o algoritmo usado para o treinamento, como a taxa de aprendizado e o

termo *momentum* no caso de utilizar o algoritmo retropropagação. Assim, geralmente, a definição de um conjunto de valores adequados é um processo complexo realizado empiricamente, exigindo uma série de tentativas [LCBS05].

2.3 Sistemas *Fuzzy*

A teoria de conjuntos *fuzzy*[1], proposta por Zadeh [Zad65] em 1965, é considerada como uma extensão da teoria de conjuntos clássica, na qual, como é conhecido, a pertinência de um elemento a um conjunto é avaliada como não ou sim, falso ou verdadeiro, ou 0 ou 1. No caso da teoria *fuzzy*, a pertinência não é estritamente absoluta, dando-se a um elemento, a possibilidade de pertencer a um conjunto em um certo grau [ABO00].

No contexto de representação do conhecimento, o fato de dar a um elemento uma pertinência parcial a um determinado conjunto pode ser entendido como modelar a realidade imprecisa de maneira mais apropriada, i.e., mais próxima à forma humana de descrever a realidade. Geralmente, o ser humano, ao encarar um problema, não fornece uma solução precisa, i.e., em termos de números exatos, mas expressa a solução dando classificações qualitativas ou categorias gerais ou conjuntos de possíveis soluções.

O fato de emitir uma resposta dando às variáveis de um determinado problema valores imprecisos e inexatos leva o ser humano a usar conceitos qualitativos em vez de quantitativos, o qual traz a idéia de *variável linguística*. Na teoria *fuzzy*, uma variável linguística apenas permite como valores palavras ou sentenças na linguagem natural ou artificial. Os valores que uma variável linguística pode tomar são chamados *termos primários* ou *termos linguísticos* [EA05b]. Alguns exemplos de termos primários poderiam ser "alta", "média", "baixa", que seriam os valores que a variável linguística *temperatura* poderia tomar.

Geralmente, os sistemas inteligentes clássicos evitam informações vagas, imprecisas ou incertas, pois este tipo de informações é considerado como uma influência negativa aos mecanismos clássicos de inferência. No entanto, a maioria de problemas reais apresentam exatamente as características que os sistemas clássicos evitam. Como resultado, obtém-se sistemas que não conseguem modelar os problemas adequadamente. A teoria *fuzzy*, fornecendo o fundamento matemático para modelar conceitos imprecisos e incertos, possibilita a construção de sistemas mais simples, mais adequados e mais próximos ao raciocínio humano [ABO00].

Os sistemas *fuzzy* estão baseados em regras de produção do tipo **se-então**, en que o antecedente da regra é um conjunto de descrições *fuzzy* dos valores de entrada e o conseqüente uma(s) saída(s) *fuzzy* correspondente(s) à entrada. Tais descrições são feitas usando *termos linguísticos* associados a *variáveis linguísticas*. Um exemplo desse tipo de

[1] O termo *fuzzy* pode ter diversos significados na língua inglesa, mas o conceito básico deste adjetivo sempre está associado com o vago, indistinto e incerto. Apesar de ser comumente traduzido ao português, como nebuloso e difuso, ainda não existe um consenso na comunidade científica brasileira. Por essa razão, optou-se por conservar o termo em inglês. No Apêndice B, apresenta-se uma sucinta exposição dos conceitos básicos da teoria *fuzzy*.

regras é o seguinte:

se a *temperatura* é "alta" **e** o *nível do agua* é "baixo"
então gire "bastante" a manivela.

Os termos linguísticos "alta" e "baixa" representam valores para as variáveis linguísticas de entrada *temperatura* e *nível de agua*. O termo "bastante" descreve uma saída para a variável linguística que representa "quanto a manivela deve ser girada".

No exemplo anterior, o *universo de discurso* para a variável linguística *temperatura* poderia variar de 0°C a 100°C num determinado contexto. Cada possível valor do universo de discurso, para uma variável dada, tem associado um *grau de pertinência* a cada um dos conjuntos *fuzzy* (os conjuntos *fuzzy* também podem ser vistos como termos linguísticos) definidos para essa variável. Geralmente, os graus de pertinência são calculados através de *funções de pertinência*. Uma função de pertinência diz em que medida um elemento de um conjunto satisfaz um conceito representado por um conjunto *fuzzy*.

As regras *fuzzy* usam os operadores **e** e **ou** *fuzzy*; ou o que é o mesmo, os operadores *intersecção* e *união fuzzy* da teoria de conjuntos *fuzzy*. Esses operadores não podem ser escolhidos arbitrariamente, pois devem satisfazer um mínimo de requisitos. As funções que satisfazem os requisitos em questão são chamadas *t-normas* e *t-conormas*, respectivamente [NKK97]. Entre as t-normas e t-conormas mais comuns, encontram-se os operadores *min* e *max*. Estes têm sido amplamente usados no desenvolvimento de sistemas *fuzzy*, e.g., no sistema de controle *fuzzy* de Mandami [Mam74].

Mandami [Mam74] propôs um método de inferência *fuzzy* que foi por muitos anos um padrão para a utilização dos conceitos da lógica *fuzzy* em processamento do conhecimento. Nas regras de produção de Mandami tantos seus antecedentes quanto seus conseqüentes possuem relações *fuzzy*, tal como na regra apresentada anteriormente. O sistema de Mandami possui módulos de aquisição e atuação baseados em valores numéricos, um banco de regras *fuzzy*, módulos de interface que convertem variáveis de entradas em conjuntos *fuzzy* equivalentes e que convertem as variáveis *fuzzy* geradas em variáveis numéricas proporcionais e adequadas para os sistemas de atuação existentes. Finalmente, possui uma máquina de inferência que utiliza como operadores lógicos **e** e **ou**, a t-norma e t-conorma, *min* e *max*, respectivamente [EA05a].

Outro sistema *fuzzy* muito referenciado na literatura é o sistema Takagi-Sugeno-Kang [TS85] [SK88]. Similarmente ao sistema de Mandami, neste sistema os antecedentes das regras também possuem relações *fuzzy*, mas os seus conseqüentes se compõem de equações que relacionam as entradas com as saídas; i.e., a saída é uma função das entradas do sistema [EA05a].

Uma limitação a respeito dos sistemas *fuzzy* é a incapacidade de aprender do seu ambiente. Uma maneira de contornar esse problema, é representar um sistema neuro-*fuzzy* mediante a arquitetura de uma RNA, e depois treiná-la com um algoritmo de aprendizado (e.g., retropropagação). No entanto, os algoritmos baseados no gradiente descendente não podem ser utilizados diretamente no treinamento, pois as funções usadas no processo de inferência destes sistemas, são usualmente não diferenciáveis (e.g., *max* e

min). Geralmente, duas alternativas são empregadas para solucionar este problema. A primeira é substituir as funções (t-norma e t-conorma) de um sistema *fuzzy* por funções diferenciáveis, e a segunda é usar um algoritmo de aprendizado que não seja baseado no gradiente descendente [NKK97].

No caso do SNFDI, Oliveira [Oli06] optou pela primeira alternativa, e usou as t-normas e t-conormas diferenciáveis e interativas de Zanusso [Zan97].

2.4 Sistemas Neuro-*Fuzzy*

As RNAs extraem conclusões genéricas (conhecimento) de um conjunto de treinamento (dados), mediante a interconexão de unidades de processamento simples. Na verdade, uma RNA é um modelo não linear cujos parâmetros são determinados a partir de um conjunto de dados e por meio de algoritmos eficientes. Embora apresente bons resultados numéricos na solução de vários problemas, uma RNA geralmente não permite a incorporação do conhecimento de especialistas numa determinada aplicação, podendo fazer com que o modelo resultante seja centrado nos dados [EA05b].

Por outro lado, um sistema *fuzzy* pode ser utilizado como sistema de apoio à decisão. Estes sistemas podem representar o conhecimento de especialistas sobre um determinado domínio e interpolar decisões a partir de entradas contaminadas com incertezas. O formalismo matemático dos sistemas *fuzzy* permite a representação do raciocínio humano sendo, portanto, aplicado para o desenvolvimento de sistemas baseados no conhecimento. Devido a essa característica, diz-se que os sistemas *fuzzy* são centrados no homem [EA05b].

Assim, levando-se em conta as características da teoria *fuzzy* e conexionista diversos autores as têm combinado procurando aproveitar suas potencialidades e contornar suas deficiências; proporcionando, conseqüentemente, ferramentas mais robustas [ABO00] [Kas96] [NKK97].

Para se referir aos modelos que integram ambos os paradigmas nasce o termo neuro-*fuzzy*. A aplicação desse termo, geralmente, é utilizada para o desenvolvimento de sistemas chamados *sistemas neuro-fuzzy* [ABO00]. Os sistemas ANFIS [Jan93], NEFCLASS [NK95], NEFPROX [NK97a] [NK97b] são sistemas neuro-*fuzzy* bastante conhecidos na literatura. Outros modelos neuro-*fuzzy* são os de Mitra et al. [MP94] [MP92], Pedrycz et al. [PR93], Fu et al. [FS94], Gupta et al. [GR94]. Geralmente, as entradas desses sistemas são conceitos *fuzzy* dos atributos de entrada *crisp*.

Nauck et al. [NKK97] apresenta uma taxonomia para as diferentes combinações de RNAs e sistemas *fuzzy*. Estas são descritas a seguir:

- Redes neurais *fuzzy*: nesta combinação, métodos *fuzzy* são usados para aperfeiçoar as capacidades de aprendizado ou de desempenho de uma RNA. Por exemplo, pode-se usar regras *fuzzy* para mudar a taxa de aprendizado.
- Sistemas concorrentemente neural-*fuzzy*: nestes sistemas, uma RNA e um sistema *fuzzy* trabalham concorrentemente na mesma tarefa, mas sem a influência de um

sobre o outro, ou seja, nenhum dos dois sistemas é usado para determinar os parâmetros do outro.

- Sistemas neuro-*fuzzy* cooperativos: os parâmetros do sistema *fuzzy* (e.g., regras, pesos das regras ou conjuntos *fuzzy*) são determinados por uma RNA. Nestes sistemas, ambas as ferramentas existem, uma independente da outra, e podem ser distinguidas claramente.
- Sistemas neuro-*fuzzy* híbridos: uma RNA e um sistema *fuzzy* são combinados em uma arquitetura homogênea. O sistema pode ser interpretado como uma RNA especial ou como um sistema *fuzzy* implementado em forma de RNA. O sistema como um tudo não pode ser divido, pois é uma estrutura única, similar à arquitetura de uma RNA. A maioria de modelos propostos na literatura pertencem nesta categoria.

Segundo a taxonomia feita por Nauck et al. [NKK97], o Sistema Neuro - *Fuzzy* Diferenciável e Interativo sobre o qual se aplica a metodologia proposta é um sistema neuro-*fuzzy* híbrido. Portanto, pode ser visto como um sistema *fuzzy* estruturado na forma de uma RNA ou uma RNA de tipo especial. No restante deste texto, algumas vezes este sistema será referenciado em termos de sistema *fuzzy* e outras, quando conveniente, em termos de RNA.

2.5 Sistema Neuro-*Fuzzy* Diferenciável e Interativo

O Sistema Neuro-*Fuzzy* Diferenciável e Interativo (SNFDI) foi implementado por Oliveira [Oli06]. Sua denominação se deve ao fato de que usa as t-normas diferenciáveis e interativas propostas por Zanusso [Zan97]. Não se descarta a existência de outros sistemas neuro-*fuzzy* que usem outras t-normas diferenciáveis e interativas; levando isso em conta, a denominação dada a este sistema ainda está em estudo.

O SNFDI atua como um sistema classificador, cujo conhecimento é adquirido usando um mecanismo de aprendizado indutivo supervisionado, i.e., na fase de aprendizado (neste caso, treinamento) é usado um conjunto de exemplos que possuem uma classificação conhecida, determinada por um atributo de classe. Após o aprendizado, o sistema é capaz de atribuir uma classe a um exemplo desconhecido, ou seja a um exemplo não utilizado no treinamento.

O SNFDI funciona sobre entradas fuzzificadas, sendo necessário determinar para os valores dos exemplos de um conjunto de dados, seus graus de pertinência aos conjuntos *fuzzy*, definidos para cada atributo do conjunto em questão. Assim, um vetor de entrada m-dimensional é transformado num vetor $3m$-dimensional, em que cada valor do vetor corresponde ao grau de pertinência a conjuntos *fuzzy*, *baixo, médio* e *alto*. Este procedimento é realizado por um pré-processador *fuzzy* que forma parte do SNFDI e que realiza uma tarefa que é indispensável para o seu funcionamento. O pré-processamento *fuzzy* será descrito na Seção 2.5.1.

O SNFDI propriamente dito, i.e., o núcleo do sistema adquire conhecimento usando dados mediante um processo de treinamento em que é usado o algoritmo retropropagação

modificado para incluir as t-normas diferenciáveis e interativas como funções de ativação. Seu estudo abarcará da Seção 2.5.2 à Seção 2.5.6, expondo-se aspectos como as t-normas e t-conormas usadas, a sua arquitetura, o processamento nos neurônios E e OU, e a atualização de pesos.

O SNFDI pode ser aplicado junto com um processo complementário de extração de regras, que tem a finalidade de dar ao sistema a capacidade de explanação das suas respostas, por meio de regras de produção, geradas com base nos pesos do SNFDI já treinado. Este processo será brevemente descrito na Seção 2.5.7. É importante esclarecer que a extração de regras não está relacionada com a geração de regras de dependência realizada na metodologia proposta. Diferentemente das primeiras, estas últimas são geradas a partir de um conjunto de dados, e mapeadas no SNFDI, fato que determina a arquitetura e pesos iniciais das conexões do sistema.

Na Figura 2.1 são mostrados os módulos correspondentes aos processos descritos previamente. A linha contínua marca o caminho do treinamento e teste; a linha pontilhada, a consulta de um exemplo cuja classe é desconhecida.

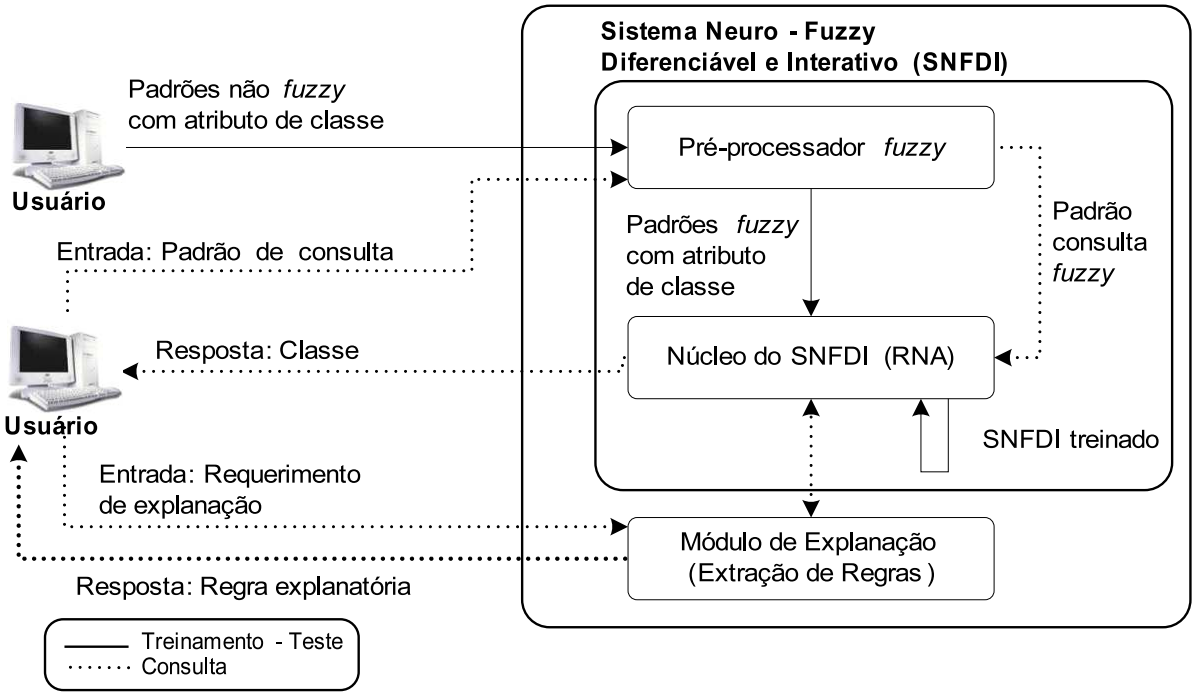


Figura 2.1: Pré-processador *fuzzy*, SNFDI propriamente dito e Módulo de Explanação.

2.5.1 Pré-processamento *Fuzzy*

O p-ésimo padrão (exemplo) de entrada m-dimensional $\overrightarrow{X}_p = [X_{p1}, X_{p2}, \ldots, X_{pm}]$ será substituído por um vetor $3m$-dimensional:

$$\overrightarrow{O}_p = [baixo(X_{p1}), médio(X_{p1}), alto(X_{p1}), \ldots, baixo(X_{pm}), médio(X_{pm}), alto(X_{pm})] \quad (2.1)$$

em que, $baixo(X_{p1}), médio(X_{p1})$, e $alto(X_{p1})$ indicam os graus de pertinência do valor X_{p1} do primeiro atributo do p-ésimo padrão aos conjuntos *fuzzy baixo, médio* e *alto*, respectivamente.

Oliveira [Oli06] transformou a função de pertinência μ (que possui imagem em [0,1]), usada por Mitra et al. [MP94], a fim de permitir graus de pertinência no intervalo $[-1, 1]$. Após a modificação, μ é definida como,

$$\mu(X) = \begin{cases} 2(2(1 - \frac{|X-c|}{\lambda})^2) - 1 & \text{se } \frac{\lambda}{2} \leq |X - c| \leq \lambda \\ 2(1 - 2(\frac{|X-c|}{\lambda})^2) - 1 & \text{se } 0 \leq |X - c| \leq \frac{\lambda}{2} \\ -1 & \text{caso contrário} \end{cases} \tag{2.2}$$

em que, $\lambda > 0$ é o raio da função μ e c é o seu centro.

Os parâmetros c e λ são calculados em função dos valores máximo e mínimo do atributo X no conjunto de treinamento. Para cada um dos três conjuntos *fuzzy*: *baixo*, *médio* e *alto*, definem-se:

$$\begin{aligned} \lambda_{médio(X)} &= \frac{1}{2}(X_{max} - X_{min}) \\ c_{médio} &= X_{min} + \lambda_{médio(X)} \\ \lambda_{baixo(X)} &= \frac{1}{s}(c_{médio(X)} - X_{min}) \\ c_{baixo(X)} &= c_{médio(X)} - \frac{1}{2}\lambda_{baixo(X)} \\ \lambda_{alto(X)} &= \frac{1}{s}(X_{max} - c_{médio(X)}) \\ c_{alto(X)} &= c_{médio(X)} + \frac{1}{2}\lambda_{alto(X)} \end{aligned}$$

em que, $0 < s \leq 1$ é o parâmetro que controla a extensão da sobreposição das três funções.

Em resumo, um exemplo de entrada m-dimensional com componentes reais é fuzzificado e passa a ser um vetor $3m$-dimensional. Cada componente do vetor terá graus de pertinência aos conjuntos *fuzzy baixo*, *médio* e *alto*. Este vetor fuzzificado constitui a entrada para o núcleo do SNFDI.

2.5.2 T-norma e T-conorma Diferenciáveis e Interativas

A diferenciabilidade é uma característica importante em sistemas neuro-*fuzzy* pois permite a aplicação direta dos algoritmos de treinamento baseados no gradiente descendente como: o algoritmo de retropropagação.

Por outro lado, uma norma é interativa quando o resultado da sua aplicação sobre graus de pertinência depende dos valores de todos os seus argumentos. A escolha de uma t-norma interativa deverá ser favorecida, se houver interatividade entre os conjuntos *fuzzy* como visto no exemplo a seguir. A semântica dos conjuntos *fuzzy caro* e *rápido*, usados para caracterizar um veículo, sugere que estes conjuntos interagem. Assim, o resultado de

uma operação **e** deve incorporar o fato destes dois conjuntos interagirem, pois se espera que um veículo rápido seja também caro.

Zanusso [Zan97] denota a t-norma e t-conorma diferenciáveis e interativas por C e D e as define como,

$$C(x_1, x_2, \ldots, x_k) = G_3(G_3(x_1) + G_3(x_2) + \ldots + G_3(x_k)), \tag{2.3}$$

$$D(x_1, x_2, \ldots, x_k) = G_1(G_2(x_1) + G_2(x_2) + \ldots + G_2(x_k)), \tag{2.4}$$

respectivamente, onde $G_1(x)$, $G_2(x)$ e $G_3(x)$, denominados geradores [Zan97], são definidos como,

$$\begin{aligned} G_1(x) &= \frac{x-1}{x+1}, \\ G_2(x) &= \frac{1+x}{1-x}, \\ G_3(x) &= \frac{1-x}{1+x}. \end{aligned}$$

Na Seção 2.5.4, ver-se-á como os operadores de conjunção e disjunção, aqui definidos, intervém no cálculo do processamento nos neurônios E e OU.

2.5.3 Arquitetura da RNA do SNFDI

O SNFDI possui um núcleo que tem a arquitetura de uma RNA *feedfordward* [2] constituída por três camadas, a saber: a camada de entrada, a camada intermediária ou camada E e a camada de saída ou camada OU.

Serão usadas as letras l, i, e j para representar os neurônios da camada de entrada, da camada E e da camada OU, respectivamente.

Por outro lado, igual aos neurônios sensoriais de uma rede PMC, os $n1$ neurônios de entrada da RNA, também não realizam nenhum cálculo; isto significa que a saída de um neurônio l é igual a sua entrada. Na RNA, as entradas representam graus de pertinência[3] a conceitos (conjuntos) *fuzzy* definidos para cada atributo. A saída do l-ésimo neurônio é representada por O_{pl}, onde p se refere ao p-ésimo padrão de treinamento. No caso dos $n2$ neurônios da camada E e dos $n3$ neurônios da camada OU, as suas saídas são representadas por O_{pi} e O_{pj}, respectivamente.

Usar-se-á w_{il} e w_{ji} para representar as conexões dos neurônios de entrada com os neurônios E e dos neurônios E com os neurônios OU, respectivamente.

Na Figura 2.2 é ilustrada a arquitetura do núcleo do SNFDI descrita no parágrafo anterior.

[2] Tipo de arquitetura em que as conexões da RNA se projetam da camada de entrada para a camada de saída, i.e., a saída do neurônio de uma determinada camada não pode ser usada como entrada de um neurônio de uma camada anterior [Hay98] [BdCL00].

[3] Esses graus de pertinência são determinados na etapa de pré-processamento descrita na Seção 2.5.1.

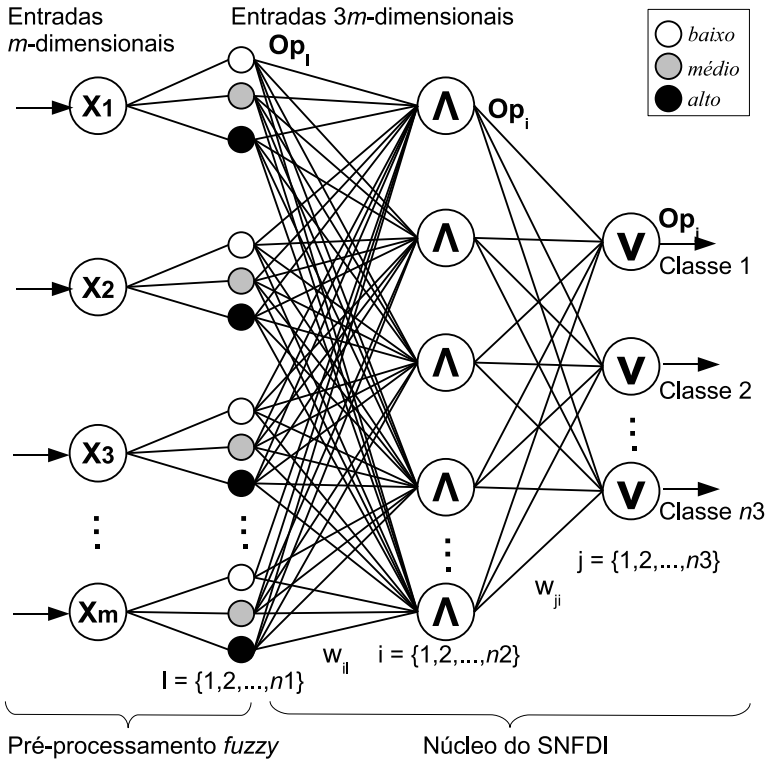


Figura 2.2: Arquitetura do núcleo do SNFDI.

2.5.4 Processamento na Camada E e na Camada OU

A função de ativação usada depende da camada na qual um neurônio se encontra. Para os neurônios da camada E, é usada a seguinte função de ativação:

$$O_{pi} = net_{pi} = net_{pi} = \bigwedge_{l=1}^{n1}(O_{pl} \vee w_{il}). \tag{2.5}$$

A t-norma e a t-conorma definidas na Seção 2.5.2 são usadas aqui. Fazendo $\wedge = C$ e $\vee = D$ nas equações 2.3 e 2.4, obtém-se que,

$$\begin{aligned} O_{pi} &= \overset{n1}{\underset{l=1}{C}}(D(O_{pl}, w_{il})) \\ &= C(D(O_{p1}, w_{i1}), \ldots, D(O_{pl}, w_{il}), \ldots, D(O_{pn1}, w_{in1})), \end{aligned} \tag{2.6}$$

para $i = 1, \ldots, n2$.

Por outro lado, para os neurônios da camada OU, a função de ativação usada é,

$$O_{pj} = net_{pj} = \bigvee_{i=1}^{n2}(O_{pi} \wedge w_{ji}). \tag{2.7}$$

Da mesma forma, fazendo $\wedge = C$ e $\vee = D$ nas equações 2.3 e 2.4, obtém-se que,

$$\begin{aligned} O_{pj} &= \underset{i=1}{\overset{n2}{D}}(C(O_{pi}, w_{ji})) \\ &= D(C(O_{p1}, w_{j1}), \ldots, C(O_{pi}, w_{ji}), \ldots, C(O_{pn2}, w_{jn2})), \end{aligned} \tag{2.8}$$

para $j = 1, \ldots, n3$.

2.5.5 Erro e a sua Derivada

O objetivo do algoritmo de treinamento é minimizar a distância entre a saída obtida pelo SNFDI e a saída desejada de todos os padrões usados para treinar a rede. Essa distância, também conhecida como *erro instantâneo* é definida como,

$$E_p = \frac{1}{2}\sum_{j=1}^{n3}(T_{pj} - O_{pj})^2, \tag{2.9}$$

em que, O_{pj} representa a saída do neurônio j da camada OU e T_{pj}, a saída desejada para o p-ésimo padrão.

A função de custo utilizada para medir o desempenho do aprendizado, é o Erro Quadrático Médio (EQM). Este é definido como,

$$EQM = \frac{1}{N}\sum_{p=1}^{N} E_p. \tag{2.10}$$

em que, N é o número de padrões de treinamento.

No SNFDI, é avaliado o EQM obtido em cada época; ou seja, depois da apresentação completa de todos os padrões de treinamento. Durante o processo de aprendizado, os pesos do SNFDI são ajustados sistematicamente visando minimizar o EQM. Para isso ser feito, é necessário calcular as derivadas parciais, $\frac{\partial E_p}{\partial w_{ji}}$ e $\frac{\partial E_p}{\partial w_{il}}$.

Para encontrar as derivadas do erro é necessário derivar a t-norma e a t-conorma, para uma constante a e uma função real qualquer, $\varphi(x)$,

$$\frac{dD(\varphi(x), a)}{dx} = \varphi'(x)\frac{(4a^2 - 8a + 4)}{[(3-a) - \varphi(x)(1+a)]^2} \tag{2.11}$$

$$\frac{dC(\varphi(x), a)}{dx} = \varphi'(x)\frac{(4a^2 + 8a + 4)}{[(3+a) + \varphi(x)(1-a)]^2} \tag{2.12}$$

Por meio de derivações sucessivas, chega-se as seguintes expressões para as derivadas,

$$\frac{\partial E_p}{\partial w_{ji}} = (T_{pj} - O_{pj}) \times \left[\frac{[4a_i^2 - 8a_i + 4]}{[(3-a_i) - C(O_{pi}, w_{ji})(1+a_i)]^2}\right] \times \left[\frac{(4O_{pi}^2 + 8O_{pi} + 4)}{[(3+O_{pi}) + w_{ji}(1-O_{pi})]^2}\right] \tag{2.13}$$

e

$$\frac{\partial E_p}{\partial w_{il}} = \left[\frac{[4O_{pl}^2 - 8O_{pl} + 4]^2}{[(3 - O_{pl}) - w_{il}(1 + O_{pl})]^4}\right] \times \left[\frac{[4a_l^2 + 8a_l + 4]}{[(3 + a_l) + D(O_{pl}, w_{il})(1 - a_l))]^2}\right] \times \sum_{j=1}^{n3}\left[(T_{pj} - O_{pj})\frac{[4a_i^2 - 8a_i + 4]}{[(3 - a_i) - C(O_{pi}, w_{ji})(1 + a_i))]^2} \frac{[4w_{ji}^2 - 8w_{ji} + 4]}{[(3 - w_{ji}) - O_{pi}(1 + w_{ji})]^2}\right], \tag{2.14}$$

em que,

$$a_i = \underset{i' \neq i}{\overset{n2}{D}}(C(O_{pi'}, w_{ji'})), \tag{2.15}$$

$$a_l = \underset{l' \neq l}{\overset{n1}{C}}(D(O_{pl'}, w_{il'})). \tag{2.16}$$

As Equações 2.13 e 2.14 são utilizadas, posteriormente, para ajustar os pesos das conexões como será visto na seção a seguir.

2.5.6 Atualização dos Pesos

Os pesos do SNFDI são atualizados após a apresentação de cada padrão, num modo de treinamento conhecido como *treinamento padrão por padrão* [Hay98]. Baseadas no cálculo do gradiente descendente, as equações de ajuste dos pesos usadas a cada iteração do algoritmo são,

$$w_{il}(t + 1) = w_{il}(t) + \Delta w_{il}(t) \tag{2.17}$$

e

$$w_{ji}(t + 1) = w_{ji}(t) + \Delta w_{ji}(t), \tag{2.18}$$

em que,

$$\Delta w_{il}(t) = \eta\frac{\partial E_p}{\partial w_{il}} + \alpha(w_{il}(t) - w_{il}(t - 1)) \tag{2.19}$$

e

$$\Delta w_{ji}(t) = \eta\frac{\partial E_p}{\partial w_{ji}} + \alpha(w_{ji}(t) - w_{ji}(t - 1)), \tag{2.20}$$

sendo η a taxa de aprendizagem e α, o momento.

É importante mencionar que após a atualização dos pesos, é realizado um procedimento de truncagem, visando garantir que os valores dos mesmos pertençam ao intervalo $[-1, 1]$. Isso é necessário pois estes valores são argumentos para as funções t-norma e t-conorma. Uma descrição completa deste procedimento pode ser encontrada em [Oli06].

2.5.7 Extração de Regras

O poder oferecido pelas RNAs esbarra na sua incapacidade de explicar, de uma forma compreensível, suas decisões. Este problema é fator de motivação para as várias pesquisas relacionadas ao desenvolvimento de técnicas para a extração de regras a partir de RNAs treinadas. Essas técnicas têm a finalidade de conferir às RNAs uma certa capacidade de explanação. Alguns domínios de aplicação, e.g., diagnóstico médico, demandam uma justificativa de como uma RNA produziu uma determinada conclusão. A explanação torna a decisão tomada por uma RNA mais aceitável para os usuários. Esta explanação, na forma de regras, serve para justificar uma saída da RNA em resposta a um dado exemplo de entrada ou mesmo subsidiar algum conhecimento sobre as propriedades do conjunto de treinamento usado. Existem outras razões que tornam a extração de regras a partir de RNAs uma tarefa importante, entre as quais a exploração dos dados, o melhoramento da generalização e a integração com os sistemas simbólicos [Oli06].

O algoritmo *backtracking* para a geração de regras foi proposto por Mitra et al. [MP94] e fornece uma justificativa ao usuário, escrita na forma de regras **se-então**, para a classificação de um determinado exemplo numa classe. Além disso, se o SNFDI tem uma boa taxa de acerto e as regras geradas são representativas da população dos dados, então pode ser usado como um classificador para um determinado domínio. Se esse for o caso, para cada entrada do conjunto de treinamento o *backtracking* gera uma regra. A capacidade de representação das regras assim obtidas é depois determinada usando medidas de avaliação de regras como a confiança, confiança negativa, suporte e novidade [Oli06].

2.6 Algoritmos Genéticos

A computação evolutiva trata de sistemas para a resolução de problemas que utilizam modelos computacionais baseados na teoria da evolução natural. Nesta área podem ser distinguidas três categorias: algoritmos genéticos, estratégias de evolução e programação genética [LCB05]. Somente, expor-se-ão os conceitos de AGs, por ser a técnica mais utilizada na computação evolutiva, e por ter sido aplicada neste trabalho.

Os Algoritmos Genéticos (AGs) foram inicialmente introduzidos por Holland [Hol75], mas popularizados por Goldberg [Gol89]. A idéia básica dos AGs têm origem na teoria da evolução das espécies, proposta por Charles Darwin. Darwin afirmou que a evolução das populações naturais, ao longo de várias gerações, ocorre de acordo com os princípios da seleção natural e sobrevivência dos indivíduos mais aptos. Assim, Goldberg define os AGs como algoritmos de otimização e busca, baseados nos mecanismos de seleção natural.

Problemas para os quais os métodos convencionais de otimização não fornecem bons resultados, constituem a maior área de aplicação dos AGs [BBM93]. Assim, os AGs têm oferecido ótimos resultados em problemas como, o agendamento de horários [BL06] [UCP00], a alocação de tarefas em sistemas multiprocessadores [TG96], o controle de irrigação [SS96], o controle de movimentos de robots [HO06], o controle de sistemas dinâmicos [Gol89], entre outros.

Dada a inspirarão biológica dos AGs, é usada a terminologia biológica para definir seus conceitos. Assim, um *cromossomo* é uma solução do problema a ser otimizado (maximizado ou minimizado). Por outro lado, para avaliar quão boas são as soluções produzidas por um AG é usada uma função chamada *função de aptidão*. O valor fornecido por esta função é denominado *índice de aptidão* ou *fitness*. Um cromossomo e seu índice de aptidão constituem um *indivíduo*. Um conjunto de indivíduos avaliados na mesma iteração formam uma *população* [LF99].

É importante distingüir a *função de aptidão* da *função objetivo*. Como mencionado no parágrafo anterior, a função de aptidão se refere à função que permite avaliar as soluções produzidas pelo algoritmo; enquanto que, a função objetivo se refere à função que se deseja otimizar. Geralmente, a função de aptidão é determinada com base na função objetivo podendo, inclusive, ser a mesma [eJES03] [LF99].

Por outro lado, um cromossomo é formado por elementos, chamados *genes*, que determinam as características de um indivíduo. Geralmente, um gene é um parâmetro codificado da função objetivo. O *genótipo* representa toda a informação armazenada nos cromossomos e permite descrever um indivíduo no nível de genes. O *fenótipo* descreve a aparência externa de um indivíduo a qual é construída usando a informação do genótipo; i.e., o fenótipo é a informação contida no cromossomo, decodificada [Rot06] [LF99]. Por exemplo, dado um cromossomo no qual estão codificadas as porcentagens da composição química de um medicamento, o fenótipo seria o medicamento resultante da mistura da composição codificada.

Dos conceitos dados anteriormente, nota-se que é possível estabelecer uma analogia entre o termo otimização e evolução, conjunto de possíveis soluções e populações naturais, e possível solução de um determinado problema e indivíduo. Outro aspecto a considerar, é a forma em que as informações contidas no cromossomo de um indivíduo são codificadas. Isso dá lugar ao tópico da representação de um indivíduo, que será abordado a seguir.

2.6.1 Representação dos indivíduos de um AG

Computacionalmente, um cromossomo é representado por uma estrutura de dados na qual cada parâmetro da função objetivo é codificado. Tradicionalmente, a estrutura de dados utilizada é um vetor, e a codificação é a binária (*representação binária*). Neste tipo de representação, os parâmetros da função objetivo são codificados como cadeias de bits concatenadas uma depois da outra, formando um vector binário. A representação binária tem sido amplamente utilizada, pois além de ser simples de analisar teoricamente, é fácil de se operar sobre cadeias de bits [LF99].

Por outro lado, existem problemas com um grande número de parâmetros de tipo contínuo. Se a codificação binária fosse usada nesses casos, seriam gerados cromossomos que produziriam um alto custo computacional. Além disso, quando parâmetros reais estão envolvidos, os operadores genéticos[4] não produzem os mesmos efeitos se forem aplicados aos

[4]Operadores utilizados para modificar indivíduos com o intuito de transformar, sucessivamente, uma população em outra até a obtenção de um resultado satisfatório [BdCL00].

bits mais à direita e não aos mais à esquerda. Este fato levou muitos pesquisadores a buscarem outras formas de representação, como a *representação real*, na qual os parâmetros da função objetivo são simplesmente números reais que ocupam uma posição dentro de um vetor [LF99].

Rothlauf [Rot06] faz uma taxonomia dos tipos de representações utilizadas na computação evolutiva, mencionando além das expostas, a *representação inteira* e a *misturada*. Nesta última, para cada parâmetro da função objetivo, é utilizada uma representação distinta. Finalmente, Rothlauf [Rot06] conclui que não existe uma representação melhor que outra, mas apenas representações mais adequadas para a solução de um determinado problema.

Por outro lado, no processo de evolução nem todos os indivíduos podem transmitir os seus genes às gerações posteriores, esse direito é limitado por algum *método de seleção*, como será visto a seguir.

2.6.2 Métodos de Seleção

Os AGs buscam os indivíduos mais aptos mediante um processo iterativo que imita a evolução das espécies. Nesse processo é realizado uma seleção, baseada em algum critério de aptidão, que determina os indivíduos de uma população que podem se reproduzir. A seleção parte do princípio que quanto maior o índice de aptidão do indivíduo, maior a probabilidade de ser selecionado; i.e., o propósito da seleção é permitir que os melhores indivíduos sejam os progenitores da geração seguinte [eJES03] [LF99] [BdCL00].

Entre os métodos de seleção mais utilizados, encontra-se o *método da roleta*. Este método seleciona os indivíduos usando a idéia de uma roleta, a qual é girada tantas vezes quanto indivíduos existam na população. Cada indivíduo ocupa uma área na roleta proporcional a seu índice de aptidão. Assim, os indivíduos com maior índice possuem áreas maiores; e portanto, têm maior probabilidade de serem selecionados.

Outro método de seleção é o *método do torneio*. Este método começa escolhendo um número z de participantes do torneio. Geralmente, é usado $z = 3$. Posteriormente, z indivíduos, com a mesma probabilidade, são escolhidos aleatoriamente para competir entre sim. Dos z indivíduos, é selecionado aquele com maior índice de aptidão. O torneio é repetido até completar o tamanho da população [LCB05] [BdCL00].

Os indivíduos selecionados são pareados para posteriormente se criar outros indivíduos, usando mecanismos de manipulação genética. Estes mecanismos, por sua vez, são executados utilizando *operadores genéticos*, os quais serão descritos a seguir.

2.6.3 Operadores Genéticos

O propósito dos operadores genéticos é criar novos indivíduos a partir de indivíduos mais velhos que possuem um alto índice de aptidão [eJES03]. Ao longo do processo de evolução, os operadores genéticos são iterativamente aplicados sobre indivíduos de uma população para passar de uma geração a outra, numa busca que se prolonga até a obtenção de um

indivíduo adaptado o suficiente (i.e., um resultado satisfatório) [BdCL00]. Os principais operadores genéticos são: *cruzamento* e *mutação*.

Mediante o operador de cruzamento é possível recombinar as características genéticas de dois progenitores permitindo que estas sejam herdadas pelas próximas gerações [eJES03] [BdCL00]. Existem diversos tipos de cruzamento, sendo os mais comuns o *cruzamento um-ponto* e o *cruzamento multipontos*. No cruzamento um-ponto, apenas um ponto de cruzamento é escolhido com probabilidade uniforme sobre o comprimento do cromossomo. A partir desse ponto são permutados o conjunto de genes restantes dos dois progenitores, obtendo-se como resultado dois cromossomos *filhos*. O cruzamento multipontos é uma generalização do tipo anterior e usa mais de um ponto de cruzamento para permutar os genes de um cromossomo [BdCL00].

Por outro lado, o operador de mutação altera um ou mais genes de um cromossomo, produzindo um indivíduo que têm pouca informação genética do progenitor. Com isso, tenta-se garantir a diversidade genética da população e assegurar que o espaço de busca do problema em questão, seja explorado em diversas regiões [eJES03] [SP94] [BdCL00]. No operador mutação tradicional, a alteração consiste em inverter o valor de um bit do vetor (de 0 para 1 ou vice-versa), de acordo com uma determinada probabilidade associada a cada um dos bits. A probabilidade aplicada, conhecida também como *taxa de mutação*, geralmente varia entre 0.1% e 0.5% [BdCL00].

A descrição anterior dos operadores corresponde à representação binária; para o caso da representação real, os operadores genéticos são aplicados sobre os indivíduos de diferente maneira. Alguns destes operadores estão baseados nos operadores convencionais mais adaptados para serem utilizados com valores reais; outros estão baseados em operações aritméticas comuns; um exemplo é o operador de cruzamento média introduzido por Davis [Dav91].

A seguir será apresentado o AG básico, no qual os conceitos expostos até o momento adquirem sentido.

2.6.4 O Algoritmo Genético Básico

O AG básico segue os passos dados no algoritmo a seguir.

No algoritmo, t é a t-ésima geração e $P(t)$, a população na geração t.

O AG básico começa com uma população inicial, $P(t)$, para $t = 0$, de indivíduos gerados aleatoriamente ou com base em algum conhecimento prévio sobre o espaço de busca. Para cada cromossomo é calculado o seu índice de aptidão usando a função de aptidão do problema a ser otimizado. Posteriormente, num processo que é repetido até atingir algum critério de parada, são selecionados os indivíduos da população $P(t)$ com direito a se reproduzirem. Para isso, é usado algum método de seleção como os expostos anteriormente. Os indivíduos selecionados dão origem a uma população intermediária de indivíduos em que são aplicados os operadores genéticos de cruzamento e mutação. Deste processo de manipulação genética são criados novos indivíduos, os indivíduos filhos e os mutados, que são avaliados, usando a função de aptidão. Os indivíduos mais aptos constituem a

Algoritmo 1 AG básico

1: $t \Leftarrow 0$
2: Cria população $P(t)$
3: Avalia população $P(t)$
4: **Enquanto** Parada = **false**
5: $t \Leftarrow t + 1$
6: Seleciona $P(t)$ a partir de $P(t-1)$
7: Aplica operador de cruzamento a $P(t)$
8: Aplica operador de mutação a $P(t)$
9: Avalia população $P(t)$
10: **Fim enquanto**

população para a seguinte geração, $P(t+1)$.

Segundo Lacerda et al. [LF99], geralmente, são usados dois critérios de parada:

- O primeiro é determinado pelo número de gerações; i.e., o processo de evolução termina quando é atingido o número máximo de gerações estabelecido;
- O segundo é atingido quando se produz um estancamento na evolução da população; i.e., quando os indivíduos com melhor índice aptidão não evoluem por um determinado número de gerações.

2.6.5 Algoritmos Genéticos e suas Hibridações

Uma hibridação bastante comum na literatura, é a de AGs e sistemas *fuzzy*. Nesta hibridação os AGs têm sido usados para ajustar os valores dos parâmetros de *sistemas fuzzy* com a finalidade de obter aqueles que produzam o melhor desempenho. Por exemplo, os AGs tem sido usados para calibrar as funções de pertinência de sistemas *fuzzy* e assim criar automaticamente bases de regras *fuzzy*. Neste último caso, os AGs podem ser usados para determinar o número de elementos no antecedente na regra, e o número de regras *fuzzy* sujeitas a uma determinada restrição [Kas96]. Alguns exemplos de aplicação são os trabalhos de Ishibuchi et al. [INK00] [IY04], Hoffmann [Hof01], Lim et al. [LRG96], e Furuhasi et al. [FMNU95].

Como foi mencionado anteriormente, as RNAs têm obtido um bom desempenho em uma grande variedade de problemas reais. No entanto, esse desempenho depende de uma série de parâmetros que precisam ser definidos, entre eles, o valor dos pesos das conexões entre os neurônios, o número de camadas, o número de nós de cada camada, a taxa de aprendizado, entre outros. Além disso, o espaço de busca destes parâmetros é complexo. Por esse motivo, torna-se difícil a utilização de técnicas convencionais de otimização na determinação dos mesmos. Por esta razão, os AGs são muitas vezes considerados para otimizar este problema [LF99].

Segundo Azevedo [ABO00], os AGs têm sido utilizados junto com as RNAs de três formas principalmente:

- Na primeira, têm sido usados para treinar ou ajudar no treinamento das RNAs, determinando os pesos das conexões entre os neurônios. Os trabalhos de Kitano [Kit90b] e de Dias [Dia98] são exemplos de aplicação desta hibridação.
- Na segunda, têm sido usados para determinar a topologia das RNAs, i.e., o número de camadas, o número de neurônios de cada camada e as conexões entre eles. Alguns exemplos de aplicação foram propostos por Kitano [Kit90a], Mandischer [Man93] e Harp et al. [HSG89]
- A terceira forma é uma combinação das duas anteriores. Os trabalhos de Maniezzo [Man94] e Pujol et al. [PP97] são exemplos da terceira hibridação.

No entanto, uma desvantagem dos AGs é o custo computacional no qual incorrem. Se a isso é adicionado o custo próprio do treinamento das RNAs, tem-se que a determinação dos parâmetros de uma RNA por meio de AGs, apesar de fornecer bons resultados, possui um aspecto indesejável, o custo computacional. Isso pode ser amenizado, em parte, usando algum conhecimento que diminua o espaço de busca na fase de geração dos indivíduos da população inicial, como é feito na metodologia de inicialização proposta.

Nesta metodologia, usa-se um AG para determinar a melhor combinação de regras de dependência obtidas a partir de um conjunto de dados. O mapeamento de distintas combinações de regras à RNA de interesse (i.e., ao SNFDI) determina redes com arquiteturas particulares. Assim, o espaço de busca está restringido às regras geradas, as quais, por conterem conhecimento de um determinado domínio, determinam arquiteturas com pesos iniciais particulares que possivelmente se encontrem próximos de aqueles que minimizam o EQM. Nesse caso, o AG precisaria de um número menor de evoluções para encontrar a arquitetura e os valores finais de seus pesos que produzam uma boa generalização.

2.7 Considerações Finais

Neste capítulo foi apresentado o sistema neuro-*fuzzy* sobre o qual aplicou-se a metodologia proposta. A este, deu-se a denominação de Sistema Neuro-*Fuzzy* Diferenciável e Interativo. Também foram descritos alguns procedimentos necessários e complementares ao seu funcionamento, a saber, a fuzzificação das entradas (parte do SNFDI), e a extração de regras a partir do sistema já treinado.

Adicionalmente, foram expostos os conceitos básicos de AGs, pois estes foram utilizados na definição da metodologia, para definir as regras a serem mapeadas ao sistema em questão. Embora a teoria de *rough sets* tenha sido utilizada na metodologia proposta, seus conceitos não foram expostos neste capítulo, pois requerem de uma descrição mais minuciosa, que será feita no capítulo a seguir.

Capítulo 3

Teoria de *Rough Sets*

3.1 Considerações Iniciais

A teoria de *rough sets*, desenvolvida por Pawlak [Paw82] no início dos anos 80', é uma abordagem matemática para manipular a incerteza e imprecisão. Esta abordagem pode ser potencialmente aplicada em diversas áreas da IA, entre elas, o aprendizado de máquina, a aquisição do conhecimento, a análise de decisões, a descoberta de conhecimento, os sistemas especialistas, o reconhecimento de padrões e a mineração de dados [PJBSZ95].

A teoria de *rough sets* supõe que as informações do mundo real estão contidas num conjunto de dados na forma de uma tabela atributo-valor, i.e., uma tabela em que as linhas representam objetos (exemplos), que possuem um número de atributos (colunas da tabela), com valores relacionados a cada um desses atributos, que são os mesmos para todos os objetos, mas seus valores podem ser diferentes.

Baseando-se no fato de que o mundo real não é exato ou preciso, objetos colhidos da tabela que representa as informações de um determinado domínio, podem ser indiscerníveis. A teoria de *rough sets* motivada pela necessidade de interpretar, caracterizar, representar, e processar o não discernimento entre objetos, utiliza conceitos matemáticos simples, envolvendo conjuntos finitos, relações e classes de equivalência, a partir dos quais são definidos outros conceitos, de cuja aplicabilidade é possível obter conhecimento sobre um determinado domínio.

No contexto de sistemas inteligentes híbridos, sabendo que nenhuma teoria, abordagem ou técnica é particularmente suficiente para resolver qualquer problema, a teoria de *rough sets* tem sido combinada com outras abordagens, como a teoria *fuzzy* e conexionista. Alguns exemplos da primeira hibridação são os trabalhos de Honh et al. [HWW07] e Tsai et al. [TCC06]. Já na última hibridação, Lingras [Lin96], por exemplo, introduziu o conceito de neurônio *rough*, criando as RNA Rough.

Outra maneira de combinar ambas as teorias é inicializando uma RNA com conhecimento representado na forma de regras de dependência, sendo estas obtidas a partir um conjunto de dados (exemplos), e aplicando os conceitos da teoria de *rough sets*. Este tipo de com-

binação é vantajosa pois permite determinar aspectos da arquitetura de uma RNA, e dar uma inicialização aos pesos das suas conexões, o que contribui a melhorar o desempenho da mesma. O trabalho aqui realizado se encaixa nesta última forma de integração.

Pode-se dizer que com a metodologia proposta, o processamento massivo dos dados realizado pelo SNFDI, inicializado usando um certo conhecimento desses dados, permite codificar em seus pesos um refinamento desse conhecimento, o qual conduz a aumentar a capacidade de generalização do SNFDI. O anteriormente afirmado será verificado com a execução dos experimentos descritos na Seção 5.3.

Neste capítulo, apresentar-se-ão os conceitos da teoria de *rough sets* que são necessários para o entendimento de como as regras de dependência são geradas a partir de um conjunto de dados. Para os conceitos a serem dados, usar-se-á como base os textos de Pawlak et al. [Paw82] [Paw91] e Skowron et al.[SR92]. Nas Seções 3.2, 3.3 e 3.4 abordar-se-ão os conceitos de sistema de informação, relação de não-discernimento e reduto, respectivamente. Na Seção 3.5, serão tratados os conceitos de matriz e função de discernimento, úteis para a determinação de redutos. Na Seção 3.6 serão estudados os conceitos associados a sistemas de decisão. Também serão apresentadas algumas heurísticas para a determinação de redutos, em particular, descrever-se-á a heurística nomeada *Feature Ranking* [HLS03]. Apresentar-se-á o procedimento para gerar regras de dependência. Finalmente, na Seção 3.7 serão dados os conceitos sobre discretização e sua relação com a teoria de *rough sets*.

3.2 Sistemas de Informação

Na teoria de *rough sets* supõe-se que as informações a respeito do mundo real são dadas na forma de uma tabela chamada de Sistema de Informação (SI)[1]. Nessa tabela, as linhas estão formadas por um conjunto de objetos de um domínio específico e as colunas por um conjunto de atributos que caracterizam tais objetos. Cada objeto tem valores específicos relacionados a cada um dos atributos [Paw91]. Por exemplo, os objetos de um domínio poderiam ser, clientes de uma empresa, pacientes de um hospital, candidatos a um cargo numa empresa, etc.; e os atributos para, por exemplo, os candidatos a um cargo numa empresa poderiam ser os critérios de avaliação usados para a contratação de um candidato. Portanto, um SI é semelhante a um banco de dados relacional simplificado numa tabela única.

Formalmente, define-se um SI como um par $S = (U, A)$, em que U é um conjunto de objetos não vazio, finito chamado *universo* e A é um conjunto não vazio, finito de *atributos*. Cada atributo $a \in A$, pode ser considerado como uma função, $a : U \rightarrow V_a$, em que V_a é o conjunto dos valores permitidos para o atributo a.

Para exemplificar esse conceito, utilizar-se-á o SI extraído de Pawlak et al. [KPPS99]. Este SI foi ligeiramente modificado, adicionando linhas e modificando alguns dos valores

[1] Outras denominações são, *tabela de informação* e *sistema de representação do conhecimento*. Na teoria de *rough sets*, aceita-se que as informações de um determinado domínio estejam completamente representadas por um SI, embora isto nem sempre ocorra em problemas reais.

de seus atributos para adaptá-lo aos conceitos a serem estudados. No SI da Tabela 3.1, são apresentadas as informações de 14 candidatos a um determinado cargo numa empresa. São considerados 4 critérios de avaliação, a saber, o tipo de diploma, a experiência de trabalho, o conhecimento do idioma francês e as referências apresentadas pelo candidato.

Para o exemplo em estudo, $U = \{u_1, \ldots, u_{14}\}$, e $A = \{$*diploma, experiência, francês, referência*$\} = \{i, e, f, r\}$. Dado que, $V_{diploma} = \{$MBA, MCE, MSc$\}$ e $V_{francês} = \{$sim, não$\}$, então $diploma(u_1) =$ MBA e $francês(u_1) =$ sim.

	diploma	*experiência*	*francês*	*referência*
u_1	MBA	média	sim	excelente
u_2	MBA	pouca	sim	neutra
u_3	MCE	pouca	sim	boa
u_4	MSc	muita	sim	neutra
u_5	MSc	média	sim	neutra
u_6	MSc	muita	sim	excelente
u_7	MBA	muita	não	boa
u_8	MCE	pouca	não	excelente
u_9	MSc	muita	sim	neutra
u_{10}	MBA	média	sim	excelente
u_{11}	MSc	muita	sim	neutra
u_{12}	MSc	muita	sim	excelente
u_{13}	MSc	muita	sim	neutra
u_{14}	MSc	muita	sim	excelente

Tabela 3.1: SI com objetos redundantes.

Geralmente, os SIs são redundantes por duas razões:

a. O SI contém o mesmo objeto ou objetos indiscerníveis representados várias vezes.

b. O SI contém atributos que podem ser supérfluos, i.e., atributos que se excluídos não provocam perdas de informações.

Na Tabela 3.1, é evidente a redundância descrita em **(a)** pois, os objetos u_1, u_{10} são indistinguíveis entre si. O mesmo acontece para os objetos, u_4, u_9, u_{11}, u_{13} e para os objetos u_6, u_{12}, u_{14}.

O conceito de relação de não-discernimento, estudado na seguinte seção, fornece o fundamento teórico básico para entender como é possível contornar ambas as formas de redundância. No primeiro caso, isso é feito mediante a determinação de classes de equivalência (ver a Seção 3.3) e no segundo, através da determinação de redutos (ver as Seções 3.4 e 3.6.3).

3.3 Relação de Não-discernimento

Uma relação de não-discernimento[2] permite determinar os objetos de um universo que são indiscerníveis considerando determinados atributos.

Dado um SI, $S = (U, A)$, existe para cada subconjunto de atributos $B \subseteq A$ uma relação binária de equivalência[3], $IND_S(B)$, chamada *relação $B-$não-discernimento*[4], definida como,

$$IND_S(B) = \{(u, y) \in U^2 : \forall a \in B, a(u) = a(y)\} \tag{3.1}$$

ou

$$IND_S(B) = \bigcap_{a \in B} IND_S(\{a\}). \tag{3.2}$$

No que segue, será usada $IND(B)$ em vez de $IND_S(B)$, sempre que S (ou seja, o SI que está sendo referenciado) seja evidente.

Diz-se que os elementos u e y que satisfazem a relação $IND(B)$ são *B-indiscerníveis*. Isto significa que não é possível distingüir u de y levando em consideração apenas os atributos de B.

Denota-se por $[u]_B$, ao conjunto de objetos de U que são indiscerníveis considerando os atributos em B que incluem a u. Este conjunto é chamado *classe de equivalência de u em relação a B*, e é definido como,

$$[u]_B = \{y \in U : uIND(B)y\}. \tag{3.3}$$

O conjunto das classes de equivalência em relação a B é conhecido como *conjunto quociente da relação $IND(B)$*. Esse é definido como,

$$U/IND(B) = \{[u]_B : u \in U\}. \tag{3.4}$$

Para o SI da Tabela 3.1 e considerando o conjunto de atributos $B = \{$*francês, referência*$\}$, tem-se que u_1, u_6, u_{10}, u_{12} e u_{14} são B-indiscerníveis. A classe de equivalência de u_1 em relação a B é

$$[u_1]_B = \{u_1, u_6, u_{10}, u_{12}, u_{14}\},$$

e o conjunto quociente da relação $IND(B)$ é

$$U/IND(B) = \{\{u_1, u_6, u_{10}, u_{12}, u_{14}\}, \{u_2, u_4, u_5, u_9, u_{11}, u_{13}\}, \{u_3\}, \{u_7\}, \{u_8\}\}.$$

É importante ressaltar que a formação de classes de equivalência permite eliminar informações redundantes. Por exemplo, considerando $B = A$, obtém-se as seguintes classes de equivalência,

$$U/IND(B) = \{\{u_1, u_{10}\}, \{u_2\}, \{u_3\}, \{u_4, u_9, u_{11}, u_{13}\}, \{u_5\}, \{u_6, u_{12}, u_{14}\}, \{u_7\}, \{u_8\}\}.$$

[2] *Indiscernibility relation.*

[3] Uma relação de equivalência R satisfaz as propriedades de reflexividade (xRx), simetria ($xRy \rightarrow yRx$) e transitividade ($xRy \wedge yRz \rightarrow xRz$).

[4] *B-indiscernibility relation.*

Nota-se que os objetos u_1 e u_{10} são indistinguíveis, pois possuem a mesma informação para todos seus atributos; em conseqüência, a presença de ambos no SI é redundante. Considerando este fato, pode-se intuir que bastaria associar cada classe de equivalência com uma descrição única, e fazer referência a essa descrição, em vez de fazer referência à descrição de cada um dos objetos da classe de equivalência tratada. Esta associação pode ser feita escolhendo um representante para cada classe de equivalência. Para indicar que um objeto é o representante de uma determinada classe, costuma-se colocá-lo entre colchetes.

Encontrando $U/IND(B)$, $B = A$, o SI da Tabela 3.1 pode ser reduzido ao SI da Tabela 3.2. Os representantes das classes de equivalência $\{u_1, u_{10}\}$, $\{u_2\}$, $\{u_3\}$, $\{u_4, u_9, u_{11}, u_{13}\}$, $\{u_5\}$, $\{u_6, u_{12}, u_{14}\}$, $\{u_7\}$ e $\{u_8\}$ são $[u_1]$, $[u_2]$, $[u_3]$, $[u_4]$, $[u_5]$, $[u_6]$, $[u_7]$ e $[u_8]$, respectivamente. No que segue, não se usará [.] para indicar que um objeto é representante de uma classe mas apenas o objeto mesmo. Por exemplo, u_1 é o representante da classe de equivalência formada pelos objetos u_1 e u_{10}.

	diploma	*experiência*	*francês*	*referência*
u_1	MBA	média	sim	excelente
u_2	MBA	pouca	sim	neutra
u_3	MCE	pouca	sim	boa
u_4	MSc	muita	sim	neutra
u_5	MSc	média	sim	neutra
u_6	MSc	muita	sim	excelente
u_7	MBA	muita	não	boa
u_8	MCE	pouca	não	excelente

Tabela 3.2: SI sem objetos redundantes.

Como mencionado na Seção 3.2, outra forma de redundância é a produzida por atributos supérfluos. Na seguinte seção, ver-se-á como esta pode ser eliminada mediante o cálculo de redutos.

3.4 Redutos

Tem-se visto até o momento, que objetos que são indiscerníveis, considerando determinados atributos, podem ser obtidos através de relações de equivalência a partir das quais é possível definir classes de equivalência, que são partições do universo. Esta é uma forma de evitar a redundância, pois somente é necessário um elemento de uma classe de equivalência e não todos eles para identificá-la.

Como dito, outra forma de evitar a redundância é eliminar os atributos supérfluos e conservar os relevantes. O problema deste tipo de redução pode ser enunciado como, dado um SI, $S = (U, A)$, determinar um conjunto $B \subseteq A$, no qual nenhum de seus atributos possa ser omitido sem que ele (i.e., B) gere diferentes partições das que gerava

inicialmente, e que ao mesmo tempo, esse conjunto (i.e., B) induza as mesmas partições que A. O conjunto resposta é chamado *reduto* de A em S.

Antes de introduzir a definição formal de reduto, serão definidos os conceitos de *indispensabilidade* e *independência* de atributos.

Dado um SI, $S = (U, A)$, um atributo $b \in B \subseteq A$ é *dispensável* em B se

$$IND(B) = IND(B - \{b\});$$

em outro caso b é chamado *indispensável* em B.

Dado o conjunto $B \subseteq A$, B é chamado *independente* em S se cada atributo de B é indispensável em B; caso contrário B é chamado *dependente* em S.

Um conjunto $B \subseteq A$ é chamado *reduto* de A em S, se B é independente em S e

$$IND(B) = IND(A).$$

O conjunto de todos os redutos de A em S é denotado por $RED_S(A)$.

No SI da Tabela 3.2, pode-se verificar que $B = \{experiência, referência\}$ é um reduto de A em S, pois $U/IND(A)$[5]$= U/IND(B) = \{\{u_1\}, \{u_2\}, \{u_3\}, \{u_4\}, \{u_5\}, \{u_6\}, \{u_7\}, \{u_8\}\}$, e B é independente em S; i.e., $U/IND(B - \{experiência\}) = \{\{u_1, u_6, u_8\}, \{u_2, u_4, u_5\}, \{u_3, u_7\}\} \neq U/IND(B) \neq U/IND(B - \{referência\}) = \{\{u_1, u_5\}, \{u_2, u_3, u_8\}, \{u_4, u_6, u_7\}\}$. Pode-se afirmar que usando os atributos *experiência* e *referência* se produzem as mesmas partições (classes de equivalência) que usando todos os atributos de A e portanto, pode-se prescindir de *diploma* e *francês* sem provocar perdas de informações.

Os redutos de um SI podem ser obtidos com base na *função de discernimento* do SI; esta, por sua vez, é calculada usando a sua *matriz de discernimento*. Ambos os conceitos serão tratados na seção a seguir.

3.5 Matriz e Função de Discernimento

Skowron [SR92] propôs a idéia de *matriz de discernimento* para representar os atributos que distinguem a cada par de objetos de um universo. Esta representação é conveniente pois facilita a determinação dos redutos de um SI.

Dado um SI, $S = (U, A)$ com $A = \{a_1, \ldots, a_{|A|}\}$ e $U = \{u_1, \ldots, u_n\}$, onde $|.|$ denota a cardinalidade do conjunto argumento. A *matriz de discernimento* de S, M_S, é uma matriz de dimensão $n \times n$, onde cada entrada é um conjunto de atributos,

$$c_{ij} = \{a \in A : a(u_i) \neq a(u_j)\} \text{ para } i, j = 1, \ldots, n. \tag{3.5}$$

Naturalmente, e $c_{ij} = \emptyset$ para $i = j$, e M_S é simétrica, pelo fato de que os atributos que diferem em valor os objetos i e j, também diferem os objetos j e i.

[5]Usou-se $U/IND(.)$ em vez que $IND(.)$ pois $IND(.)$ produz um $U/IND(.)$ único.

A *função de discernimento* de S, f_S, é uma função de $|A|$ variáveis booleanas $\overline{a}_1, \ldots, \overline{a}_{|A|}$ correspondentes aos atributos $a_1, \ldots, a_{|A|}$, respectivamente. Define-se f_S como,

$$f_S(\overline{a}_1, \ldots, \overline{a}_{|A|}) = \wedge\{\vee(c_{ij}) : 1 \leq j < i \leq n, c_{ij} \neq \emptyset\}, \tag{3.6}$$

onde $\vee(c_{ij})$ é a disjunção de todas as variáveis $\overline{a}$ tal que $a \in c_{ij}$.

A partir da função de discernimento f_S é possível encontrar os redutos de A em S, calculando o conjunto de primos implicantes [6] da função de discernimento em questão. Assim, os atributos correspondentes as variáveis booleanas presentes num primo implicante conformam um reduto de A em S.

A matriz de discernimento do SI da Tabela 3.2 é apresentada na Tabela 3.3. Por exemplo, a posição (4,3) indica que os objetos u_4 e u_3 são discerníveis em relação aos atributos *diploma, experiência* e *referência*. Também, dado que M_S é simétrica, o conjunto de variáveis booleanas correspondente à posição (4,3) e igual ao da posição (3,4). Visando facilitar a sua visualização, apenas mostra-se a matriz triangular inferior de M_S.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
u_1	$\emptyset$							
u_2	e, r	$\emptyset$						
u_3	i, e, r	i, r	$\emptyset$					
u_4	i, e, r	i, e	i, e, r	$\emptyset$				
u_5	i, r	i, e	i, e, r	e	$\emptyset$			
u_6	i, e	i, e, r	i, e, r	r	e, r	$\emptyset$		
u_7	e, f, r	e, f, r	i, e, f	i, f, r	i, e, f, r	i, f, r	$\emptyset$	
u_8	i, e, f	i, r	f, r	i, e, f, r	i, e, f, r	i, e, f	i, e, r	$\emptyset$

Tabela 3.3: Matriz de discernimento do SI da Tabela 3.2.

A função de discernimento, f_S, é obtida a partir de M_S,

$$\begin{aligned} f_S = &(\overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e}) \wedge (\overline{e} \vee \overline{f} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f}) \wedge \\ &(\overline{i} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e}) \wedge (\overline{i} \vee \overline{e}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{e} \vee \overline{f} \vee \overline{r}) \wedge (\overline{i} \vee \overline{r}) \wedge \\ &(\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f}) \wedge (\overline{f} \vee \overline{r}) \wedge \\ &\overline{e} \wedge \overline{r} \wedge (\overline{i} \vee \overline{f} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f} \vee \overline{r}) \wedge \\ &(\overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f} \vee \overline{r}) \wedge \\ &(\overline{i} \vee \overline{f} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}). \end{aligned}$$

Simplificando esta expressão, utilizando teoremas, propriedades e postulados da algebra booleana, obtém-se,

[6] Do inglês *prime implicant*. Um implicante de uma função booleana f é uma conjunção de literais (variáveis booleanas ou suas negações) tal que se os valores desses literais são verdade sob uma avaliação arbitrária v de variáveis, então o valor da função f sob v também será verdade. Um primo implicante é o implicador mínimo. No contexto de *rough sets*, o interesse está apenas em implicantes de funções booleanas monotônicas, i.e., funções construídas sem negação.

$$f_S = (\overline{e} \wedge \overline{r}).$$

Do cálculo de f_S, tem-se que $RED_S(A) = \{\{$*experiência, referência*$\}\}$. Neste exemplo, somente existe um primo implicante, $(\overline{e} \wedge \overline{r})$; em conseqüência, existe apenas um reduto em S. Portanto, é possível afirmar que os atributos *experiência* e *referência* produzem as mesmas partições que A, como foi provado na seção anterior.

3.6 Sistemas de Decisão

Em muitas aplicações, entre elas, nas aplicações a que destina o SNFDI, os objetos de U possuem uma classificação conhecida, expressada por um atributo chamado de *atributo de decisão*. Os SI destas aplicações são um tipo especial de SI, conhecido como Sistema de Decisão (SD)[7].

Formalmente, um SD é um SI da forma, $S = (U, A \cup \{d\})$, onde d representa o *atributo de decisão* e A representa um conjunto de atributos chamados *atributos condicionais* ou simplesmente *condições*.

Assim, o SI da Tabela 3.2, pode ser convertido num SD adicionando um atributo de decisão (neste caso esse atributo será chamado *decisão* e será abreviado por d) que indica se um candidato foi "aceito" ou "rejeitado" pela empresa. O SD é mostrado na Tabela 3.4, cujos objetos, por conveniência, foram agrupados pelo atributo *decisão*.

	diploma	*experiência*	*francês*	*referência*	*decisão*
u_1	MBA	média	sim	excelente	aceito
u_4	MSc	muita	sim	neutra	aceito
u_6	MSc	muita	sim	excelente	aceito
u_7	MBA	muita	não	boa	aceito
u_2	MBA	pouca	sim	neutra	rejeitado
u_3	MCE	pouca	sim	boa	rejeitado
u_5	MSc	média	sim	neutra	rejeitado
u_8	MCE	pouca	não	excelente	rejeitado

Tabela 3.4: SD consistente com atributos redundantes.

Por sua vez, os SDs podem ser divididos em dois tipos. Usar-se-á um exemplo para defini-los. Nota-se que no SD da Tabela 3.4 não existem objetos que tendo as mesmas informações dadas por seus atributos condicionais pertençam a classes diferentes. A este tipo de SD se denomina *consistente* ou *determinístico*. Em contraste, no SD da Tabela 3.5, os objetos u_4 e u_5 possuem o mesmos valores para seus atributos condicionais, mas diferentes valores para seus atributos de decisão. Um SD que apresenta esta ambigüidade é denominado *inconsistente* ou *não determinístico*.

[7]Outra denominação é *tabela de decisão*.

	diploma	*experiência*	*francês*	*referência*	*decisão*
u_1	MBA	média	sim	excelente	aceito
u_4	MSc	média	sim	neutra	aceito
u_6	MSc	muita	sim	excelente	aceito
u_7	MBA	muita	não	boa	aceito
u_2	MBA	pouca	sim	neutra	rejeitado
u_3	MCE	pouca	sim	boa	rejeitado
u_5	MSc	média	sim	neutra	rejeitado
u_8	MCE	pouca	não	excelente	rejeitado

Tabela 3.5: SD inconsistente com atributos redundantes.

Nas próximas subseções serão descritos os conceitos necessários para a geração de regras de dependência, que ao serem mapeadas no SNFDI, determinam a sua arquitetura e os pesos iniciais das suas conexões. É importante esclarecer que alguns desses conceitos, como os conjuntos aproximados, e região positiva são aplicados a SIs em geral. Porém, dado que na metodologia de inicialização proposta usaram-se no contexto de SDs, optou-se por incluí-los dentro desta seção.

3.6.1 Conjuntos Aproximados

Uma relação de equivalência induz partições do universo, i.e., conjuntos disjuntos do universo. Em problemas de classificação, os conjuntos de maior interesse são aqueles que têm o mesmo valor no atributo de decisão. No entanto, definir esses conjuntos de maneira exata não é sempre simples, pois geralmente, a ambigüidade e a imprecisão estão presentes. Nesses casos, aplicando a teoria de *rough sets* é possível determiná-los aproximadamente (*roughly*), encontrando, por sua vez, dois conjuntos conhecidos como *aproximação superior* e *aproximação inferior*[8] dos conjuntos em questão. Diz-se que um conjunto que não pode ser definido de maneira exata é um *rough set*.

Dado um SD, $S = (U, A \cup \{d\})$, $B \subseteq A$ e $H \subseteq U$, a *B-aproximação superior* e *inferior de* H são definidas como,

$$\overline{B}(H) = \bigcup \{Y \in U/IND(B) : Y \cap H \neq \emptyset\} \tag{3.7}$$

e

$$\underline{B}(H) = \bigcup \{Y \in U/IND(B) : Y \subseteq H\}, \tag{3.8}$$

respectivamente, onde H é o conjunto que se quer definir, cujos objetos têm o mesmo valor para o atributo de decisão.

Por exemplo, dado o SD da Tabela 3.5, suponha-se necessário determinar os candidatos "rejeitados" (i.e., $H = \{u \in U : \textit{decisão}(u) = \text{rejeitado}\}$) usando as informações de todos os atributos condicionais, i.e., $B = A$. Pode-se observar que os objetos u_4 e u_5 possuem

[8] Outras denominações são *região superior* e *região inferior*, respectivamente.

os mesmos valores para seus atributos condicionais, mas valores diferentes para seus atributos de decisão. Este fato influi na determinação do conjunto dos "rejeitados" sendo apenas possível uma determinação aproximada. Assim, calculando-se as B-aproximações superior e inferior de H obtém-se, $\underline{B}(H) = \{u_2, u_3, u_8\}$, $\overline{B}(H) = \{u_2, u_3, u_4, u_5, u_8\}$, respectivamente. Por não ser possível definir o conjunto dos "rejeitados" de maneira exata, pode-se afirmar que esse conjunto é um *rough set*.

Intuitivamente, a B-aproximação inferior de $H \subseteq U$ é o conjunto dos objetos em U que podem ser classificados com certeza[9] como objetos de H, usando o conhecimento dado por B, i.e., $\underline{B}(H)$ está formado por objetos para os que é possível afirmar com certeza que pertencem ao conjunto H. Assim mesmo, a B-aproximação superior de $H \subseteq U$ é o conjunto dos objetos em U que, possivelmente, podem ser classificados como objetos de H, usando o conhecimento em B. Do exemplo apresentado no parágrafo anterior, pode-se afirmar que os candidatos u_2, u_3 e u_8 são classificados com certeza como candidatos "rejeitados". O conjunto de candidatos formado por u_2, u_3, u_4, u_5 e u_8 é considerado como dos possíveis candidatos "rejeitados", porque apesar da certeza da classificação de u_2, u_3 e u_8, nada pode ser afirmado sobre u_4 e u_5 considerando o conhecimento dado por B.

No contexto deste trabalho, o interesse é centrado na aproximação inferior de um conjunto, pois ele está relacionado com o cálculo da *região positiva* que será visto na seção a seguir.

3.6.2 Região Positiva

Dado um SD, $S = (U, A \cup \{d\})$, a *B-região positiva de d* ($B \subseteq A$), i.e., a região positiva de d considerando os atributos de B, é definida como,

$$POS_B(d) = \bigcup \{\underline{B}(H) : H \in U/IND(d)\}. \tag{3.9}$$

Intuitivamente, a B-região positiva de d é o conjunto dos objetos de U que podem ser classificados com certeza em alguma das classes de equivalência determinadas por d, tendo as informações dos atributos de B.

Do SD da Tabela 3.5, tem-se que a B-região positiva de d, $B = A$, é,

$$\begin{aligned} U/IND(B) =& \{\{u_1\}, \{u_2\}, \{u_3\}, \{u_4, u_5\}, \{u_6\}, \{u_7\}, \{u_8\}\}, \\ U/IND(d) =& \{\{u_1, u_4, u_6, u_7\}, \{u_2, u_3, u_5, u_8\}\}, \\ POS_B(d) =& \underline{B}(\{u_1, u_4, u_6, u_7\}) \cup \underline{B}(\{u_2, u_3, u_5, u_8\}) \\ =& \{u_1, u_6, u_7\} \cup \{u_2, u_3, u_8\} \\ =& \{u_1, u_2, u_3, u_6, u_7, u_8\}. \end{aligned}$$

Pode-se verificar que para todos os objetos em $POS_B(d)$ tem-se certeza absoluta da sua classificação. Em outras palavras, não existem objetos em $POS_B(d)$ que possuindo os mesmos valores para os atributos de B, possuam diferentes valores para o atributo

[9] Dado que a teoria *rough sets* assume que as informações de um domínio estão completamente contidas num SI, cabe o uso da palavra "certeza". Contudo, é conhecido que em problemas de classificação não é possível ter certeza absoluta da classe à qual um objeto de um domínio pertence.

de decisão. Como pode ser visto, os objetos u_4 e u_5 que introduzem ambigüidade na classificação dos candidatos "rejeitados" ou "aceitos" não se encontram em $POS_B(d)$.

É importante destacar que a aproximação inferior se refere a certeza da classificação dos objetos pertencentes a uma das classes determinadas por d. Em contraste, a certeza da região positiva de d se refere a todas as classes determinadas por d.

Neste trabalho, este conceito será usado para definir os redutos d-relativos, estudados na seção a seguir. A partir destes podem ser geradas regras de dependência, como será visto na Seção 3.6.6.

3.6.3 Redutos d-Relativos

O problema de encontrar um reduto relativo ao atributo de decisão de um SD, $S = (U, A \cup \{d\})$ pode ser enunciado como o de determinar um conjunto de atributos $B \subseteq A$ no qual a B-região positiva de d seja a mesma que a A-região positiva de d, e para o qual, por sua vez, não seja possível excluir nenhum de seus atributos (i.e., de B) sem gerar uma B-região positiva de d diferente à original. Em outras palavras, a idéia é encontrar um conjunto com apenas os atributos necessários para fazer que os objetos de U que são classificados com certeza usando todos os atributos condicionais continuem sendo os mesmos.

Em outras palavras, um reduto d-relativo é um conjunto de atributos condicionais, (com o menor número possível de atributos), que permitem discernir os objetos de uma classe determinada pelo atributo de decisão de todas as demais classes de um SD. Isso equivale a dizer que um d-reduto possui o mesmo poder de classificação que o conjunto de todos os atributos de A.

Antes de introduzir o conceito formal de reduto d-relativo, serão definidas a *dispensabilidade* e a *independência* relativa ao atributo de decisão, ou a *d-dispensabilidade* e a *d-independência*, respectivamente.

Dado um SD, $S = (U, A \cup \{d\})$, um atributo $b \in B$, $B \subseteq A$ é d-dispensável em B se,

$$POS_B(d) = POS_{B-\{b\}}(d);$$

caso contrario b é d-indispensável em B. Se cada atributo em B é d-indispensável então B é d-independente em S.

Um conjunto $B \subseteq A$ é um reduto de A relativo ao atributo d em S, i.e., um *reduto d-relativo de A* em S se, B é d-independente em S e $POS_A(d) = POS_B(d)$. O conjunto de redutos d-relativos de A em S é denotado por $RED_S^d(A)$. No restante do texto, o termo d-reduto será usado, em vez de reduto d-relativo.

Do SD da Tabela 3.5, pode-se verificar que $B = \{$*experiência, referência*$\}$ é um d-reduto, pois $POS_B(d) = POS_A(d) = \{u_1, u_2, u_3, u_6, u_7, u_8\}$ e B é independente em S; i.e., tanto *experiência* quanto *referência* são indispensáveis em B, $POS_{B-\{experiência\}}(d) = \emptyset \neq POS_B(d) \neq POS_{B-\{referência\}}(d) = \{u_2, u_3, u_5, u_6, u_7\}$. Portanto, é possível afirmar que

o conjunto de atributos, formado por *experiência* e *referência*, possui o mesmo poder de classificação que o do conjunto de todos os atributos de A.

Determinar os d-redutos de um SD é importante para a metodologia proposta, pois a partir destes são geradas regras de dependência (ver Seção 3.6.6). Os d-redutos podem ser calculados mediante sua *função de discernimento relativa a d*, a que por sua vez é calculada a partir da *matriz de discernimento relativa a d*. Ambos os conceitos serão expostos na seguinte seção.

3.6.4 Matriz e Função de Discernimento d-Relativas

Diferente de uma matriz de discernimento que mostra os atributos que distinguem cada par dos objetos de U; uma matriz de discernimento d-relativa, mostra os atributos condicionais que distinguem cada par de objetos de U com diferente atributo de decisão.

Dado um SD, $S = (U, A \cup \{d\})$, $A = \{a_1, \ldots, a_{|A|}\}$, $U = \{u_1, \ldots, u_n\}$. A matriz de discernimento d-relativa de S, denotada por M_S^d, é uma matriz simétrica de dimensão $n \times n$, onde cada componente,

$$c_{ij}^d = \{a \in A : a(u_i) \neq a(u_j)\} \text{ para } i, j = 1, \ldots, n, \tag{3.10}$$

onde se $d(u_i) = d(u_j)$ então $c_{ij}^d = \emptyset$.

A função de discernimento d-relativa de S, f_S^d, é uma função de $|A|$ variáveis booleanas $\overline{a}_1, \ldots, \overline{a}_{|A|}$ correspondentes aos atributos condicionais, $a_1, \ldots, a_{|A|}$, respectivamente. Esta é definida como,

$$f_S^d(\overline{a}_1, \ldots, \overline{a}_{|A|}) = \wedge\{\vee(c_{ij}^d) : 1 \leq j < i \leq n, c_{ij}^d \neq \emptyset\}, \tag{3.11}$$

onde $\vee(c_{ij}^d)$ é a disjunção de todas as variáveis $\overline{a}$ tal que $a \in c_{ij}^d$.

A partir da função de discernimento f_S^d é possível encontrar os d-redutos de A em S, calculando o conjunto de primos implicantes da função de discernimento em questão. Assim, os atributos correspondentes as variáveis booleanas presentes num primo implicante conformam um d-reduto de A em S. No restante do texto, em vez de usar o termo matriz de discernimento d-relativa, usar-se-á o termo matriz d-relativa.

A M_S^d do SD-exemplo é apresentada na Tabela 3.6. Nota-se que M_S^d mostra apenas os atributos condicionais que discernem cada par de objetos de U pertencentes a classes diferentes.

Baseando-se em M_S^d, calcula-se sua f_S^d obtendo,

$$\begin{aligned} f_S^d = &(\overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f}) \wedge \\ &(\overline{i} \vee \overline{e}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f} \vee \overline{r}) \wedge \\ &(\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{r}) \wedge (\overline{e} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f}) \wedge \\ &(\overline{e} \vee \overline{f} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f}) \wedge (\overline{i} \vee \overline{e} \vee \overline{f} \vee \overline{r}) \wedge (\overline{i} \vee \overline{e} \vee (\overline{r}). \end{aligned}$$

	u_1	u_4	u_6	u_7	u_2	u_3	u_5	u_8
u_1	$\emptyset$							
u_4	$\emptyset$	$\emptyset$						
u_6	$\emptyset$	$\emptyset$	$\emptyset$					
u_7	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$				
u_2	e, r	i, e	i, e, r	e, f, r	$\emptyset$			
u_3	i, e, r	i, e, r	i, e, r	i, e, f	$\emptyset$	$\emptyset$		
u_5	i, r	$\emptyset$	e, r	i, e, f, r	$\emptyset$	$\emptyset$	$\emptyset$	
u_8	i, e, f	i, e, f, r	i, e, f	i, e, r	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Tabela 3.6: Matriz de discernimento relativa do SD-exemplo.

Reduzindo a expressão, obtém-se,

$$f_S^d = (\bar{i} \wedge \bar{e}) \vee (\bar{e} \wedge \bar{r}) \vee (\bar{i} \wedge \bar{r}).$$

Do cálculo de f_S^d, obtém-se três primos implicantes; conseqüentemente, três d-redutos, $RED_S^d(A) = \{\{$*diploma, experiência*$\}$, $\{$*experiência, referência*$\}$,$\{$*diploma, referência*$\}\}$. Pode-se verificar que usando qualquer dos d-redutos calculados, é possível classificar com certeza os objetos u_1, u_2, u_3, u_4, u_6, u_7 e u_8 nas classes determinadas por d. Estes são os mesmos objetos que os atributos de A classificam com certeza.

3.6.5 Heurísticas para a Determinação de d-Redutos

Um d-reduto, como visto neste capítulo, é um conjunto mínimo de atributos que preservam o poder de classificação de um conjunto inicial de dados. Os d-redutos de um SD podem ser encontrados, construindo e depois reduzindo a função de discernimento do SD em questão. Contudo, tem sido demonstrado que o problema de encontrar d-redutos é NP-difícil [PS99].

Para contornar este problema várias heurísticas têm sido propostas. Por exemplo, Michal et al. [MJ94] e Hu [Hu95] utilizam uma estratégia gulosa baseada no cálculo de fatores de dependência [10]. O algoritmo adiciona atributos a um d-reduto candidato, visando incrementar seu fator de dependência. Contrariamente, Deogun et al. [DCRS98] desenvolveram um algoritmo que elimina sistematicamente atributos não relevantes de um conjunto inicial de atributos, baseando-se, para isso, no conceito de aproximação superior.

Um aspecto importante a ser considerado é que na pesquisa realizada por Susmaga [Sus98] foram comparados diferentes algoritmos para encontrar d-redutos, concluindo-se que aqueles baseados em matrizes de discernimento relativas oferecem melhores resultados.

Também existem algoritmos que buscam redutos usando AGs como os propostos em [PS99]

[10] A importância deste conceito em problemas de classificação deve-se ao fato de que usando o fator de dependência é possível medir o grau em que as classes determinadas pelo atributo de decisão dependem dos atributos de condição.

e [ZKF02]; porém, o seu funcionamento é difícil de analisar e o tempo de resposta é maior que o das outras heurísticas [HLS03]. Outro algoritmo proposto é o de Hedar et al. [HWF06], que utilizam a busca tabu, no entanto, este algoritmo não é baseado no conceito de matriz de discernimento.

Por outro lado, Hu et al. [HLS03] propuseram uma nova heurística denominada *Feature Ranking* que é baseada num mecanismo de *ranking* de atributos e nas idéias provenientes da observação de matrizes de discernimento. Como toda heurística, esta não garante encontrar o reduto ótimo; mas, nos experimentos realizados por Hu et al. [HLS03], é mostrado que este objetivo é atingido na maioria dos casos, superando outras heurísticas existentes.

Dadas essas vantagens, optou-se por usar essa heurística na metodologia de inicialização proposta. As idéias nas quais *Feature Ranking* se baseia, são descritas a seguir:

- Considere-se um SD $S = (U, A \cup \{d\})$. Chame-se a sua uma matriz de discernimento relativa de M, e a suas entradas de c_{ij}. A intersecção de um d-reduto com cada entrada da matriz, não pode ser vazia; pois, se não existisse intersecção entre alguma entrada c_{ij} e algum d-reduto do SD, o suposto d-reduto não conseguiria discernir o objeto i do objeto j, fato que contradiz o conceito de d-reduto. Isto é válido desde que a entrada em questão seja não vazia.

  Usando esta idéia, é possível encontrar um d-reduto de um SD, primeiro, escolhendo uma entrada qualquer de M; posteriormente, verificando se existe intersecção entre essa entrada e o d-reduto candidato, chame-se de red, (que inicialmente não contém nenhum atributo), e finalmente, no caso de não existir intersecção, escolhendo aleatoriamente um atributo dessa entrada, e adicionando-o a red. Repetindo estes passos até que todas a entradas da matriz sejam examinadas pode-se encontrar um d-reduto, de maneira direta.

  No entanto, aplicando apenas a idéia exposta anteriormente, nem sempre é possível encontrar o reduto ótimo. No exemplo apresentado em [HLS03], é dado um SD cuja matriz possui três entradas:$\{a_1, a_3\}$,$\{a_2, a_3\}$ e $\{a_3\}$; aplicando os passos descritos previamente, caberia a possibilidade de encontrar o d-reduto, $\{a_1, a_2, a_3\}$, sendo este, o resultado de escolher os atributos a_1, a_2 e a_3 para as entradas $\{a_1, a_3\}$,$\{a_2, a_3\}$ e $\{a_3\}$, respectivamente. No entanto, é claro que o d-reduto para esse exemplo é $\{a_3\}$.

- Visando contornar esse problema, Hu et al. [HLS03] consideraram dar uma ordenação à forma na qual as entradas da matriz são examinadas. Hu et al. [HLS03] observaram que as entradas c_{ij} que aparecem com maior freqüência na matriz, e que contêm poucos atributos, geralmente, possuem atributos que formam parte de um d-reduto. Para o exemplo em estudo, ordenando a matriz de acordo com o critério explicado, tem-se que a primeira entrada examinada é $\{a_3\}$, pelo que o atributo escolhido para formar parte de red é a_3; assim seria obtida a resposta desejada, i.e., $red = \{a_3\}$.

- Hu et al. [HLS03] notaram outro aspecto importante. Geralmente, quanto mais vezes aparece um atributo na matriz de discernimento, mais importante este é. Assim, no caso de não existir intersecção entre o d-reduto candidato red e uma

entrada c_{ij} examinada, a escolha do atributo de c_{ij} para formar parte de red não é aleatória; mas, está baseada na significação dos atributos de c_{ij}. Com esse intuito, Hu et al. [HLS03] consideraram fazer, no inicio do seu algoritmo, uma ordenação dos atributos do SD, baseada na significação dos mesmos.

Para isso, calcularam para cada atributo $a \in A$, um peso ν, inicializado em zero e atualizado cada vez que a estivesse numa entrada c_{ij} da matriz. O peso ν de um atributo qualquer a, é definido como,

$$\nu(a) = \nu(a) + \frac{|A|}{|c_{ij}|}, \tag{3.12}$$

sendo $|A|$ a cardinalidade do conjunto de atributos de problema, e $|c_{ij}|$ a cardinalidade da entrada atualmente examinada.

A seguir, será dado o algoritmo para a determinação de redutos proposto por Hu et al. [HLS03].

Algoritmo 2 *Feature Ranking*

Entrada: Um SD, $S = \{U, A \cup \{d\}\}$
Saída: Um conjunto de atributos red
1: $red = \emptyset$, $\forall a \in A$, $\nu(a) = 0$.
2: Criar a matriz de discernimento M, e calcular o peso $\nu(a)$, $\forall a \in A$
3: Ordenar a matriz de discernimento M
4: **Para cada** entrada c_{ij} de M **faça**
5: **Se** $c_{ij} \cap red == \emptyset$ **então**
6: Selecionar o atributo $a \in c_{ij}$ com o maior peso
7: $red = red \cup \{a\}$
8: **Fim se**
9: **Fim para**
10: **Retornar** red

Na linha 2 quando uma entrada c_{ij} de M é calculada, $\nu(a)$ é atualizada usando a Equação 3.12 para cada $a \in c_{ij}$. Na linha 3, as entradas são ordenadas usando o critério da cardinalidade e freqüência de cada c_{ij}. Nas linha de 4 a 10, cada entrada de M é examinada segundo a ordem estabelecida e é gerado o reduto ótimo.

Merece ser mencionado, que o algoritmo pode ser aplicado tanto para encontrar d-redutos quanto para encontrar redutos. Para o primeiro caso é encontrada a matriz de discernimento relativa do SD e sobre esta é aplicado o algoritmo em questão. Para o segundo, o algoritmo é aplicado sobre a matriz de discernimento do SI.

3.6.6 Regras de Dependência

Uma regra de dependência indica os atributos condicionais que permitem discernir as classes dadas pelo atributo de decisão, ou o que equivale a dizer que indica os atributos condicionais dos quais depende o atributo de decisão.

Encontrados os d-redutos de um SD, a geração de regras de dependência é praticamente automática. Para transformar um d-reduto numa regra de dependência, deve-se conectar os atributos presentes no d-reduto mediante conjunções, obtendo-se assim o antecedente da regra. Para completá-la, coloca-se no conseqüente o valor correspondente ao atributo de decisão do SD utilizado. Para representar uma regra de dependência, usar-se-á a notação: $AT \rightarrow CT$, em que AT é o antecedente da regra, constituído pelos atributos condicionais presentes no d-reduto B; CT é o conseqüente da mesma, i.e., o atributo de decisão.

O procedimento de geração de regras será ilustrado, usando os d-redutos encontrados na Seção 3.6.4, os quais são mostrados a seguir:

$$\{\textit{diploma, experiência}\}, \{\textit{experiência, referência}\}, \{\textit{diploma, referência}\}.$$

Aplicando-se o procedimento descrito ao primeiro reduto e conforme a notação estabelecida, obtém-se a seguinte regra de dependência:

$$i \wedge e \rightarrow d.$$

A regra de dependência obtida indica que a decisão de contratar ou não um candidato depende dos atributos *diploma* e *experiência*. Pode-se ver que uma regra de dependência é uma tradução direta de um d-reduto. Também, é importante mencionar que as regras de dependência somente indicam os atributos condicionais que determinam uma decisão, e não os valores que estes devem tomar.

Para os redutos restantes, suas regras de dependência são as seguintes:

$$e \wedge r \rightarrow d,$$
$$i \wedge r \rightarrow d.$$

Observa-se que o número de regras de dependência geradas depende do número de d-redutos determinados.

3.7 Discretização de Atributos *Fuzzy*

Discretizar é um processo que converte atributos numéricos em atributos simbólicos, particionando o domínio dos mesmos em intervalos. Os valores que dividem o domínio dos atributos são denominados *pontos de corte*. Em alguns algoritmos de discretização, os pontos de corte são os mesmos valores dos atributos; em outros, os pontos de corte são médias de dois valores consecutivos de um atributo [GB02].

Dado que parte da metodologia proposta usa os conceitos de *rough sets*, apenas o método de discretização relacionado com a metodologia em questão, será abordado.

Quando os atributos de um SI representam conjuntos *fuzzy* e precisa-se discretizar os seus valores (i.e., os graus de pertinência aos conjuntos *fuzzy* definidos), como é o caso

de um SI cujos atributos têm sido fuzzificados, a discretização pode ser feita empregando apenas um ponto de corte. Assim, os valores maiores ou iguais que o ponto de corte, são estabelecidos como 1; enquanto que, os valores menores que o ponto de corte, são estabelecidos como 0.

O exemplo a seguir, ilustra a maneira como o processo de discretização (nesse caso específico, também uma binarização) forma grânulos de informação [11]. O espaço dos atributos, quando é discretizado dessa forma, gera C^m grânulos *crisp*, em que C é o número de conjuntos *fuzzy* por atributo e m é o número de atributos. Por exemplo, considere-se um problema com 2 atributos (i.e., $m = 2$), a_1 e a_2, e com 3 conjuntos *fuzzy* por atributo (i.e., $C = 3$), a saber, *baixo*, *médio* e *alto*. O número de grânulos *crisp* formados será igual a 9. A Figura 3.1[12] ilustra a geração de grânulos *crisp* a partir da representação *fuzzy* dos atributos a_1 e a_2, sendo o ponto de corte igual a 0.5.

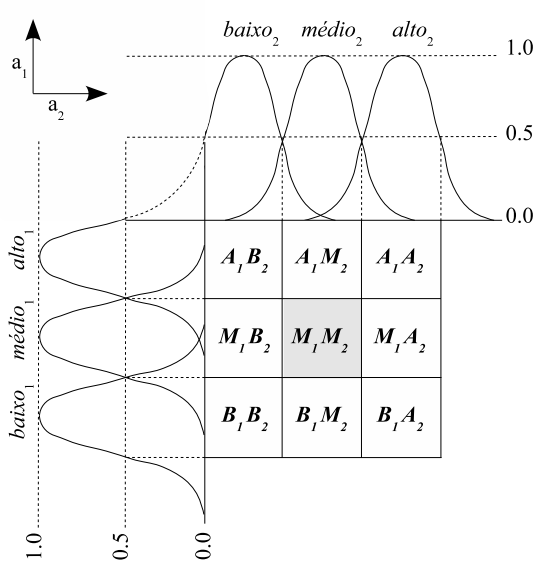


Figura 3.1: Formação de grânulos *crisp* a partir de conjuntos *fuzzy*.

A região sombreada (M_1M_2) ressalta o grânulo *crisp* obtido, após aplicar o ponto de corte aos conjuntos *fuzzy*, *médio*$_1$ e *médio*$_2$ dos atributos a_1 e a_2, respectivamente. Este grânulo agrupa os objetos do problema que pertencem em maior grau (num grau maior que 0.5) aos conjuntos *fuzzy*, *médio*$_1$ e *médio*$_2$ dos atributos a_1 e a_2, respectivamente.

3.8 Considerações Finais

Neste capítulo foram expostas as noções básicas de *rough sets* necessárias para a determinação de redutos e para a geração de regras. Também foi feita uma revisão das heurísticas propostas na literatura para a determinação de d-redutos; e dadas suas vantagens, foi descrita a heurística proposta por Hu et al. [HLS03]. Por último, visto que

[11] Ver Apêndice A

[12] Fonte: Pattern Recognition Algorithms for Data Mining [PM04], página 115.

a teoria de *rough sets* opera sobre grânulos de informação, foi apresentado o tópico da discretização de atributos, e dentro deste, a maneira como os valores de atributos que representam conjuntos *fuzzy* podem ser discretizados.

No seguinte capítulo, será descrita a metodologia de inicialização do SNFDI que usa os conceitos abordados neste capítulo e os conceitos de AGs dados na Seção 2.6.

Capítulo 4

Metodologia de Inicialização para o SNFDI

Neste capítulo será descrita a metodologia proposta para inicializar o SNFDI. Entende-se "inicializar" como determinar o número de neurônios de camada intermediária, e os pesos das conexões entre seus neurônios. Lembre-se que o número de neurônios da camada de saída e da camada de entrada são trivialmente determinados, com base no número de entradas dos exemplos fuzzificados e no número de classes do problema a ser tratado, respectivamente; e portanto, não são considerados nesta inicialização.

A metodologia de inicialização está dividida em três etapas que serão descritas detalhadamente nas Seções 4.1, 4.2 e 4.3; mas que de maneira sucinta, serão expostas a seguir.

- Etapa 1: O objetivo desta etapa é preparar os exemplos de treinamento para que possam ser utilizados pela teoria de *rough sets* na etapa posterior. Entre outras tarefas, é realizado um processo de binarização, usando os exemplos de treinamento previamente fuzzificados.

- Etapa 2: Nesta etapa é realizado o processo de geração de regras de dependência aplicando os conceitos da teoria de *rough sets* (ver Capítulo 3) e usando os exemplos de treinamento já processados na etapa anterior. São geradas regras de dependência de dois tipos, as do primeiro tipo têm o poder de discernir os representantes de classes diferentes, e são extraídas do SD inteiro. As do segundo tipo discernem os representantes dentro de uma classe, e são extraídas de uma parte do SD correspondente à classe tratada.

  No restante do texto, quando se fizer menção ao termo "regras" estará se referindo a regras de dependência.

- Etapa 3: Integrando uma regra do primeiro tipo com uma do segundo é criada uma única regra de dependência que cumpre a função das regras de ambos os tipos, i.e., discernir qualquer um representante de qualquer uma classe de algum outro representante de uma classe diferente, e discernir os representantes de uma classe. Nesta metodologia, esta regra é chamada de *regra integrada*.

Na metodologia proposta foi feita a suposição de que nem todas as regras integradas de uma determinada classe são necessárias; conseqüentemente, nem todas elas serão mapeadas no SNFDI. A escolha das regras integradas é feita mediante a aplicação de Algoritmos Genéticos (AGs), sendo cada indivíduo da população em que o AG atua, uma combinação específica das regras integradas das classes. Assim, o AG busca no espaço de regras uma combinação delas que ao ser mapeada determine o melhor SNFDI, i.e., aquele com a maior capacidade de generalização.

Cada indivíduo dá origem a um SNFDI com uma determinada configuração inicial; ou seja, a um SNFDI com uma arquitetura e pesos específicos particulares. A função de aptidão[1] deste AG é baseada na taxa de acertos, no coeficiente *kappa*[2] e no número de neurônios da camada intermediária. Assim, o AG realiza:

- A integração de regras de ambos os tipos, formando-se regras integradas para cada classe;
- O mapeamento de combinações de regras integradas a diferentes SNFDI;
- O treinamento e teste destes últimos, visando encontrar o indivíduo ótimo (i.e., a melhor arquitetura de SNFDI, entre as avaliadas) que maximize a função de aptidão.

Visando ilustrar esta metodologia, usar-se-á, como exemplo, o conjunto de dados Iris[3]. Os exemplos deste conjunto correspondem às observações de uma flor do gênero Iris. Cada exemplo possui 4 atributos (observações) correspondentes à largura e comprimento da sépala (SL e SW), e à largura e comprimento da pétala (PL e PW), e 1 atributo correspondente à espécie (classe) de Iris, podendo ser, Iris setosa (SE), Iris versicolor (VE) ou Iris virgínica (VI). Vale destacar que os exemplos exibidos para ilustrar está metodologia, são apenas suposições e não resultados reais da aplicação desta metodologia.

Nas seções a seguir, serão descritas as etapas da metodologia proposta.

4.1 Etapa 1: Pré-processamento dos Dados

O objetivo desta etapa é tratar os exemplos de treinamento a fim de que possam ser utilizados pela teoria de *rough sets* na Etapa 2. Os exemplos de teste não passam por esta etapa, e são apenas fuzzificados (ver Seção 2.5.1) e posteriormente, utilizados para a mensuração da capacidade de generalização na Etapa 3.

É importante ressaltar que no início do pré-processamento, os exemplos de treinamento já devem estar fuzzificados. Por exemplo, dado o padrão $\overrightarrow{X} = [5.8\ 2.7\ 3.9\ 1.2\ ]$ correspondente à classe Iris VI; a fuzzificação dos atributos condicionais converte a $\overrightarrow{X}$ num

[1] Nesta metodologia a função de aptidão é a mesma que a função objetivo, pois o que se deseja maximizar é o *kappa* e a taxa de acertos, e minimizar o número de neurônios.

[2] A taxa de acertos e o coeficiente *kappa* são medidas para avaliar a precisão da classificação. Ambos os conceitos serão definidos na Seção 5.2.

[3] O conjunto de dados Iris foi criado por E. Anderson. Os dados estão disponíveis no UCI Machine Learning Repository, http://www.ics.uci.edu/∼ mlearn/MLRepository.html.

vetor 12-dimensional com valores entre -1 e 1, $\overrightarrow{X_f}$ = [0.2 0.9 -0.2 0.5 -0.2 -0.5 0.0 0.9 0.0 0.1 0.9 -0.1]; sendo cada um dos elementos do vetor, graus de pertinência aos conjuntos *fuzzy*, *baixo* (B), *médio* (M), e *alto*(A) relativos a cada atributo. Os conjuntos *fuzzy* para os 4 atributos são representados por, $B_1, M_1, A_1, \ldots, B_4, M_4, A_4$. Portanto, um padrão fuzzificado de treinamento para o conjunto de dados Iris terá a forma:

$$B_1, M_1, A_1, B_2, M_2, A_2, B_3, M_3, A_3, B_4, M_4, A_4, \ \textit{espécie}.$$

A etapa de pré-processamento, por sua vez, está dividida em 3 subetapas: Binarização dos exemplos de treinamento, Remoção de exemplos repetidos e Remoção de exemplos pouco freqüentes. As duas primeiras subetapas são realizadas conforme Banerjee et al. [BMP98]. As subetapas citadas previamente serão descritas nas seções a seguir.

4.1.1 Etapa 1a: Binarização dos Dados

Devido a que a teoria de *rough sets* opera sobre grânulos de informação *crisp*[4] (i.e., partições de um universo) precisa-se converter os graus de pertinência a valores *crisp*. Geralmente, a discretização é o processo empregado para se criar esse tipo de grânulos. Porém, na literatura, quando conjuntos *fuzzy* estão envolvidos no processo, geralmente, a discretização é substituída pela binarização. Esta última é realizada escolhendo um ponto de corte (pc), e aplicando-o sobre os valores fuzzificados como descrito a seguir: Aqueles valores maiores ou iguais que o ponto de corte escolhido, tornam-se 1; os restantes, tornam-se 0.

A política de formação de grânulos *crisp*, empregada nesta metodologia foi a binarização. Assim, os N exemplos fuzzificados de treinamento $3m$-dimensionais são convertidos em N exemplos de treinamento $3m$-dimensionais binários. No exemplo anterior, usando um ponto de corte igual a 0.2, $\overrightarrow{X_f}$ torna-se $\overrightarrow{X_b}$ = [1 1 0 1 0 0 0 1 0 0 1 0].

4.1.2 Etapa 1b: Remoção de Objetos Repetidos e Indiscerníveis

Após a binarização, é possível que objetos (exemplos) que tiveram diferentes valores (i.e., diferentes graus de pertinência aos conjuntos *fuzzy* definidos para os atributos) possuam os mesmos valores após a binarização; ou seja, que formem parte do mesmo grânulo de informação. Devido a esse fato, não é necessária a presença de todos os objetos de um conjunto de dados, mas apenas de um elemento representativo. Os objetos repetidos e/ou indiscerníveis dentro de cada classe do domínio são eliminados aplicando o conceito de classes de equivalência tratados na Seção 3.3.

Considere 30 exemplos de treinamento para a classe SE. Suponha-se que após a fase de binarização, a parte do SD correspondente à classe SE seja como mostrado na Tabela 4.1 (i.e., com os primeiros 14 elementos (de u_1 a u_{14}) indiscerníveis entre sim, com os 14 seguintes (de u_{15} a u_{28}) indiscerníveis entre sim, e com os elementos u_{29} e u_{30} indiscerníveis

[4]Ver o Apêndice A.

entre sim). Encontrando classes de equivalência, e escolhendo um objeto representante para elas, os exemplos para a classe Iris SE ficam como mostrado na Tabela 4.2; i.e., com apenas 3 objetos representativos. A remoção deve ser feita para todas as classes do domínio; ou seja, o mesmo processo deve ser aplicado às classes VI e VE.

	B_1	M_1	A_1	B_2	M_2	A_2	B_3	M_3	A_3	B_4	M_4	A_4	*espécie*
u_1	1	0	0	0	1	0	1	0	0	1	0	0	SE
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
u_{14}	1	0	0	0	1	0	1	0	0	1	0	0	SE
u_{15}	1	0	1	0	0	0	1	0	0	1	0	0	SE
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
u_{28}	1	0	1	0	0	0	1	0	0	1	0	0	SE
u_{29}	0	1	1	0	0	1	1	0	0	1	0	0	SE
u_{30}	0	1	1	0	0	1	1	0	0	1	0	0	SE

Tabela 4.1: Exemplos da espécie Iris SE antes da fase de remoção de objetos repetidos e indiscerníveis.

	B_1	M_1	A_1	B_2	M_2	A_2	B_3	M_3	A_3	B_4	M_4	A_4	*espécie*
u_1	1	0	0	0	1	0	1	0	0	1	0	0	SE
u_{15}	1	0	1	0	0	0	1	0	0	1	0	0	SE
u_{29}	0	1	1	0	0	1	1	0	0	1	0	0	SE

Tabela 4.2: Exemplos da espécie Iris SE após a fase de remoção de objetos repetidos e indiscerníveis.

4.1.3 Etapa 1c: Remoção de Objetos Pouco Freqüentes

Nota-se no exemplo da seção anterior que existem objetos que se repetem com mais freqüência que outros. Os primeiros dois elementos da Tabela 4.2 representam cada um o 47% (14 elementos de 30) do total de exemplos; enquanto que, o terceiro elemento representa apenas o 6% (2 elementos de 30) dos exemplos. Este fato pode refletir duas situações:

- Existem objetos que são consideravelmente mais representativos que outros;
- A binarização pode dar origem a novos falsos representantes, devido à sensibilidade do ponto de corte escolhido.

Em ambas as situações, optou-se por remover estes objetos. Visando esse objetivo, foram empregadas duas heurísticas:

- A primeira heurística é utilizada quando há objetos muito representativos; i.e., quando existem porcentagens de representatividade significativas (altas), como no caso do exemplo em estudo. Esta heurística consiste em conservar os elementos com alta porcentagem e remover aqueles com baixa porcentagem. Na implementação, a porcentagem usada foi de 10%. No exemplo, seguindo esta heurística, o elemento que representa apenas o 6% do conjunto de SE é eliminado.
- A segunda heurística é utilizada quando as representatividades não são muito marcantes. Neste caso, a média aritmética das porcentagens (não repetidas) é usada como limiar para determinar os elementos que são eliminados. Por exemplo, aplicando esta heurística, numa situação na qual tem-se as seguintes porcentagens de representatividade, 7%, 7%, 7%, 6%, 6%, 6%, 6%, 6%, 6%, 4%, 4%, 4%, 5%, 5%, 5%, 3%, 3%, 3%, 2%, 2%, 0.5%, 0.5%, 0.5%, 0.5%, 0.5%, 0.5%; os objetos com porcentagens de representatividade menor que 4% (média de 7, 6, 4, 5, 3, 2 e 0.5) são eliminados.

Como resultado da etapa de pré-processamento obtém-se um SD binarizado constituído apenas pelos representantes mas significativos de cada classe. Na seguinte seção ver-se-á como a partir do SD obtido, pode-se gerar regras de dependência tanto para o SD quanto para cada classe do domínio.

4.2 Etapa 2: Geração de Regras de Dependência

Na Seção 3.6.6 foi mostrado o modo como são geradas regras de dependência, as quais permitem distingüir as diversas classes de um domínio. Assim mesmo, sabendo da existência de classes com mais de um representante, faz-se necessário a inclusão de algum mecanismo que leve em conta a multirepresentatividade das mesmas, a qual pode ser tratada, encontrando regras de dependência que possuam informação para discernir os vários representantes de uma determinada classe. Desta perspectiva, o processo de geração de regras de dependência se refere à geração de regras com dois objetivos distintos. Estes são atingidos nas 2 subetapas citadas a seguir:

- Geração das regras de dependência RSD, a fim de encontrar regras que distinguam qualquer um representante de qualquer uma classe de outro representante de uma classe diferente.
- Geração das regras de dependência R, visando encontrar regras que distinguam os representantes, se houvesse mais de um, das diferentes classes do domínio.

Nas seções a seguir, serão descritas as subetapas listadas previamente.

4.2.1 Etapa 2a: Geração de Regras de Dependência RSD

Uma regra ou um conjunto de regras de dependência são gerados para o SD encontrado na etapa de pré-processamento de dados (Etapa 1). Essas regras têm o poder de discernir

as diferentes classes de um domínio, i.e., qualquer um representante de qualquer uma classe de outro representante de uma classe diferente. Conforme explicado na Seção 3.6.6, o primeiro passo na geração de regras de dependência é a determinação dos d-redutos do SD. Por ser sua determinação um problema NP-difícil, são usadas heurísticas, como as descritas na Seção 3.6.5.

Nesta metodologia optou-se por usar, para a determinação de d-redutos, o algoritmo *Feature Ranking* [HLS03]. No entanto, foram realizadas algumas modificações, a fim de possibilitar a determinação de mais de um reduto, se o houvesse.

A modificação feita consiste no seguinte: depois que foi encontrado o reduto ótimo, usando *Feature Ranking*, para cada atributo do reduto é realizado um procedimento que consiste em criar um suposto reduto, removendo o atributo em questão e adicionando outro atributo não presente no reduto original. Se após esta mudança, o poder de classificação do reduto original for igual ao do suposto reduto, então este último passará a fazer parte do conjunto de redutos. A idéia por trás desta modificação é baseada no fato de que os redutos de um SI possuem atributos em comum; portanto, é muito provável que eles se diferenciem em apenas um atributo. Esta observação também é válida para o caso dos d-redutos.

Por outro lado, lembre-se que para cada d-reduto encontrado no SD, uma regra de dependência é gerada. Portanto, existirão tantas regras quanto d-redutos existam no SD. Estas regras serão chamadas *regras RSD* e serão denotadas por RSD. Por exemplo, supondo que para o SD do conjunto Iris existam os seguintes d-redutos:

$$\{M_3, A_3, B_4, M_4\},\{M_3, A_3, M_4, A_4\};$$

as regras de dependência geradas seriam as seguintes:

$$RSD_1 : M_3 \wedge A_3 \wedge B_4 \wedge M_4 \rightarrow \textit{espécie},$$
$$RSD_2 : M_3 \wedge A_3 \wedge M_4 \wedge A_4 \rightarrow \textit{espécie}.$$

Estas regras indicam que para diferenciar as espécies de Iris VI, SE e VE é necessário considerar os atributos M_3, A_3, B_4 e M_4 ou considerar os atributos M_3, A_3, M_4 e A_4.

4.2.2 Etapa 2b: Geração de Regras de Dependência R

Dado que os problemas reais nem sempre têm apenas um representante por classe, mas vários; optou-se por desenvolver uma metodologia que considere a multirepresentatividade das classes. As regras geradas conforme será explicado a seguir, permitem discernir seus representantes de uma classe, desde que exista mais de um representante na classe tratada. A idéia principal, extraída de Banerjee et al. [BMP98], consiste em dividir o SD em vários subsistemas, sendo cada subsistema correspondente a uma classe de um domínio. Todos os objetos (exemplos) dentro de um subsistema possuem o mesmo valor para o atributo de decisão. Cada objeto de um subsistema pode ser considerado como um representante da classe correspondente a esse subsistema.

Cada subsistema pode ser considerado como um SI, pois dentro dele, o atributo de classe deixou de ser relevante. Calculando-se os redutos dos subsistemas é possível se obter informação suficiente para discernir os objetos dentro de cada subsistema, o que equivale a diferenciar os representantes de uma determinada classe, para cada classe. Para encontrar esses redutos, usa-se o algoritmo *Feature Ranking* descrito na Seção 3.6.5. Lembre-se que este algoritmo, também pode ser empregado para calcular redutos, usando a matriz de discernimento, em vez que a matriz de discernimento relativa.

Com base nos redutos encontrados, geram-se regras de dependência de maneira similar à descrita na Seção 3.6.6. O antecedente de uma destas regras é a conjunção dos atributos do reduto que dá origem à regra em questão, e seu conseqüente, a classe correspondente ao subsistema tratado. As regras geradas para cada subsistema do SD serão chamadas *regras* R e serão denotadas por R. Por exemplo, uma regra gerada para o subsistema correspondente à classe SE, poderia ser:

$$R : M_2 \wedge A_3 \wedge B_4 \wedge M_4 \rightarrow \text{SE}.$$

Nota-se que o conseqüente desta regra é específico à classe SE, ou seja, é específico ao subsistema que deu-lhe origem. Esta regra indica que para diferenciar os diversos representantes da classe SE, devem ser considerados os atributos M_2, A_3, B_4 e M_4. Diferentemente, nas regras do SD Iris, dadas na seção anterior, o conseqüente é simplesmente, *espécie*.

Um aspecto a ser considerado é que podem existir classes com apenas um representante, i.e., subsistemas com só um elemento. Naturalmente, nesses casos não existem regras para distingüir os diferentes representantes da classe em questão, pois só existe um representante. Porém, isso não significa que não existam regras para diferenciar o representante dessa classe do resto das outras, pois considerando as regras do SD inteiro, i.e., aquelas que diferenciam as classes, esse problema é contornado.

Finalmente, como resultado de ambas as subetapas, obtém-se uma regra ou um conjunto de regras de dependência que distinguem as várias classes de um problema dado; e também, uma regra ou várias regras de dependência para cada classe, as quais permitem diferenciar os diversos representantes das mesmas. No entanto, para que uma regra atinja os objetivos que cada tipo de regra tem, individualmente, é preciso integrá-las em uma regra única. Esse processo será descrito na Seção 4.3.1.

Na seguinte seção será visto o funcionamento do AG proposto, que inclui o processo de integração e mapeamento de regras de dependência a diferentes SNFDI e a sua respectiva avaliação.

4.3 Etapa 3: Execução do Algoritmo Genético

Uma regra de dependência R é capaz de distingüir os representantes de uma determinada classe. Para que essa regra seja capaz de distingüir as classes é necessário integrá-la com alguma das regras obtidas para o SD inteiro, e deste modo formar um nova regra de

dependência. A integração de ambos os tipos de regras é outra das tarefas realizadas pelo AG, e será descrita na Seção 4.3.1.

Por outra parte, a metodologia apresentada foi desenvolvida sob o suposto de que nem todas as regras integradas são necessárias para inicializar o SNFDI; conseqüentemente, nem todas elas devem ser consideradas no mapeamento. A idéia geral desta etapa é criar vários grupos (combinações) de regras integradas, que não necessariamente incluem todas as regras geradas, e mapeá-los a seus respectivos SNFDIs. Por ser a criação dos grupos de regras integradas um problema combinatório, optou-se por realizar esta etapa usando um AG. Destaca-se que Banerjee et al. [BMP98] não propõem nenhuma forma para resolver este problema, e que realizam a combinação de todas as regras possíveis.

Por meio deste AG, determina-se o melhor indivíduo (arquitetura e seus pesos iniciais) que fornece a maior capacidade de generalização. Cada indivíduo da população do AG é uma combinação das regras integradas e dá origem a um SNFDI com uma configuração inicial diferente. Estes indivíduos evoluem mediante um processo iterativo, no qual são aplicados os operadores de mutação e cruzamento projetados para este AG. Para avaliar a evolução dos indivíduos, usa-se uma função de aptidão que é baseada na taxa de acertos, no coeficiente *kappa* e no número de neurônios da camada intermediária. Assim, o AG é o responsável da integração de regras de dependência, da combinação de regras integradas, do mapeamento das mesmas a diferentes SNFDIs, e da avaliação destes últimos, mediante a sua função de aptidão. Esta última implica a execução do treinamento e teste.

4.3.1 Integração de Regras de Dependência

A finalidade de integrar uma regra de dependência que discerne as classes de um SD, com uma que discerne os representantes das classes, é criar uma regra de dependência, denominada *regra integrada*, com o poder que as anteriores têm isoladamente, e assim, contornar a limitação da metodologia de Banerjee et al. [BMP98], que no seu modelo de multirepresentatividade não considera a informação para diferenciar as classes, e somente considera a dos representantes destas.

Integrando duas regras, uma R e outra RSD, o antecedente da nova regra é a conjunção dos antecedentes das duas primeiras. Por exemplo, suponha-se que uma das regras geradas para o subsistema da classe SE, i.e., uma regra R, fosse:

$$R\text{: } A_2 \wedge M_2 \wedge B_1 \rightarrow \text{SE},$$

e que uma das regras do SD Iris, i.e., uma regra RSD, fosse:

$$RSD\text{: } A_3 \wedge B_1 \rightarrow \textit{espécie};$$

integrando ambas as regras, conforme o explicado previamente, obtém-se, a regra a seguir:

$$R - RSD\text{: } A_3 \wedge B_1 \wedge A_2 \wedge M_2 \rightarrow \text{SE}.$$

No texto, "-" representará integração de regras. Vale ressaltar que no conseqüente de uma regra integrada, apenas aparece o conseqüente da regra R. No entanto, isto não significa que a nova regra não possua a capacidade de diferenciar as diversas classes de um domínio, pois ao adicionar os atributos do antecedente da regra RSD, esse problema é contornado. A escolha de deixar apenas o conseqüente da regra R, no conseqüente da regra integrada, foi adotada visando simplificar a notação.

Supondo que foram geradas 3 regras de dependência para o subsistema da classe SE, as que integradas com alguma regra RSD deram lugar às seguintes regras integradas,

$$A_3 \wedge B_1 \wedge A_2 \wedge M_2 \rightarrow \text{SE},$$
$$A_1 \wedge M_2 \wedge A_3 \rightarrow \text{SE},$$
$$M_2 \wedge B_3 \rightarrow \text{SE};$$

é possível expressá-las, unidas por disjunções, como a seguir:

$$(A_3 \wedge B_1 \wedge A_2 \wedge M_2) \vee (A_1 \wedge M_2 \wedge A_3) \vee (M_2 \wedge B_3) \rightarrow \text{SE},$$

isto significando que a primeira ou a segunda, ou terceira pode ser usada (pois qualquer uma destas regras cumpre a mesma função), para distingüir os representantes de SE e ao mesmo tempo, distingüir qualquer um representante de SE de qualquer representante de outra espécie de Iris. Também pode ser utilizada qualquer combinação de dois delas, como visto na regra a seguir:

$$(A_1 \wedge M_2 \wedge A_3) \vee (M_2 \wedge B_3) \rightarrow \text{SE},$$

En todo caso, nota-se que os antecedentes destas regras se encontram na Forma Normal Disjuntiva (FND)[5], i.e., os antecedentes são disjunções de conjunções. Uma regra de dependência cujo antecedente é expressado nessa forma, está apta para ser mapeada a um SNFDI com uma arquitetura e pesos iniciais particulares.

Perceba-se que dado um conjunto de regras RSD e de regras R existem não só várias possíveis regras resultantes da integração, mas também, várias possíveis regras na FND dependendo se foram usadas combinações de 1, 2 ou mas regras integradas. Por ser este um problema combinatório, foi resolvido mediante a aplicação de um algoritmo genético que será descrito a seguir.

4.3.2 Representação dos Indivíduos do AG

O objetivo principal desta pesquisa é encontrar o melhor SNFDI no sentido que consiga uma boa taxa de acerto sobre o conjunto de teste. Através do algoritmo genético, busca-se num espaço de regras uma combinação das mesmas que ao ser mapeada determine a arquitetura e pesos de um SNFDI, com os quais sua capacidade de generalização seja

[5]Uma expressão lógica está na Forma Normal Disjuntiva se for a disjunção de conjunções de variáveis booleanas.

maximizada, mas que essa otimização seja atingida com o menor número de neurônios possível. O AG operará sobre uma população cujos indivíduos são conjuntos de regras de dependência obtidas pela aplicação da teoria de *rough sets* sobre um domínio de dados.

Considere um domínio de dados com n classes. Na Figura 4.1, as regras de dependência geradas a partir dos redutos do SD inteiro são denotadas por RSD_q, $q = 1, 2, \ldots, Q$, e as regras geradas a partir dos subsistemas são denotadas por R_{lj}, $l = 1, 2, \ldots, n$ e $j = 1, 2, \ldots, Q_l$, onde n é o número de classes, e Q_l, o número de regras da l-ésima classe.

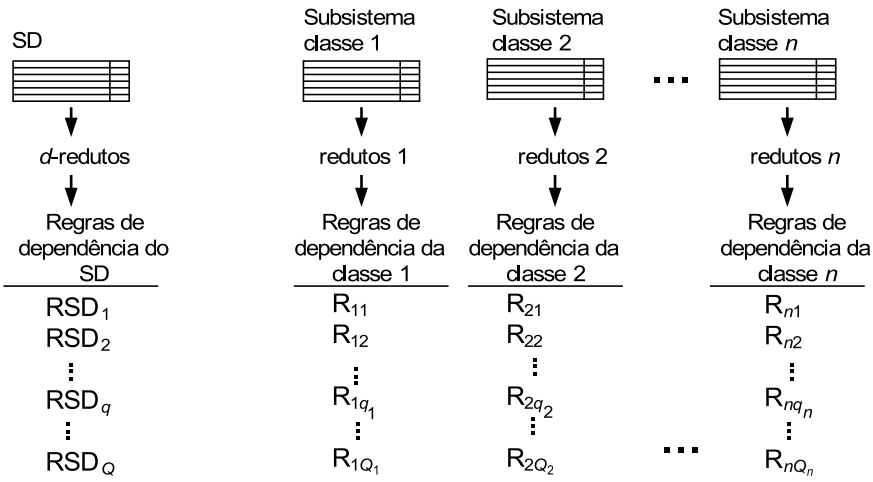


Figura 4.1: Regras de dependência RSD e R.

O indivíduo representado na Figura 4.2, é obtido fazendo a integração, como definido na Seção 4.3.1, entre as regras RSDs e Rs. Observa-se que para cada classe existe uma região no cromossomo; deste modo, existirão $Q_1 + Q_2 + \ldots + Q_n$ posições do tipo $R_{lj} - RSD^*$, em que "-" representa a integração de regras, e "*" indica que a regra RSD é escolhida aleatoriamente, para formar a população inicial de indivíduos. Cada posição do tipo $R - RSD$ é um gene no cromossomo.

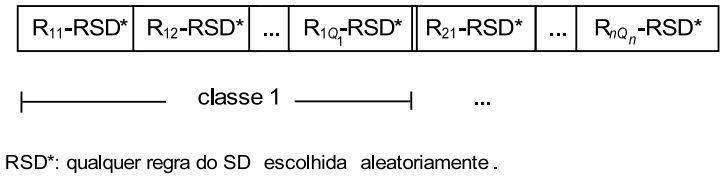


Figura 4.2: Representação de um indivíduo do AG.

É importante mencionar que quando o AG opera sobre os indivíduos não muda a estrutura dos antecedentes e conseqüentes das regras integradas.

4.3.3 Criação da População Inicial

A população inicial de indivíduos é criada, fazendo primeiro a integração de todas as regras R geradas a partir dos subsistemas, com uma RSD escolhida aleatoriamente do

conjunto de regras do SD. Estes indivíduos compostos de regras integradas já poderiam ser mapeados em um SNFDI, pois as regras de cada região formariam uma regra $\Re$ (ver Seção 4.3.7). Contudo, cada indivíduo produziria redes com $Q_1 + Q_2 + \ldots + Q_n$ neurônios na camada intermediária, ou seja, o número máximo possível.

Entretanto, o objetivo é encontrar o SNFDI que generalize melhor, mas que ele faça isso com o menor número possível de neurônios. Visando esse objetivo e a fim de que haja a possibilidade de treinar e testar redes com diferentes arquiteturas, é empregada a política de ativação de regras integradas. Define-se uma regra ativa como uma regra que participa do mapeamento, e uma inativa como uma que não participa.

Foram usados três critérios, definidos a seguir, nos quais é empregada de diferente forma a política de ativação de regras. Dos indivíduos da população inicial, em um quarto deles todas as regras integradas são deixadas ativas, conseqüentemente, o mapeamento destes construirá redes com o número máximo de neurônios.

Em outro um quarto, apenas uma regra integrada, na região de cada classe, é escolhida aleatoriamente para ser ativada. Para estes, a rede terá o número mínimo de neurônios possíveis, ou seja, n neurônios.

Nos dois quartos restantes, o número de regras integradas que serão ativadas variará aleatoriamente entre 1 e Q_l, $l = 1, 2, \ldots, n$, sendo essas regras escolhidas aleatoriamente, dentro da região de cada classe. Estes indivíduos quando mapeados, produzirão redes com um número de neurônios, variável de n a $Q_l + Q_2 + \ldots + Q_l$. Nas Figuras 4.3(a), 4.3(b) e 4.3(c) são mostrados exemplos de indivíduos, criados conforme estes três critérios. As regras usadas para a sua criação são as mostradas na Figura 4.1.

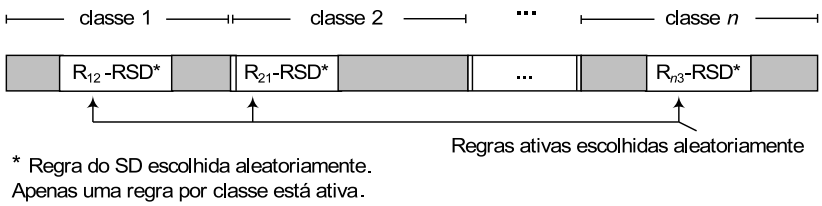
Em uma classe l com apenas um representante, ou seja, em uma classe l para a qual não foi gerada nenhuma regra R ($Q_l = 0$) não é realizada a integração de regras. A região correspondente à classe l no cromossomo possui apenas uma posição $R - RSD$, mas nela só é considerada a regra RSD.

O número de indivíduos da população inicial, denotado por ρ, é um parâmetro que será mudado nos experimentos para avaliar a metodologia de inicialização. Alguns resultados importantes da metodologia, variando o tamanho da população, podem ser vistos nas Tabelas 5.5, 5.10 e 5.15. É importante mencionar que neste trabalho, optou-se por estabelecer tamanhos de população pequenos, pois como é conhecido, o treinamento de RNAs e o processo evolutivo de AGs são tarefas computacionalmente custosas.

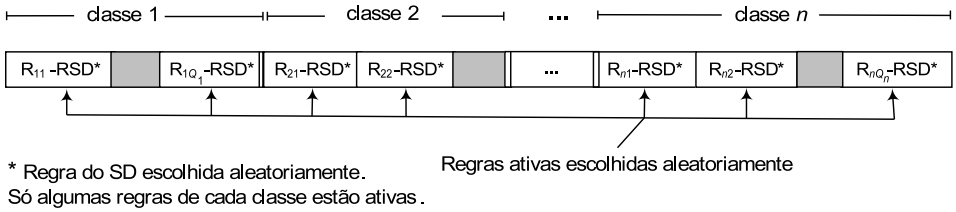
4.3.4 Método de Seleção

Na Seção 4.3.3 foi descrita a composição da população inicial de indivíduos, criada mediante a aplicação de uma política de ativação que busca encontrar uma arquitetura com uma boa generalização e com o menor possível número de neurônios. O seguinte passo é selecionar dentro dessa população, os indivíduos que se reproduzirão, para o qual será necessário usar algum método de seleção. Na literatura, os mais usados são o método da roleta e do torneio, descritos na Seção 2.6.2.

(a) Indivíduo criado conforme o primeiro critério.



(b) Indivíduo criado conforme o segundo critério.



(c) Indivíduo criado conforme o terceiro critério.

Figura 4.3: Criação da população inicial.

Estes últimos foram implementados na metodologia de inicialização; porém, nos experimentos iniciais apresentou-se o problema da convergência prematura. A aplicação destes métodos dá a cada indivíduo chances proporcionais a seu índice de aptidão de estar na próxima população ou de se cruzar com algum outro para gerar novos indivíduos. Isso implica que aqueles mais aptos têm maior probabilidade de estar em uma maior quantidade na população de indivíduos reprodutivos. Isso implica também que aqueles com menor aptidão podem não aparecer nessa população reprodutiva.

Em populações que possuem poucos indivíduos, como as criadas nos experimentos realizados neste trabalho, a aplicação de tais métodos diminui a diversidade da população reprodutiva, produzindo cruzamentos entre indivíduos idênticos em gerações cedas do processo evolutivo. Isto provoca a parada prematura da evolução dos indivíduos.

Devido a esse inconveniente optou-se por dar o direito de reprodução a todos os indivíduos de uma população. Para cada indivíduo é escolhido aleatoriamente um parceiro com o qual este reproduzir-se-á, sendo tal reprodução, guiada pelo operador de cruzamento descrito na seguinte seção.

Vale destacar que é incorreto pensar que o propósito da seleção (i.e., selecionar os indivíduos da população que por terem boa capacidade de adaptação têm direito a transmitir o seus genes à próxima geração) não é considerado, pois esse é cumprido após a aplicação dos operadores genéticos, em um processo chamado "Seleção da Nova População" que será visto na Seção 4.3.10.

4.3.5 Operador de Cruzamento

Os indivíduos da população, como definidos na Seção 4.3.2, são formados por unidades básicas definidas pela integração de uma regra gerada de um subsistema de uma classe e uma regra gerada de um SD. Uma regra integrada, i.e., uma regra $R - RSD$, é o que se denomina gene na nomenclatura de AG, podendo estar ativa ou inativa nas regiões de classe.

Este operador é realizado, escolhendo-se aleatoriamente, para cada região de classe, um ponto de cruzamento a partir do qual haverá troca de subconjuntos de regras integradas. Em outras palavras, dentro de cada região é realizado o cruzamento um-ponto (ver Seção 2.6.3). Para a i-ésima região de classe, $i = 1, \ldots, n$, este ponto de cruzamento varia de 1 a $Q_i - 1$.

Na Figura 4.4, exibe-se dois indivíduos progenitores com duas regiões de classe, sobre os quais foi aplicado o operador de cruzamento, originando dois filhos que são apresentados na parte inferior da mesma figura. É importante esclarecer que a fim de não perder os indivíduos que possuem um alto índice de aptidão, os progenitores não são substituídos por seus filhos, como realizado em um algoritmo genético típico. Os filhos e seus progenitores passam a formar parte de uma população que se chamará de população intermediária[6], a partir da qual é selecionada a população da seguinte geração. Isso será visto na Seção 4.3.9.

Aqueles filhos que após o cruzamento ficaram com alguma região de classe, com todas as suas regras inativas, são desconsiderados, e não são introduzidos na população intermediária.

Pode acontecer que para uma classe l, exista só uma regra de dependência gerada (i.e., $Q_l = 1$), conseqüentemente, a região do cromossomo correspondente a essa classe terá apenas uma posição do tipo $R - RSD$ para todo indivíduo da população. Nesse caso, cruzamento nessa região de classe funciona da seguinte maneira: Um dos filhos herda a regra $R - RSD$ da classe l de um dos progenitores, e o outro filho a herda do outro progenitor. O descrito também vale para classes com apenas um representante, i.e., $Q_l = 0$.

4.3.6 Operador de Mutação

O operador de mutação é aplicado, ou não, con certa probabilidade (determinada pela taxa de mutação), sobre um indivíduo da população intermediária. Dentro de cada indivíduo escolhe-se aleatoriamente uma região de classe, onde será aplicada a mutação *positiva* ou *negativa*, definidas a seguir.

Após a escolha da região de classe a ser mutada, é havendo verificado que esta possui pelo menos duas classes ativas, é aplicada a mutação negativa, que consiste em escolher aleatoriamente uma regra integrada ativa, e desativá-la. Caso contrário, é aplicada a mutação positiva, na qual se ativa uma regra inativa, escolhida aleatoriamente. Nas

[6]Na nomenclatura de AGs, a população intermediária contém os indivíduos que irão se reproduzir. Neste trabalho, além destes indivíduos, esta população contém os filhos e os indivíduos mutados.

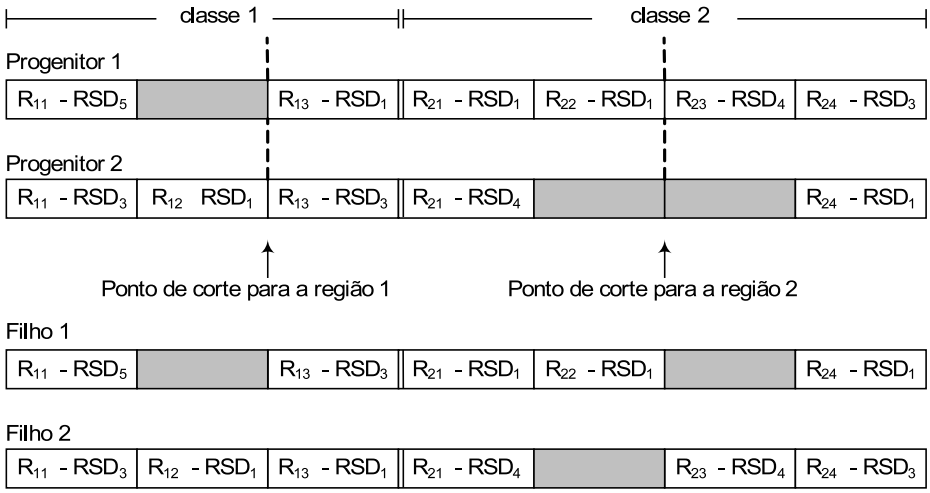


Figura 4.4: Exemplo do operador de cruzamento do AG.

Figura 4.6 e 4.5, ilustra-se a mutação negativa e positiva, respectivamente. Em ambos os exemplos, a mutação é realizada sobre a região da classe 1 que foi escolhida aleatoriamente.

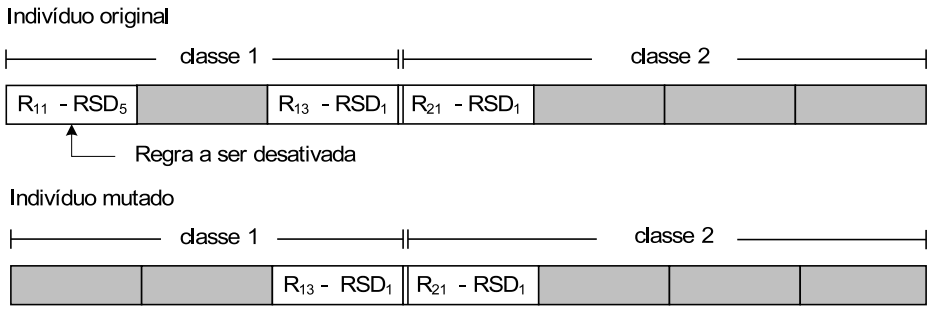


Figura 4.5: Exemplo de mutação positiva.

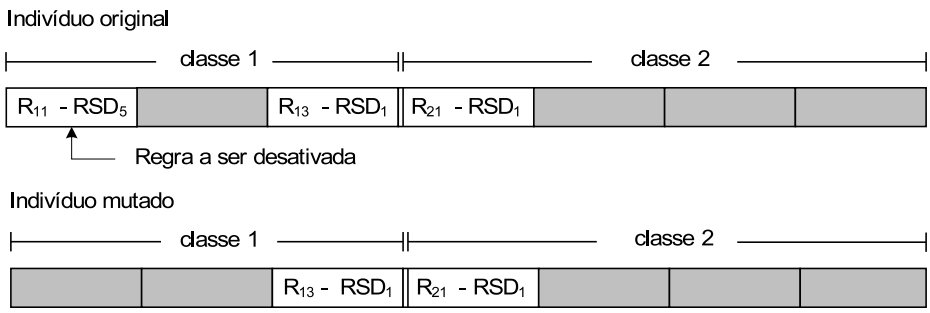


Figura 4.6: Exemplo de mutação negativa.

A mutação negativa busca encontrar indivíduos que ao serem mapeados a seus respectivos SNFDIs possuam arquiteturas com um menor número de neurônios na camada intermediária. A mutação positiva garante que todo indivíduo tenha pelo menos uma regra integrada ativa em cada região de classe.

Em caso de que exista somente uma regra integrada numa região, i.e., apenas uma posição do tipo $R - RSD$ ($Q_l = 1$ ou $Q_l = 0$), a mutação consiste em escolher aleatoriamente outra regra RSD entre as existentes e trocá-la pela regra RSD dentro de $R - RSD$.

Com o objetivo de não perder nunca o melhor indivíduo, o operador de mutação não substitui o indivíduo escolhido para ser mutado; mas, o clona e, posteriormente, realiza a mutação sobre o clone criado. Os novos indivíduos criados pela mutação passam a formar parte da população intermediária.

Determinada assim a população intermediária, é necessário calcular a aptidão de cada um dos seus indivíduos, ou seja, avaliar esta população. Contudo, antes de realizá-la é necessário para mapear as regras ativas de cada indivíduo o a seu respectivo SNFDI.

4.3.7 Mapeamento de Regras de Dependência a um SNFDI

O mapeamento de regras, conforme Banerjee et al. [BMP98], é realizado como a seguir:

1. Criam-se $3m$, sendo que m é o número de atributos.
2. Criam-se n neurônios de saída, onde n é o número de classes.
3. Os antecedentes das regras ativas dentro de cada região de classe de um indivíduo podem ser conectados mediante disjunções formando uma regra $\Re$, cujo antecedente está na FND. De maneira formal para a j-ésima classe, $\Re_j$ é definida via:

$$\Re_j : T_{j1} \vee T_{j2} \vee \ldots \vee T_{ji'} \vee \ldots \vee T_{jn2_j} \rightarrow c_j, \tag{4.1}$$

onde $j = 1, 2, \ldots, n$ e $n2_j$ é o número de regras ativas com conseqüente c_j. Para cada j fixo, $1 \leq n2_j \leq Q_j$ e $T_{ji'}$(regra integrada) é o i'-ésimo componente da FND, com $i' = 1, 2, \ldots, n2_j$, dado por:

$$T_{ji'} = f^j_{i'1} \wedge f^j_{i'2} \wedge \ldots \wedge f^j_{i'l'} \wedge \ldots \wedge f^j_{i'z_{l'}}, \tag{4.2}$$

onde $l' = 1, 2, \ldots z_{l'}$, sendo $z_{l'}$ o número de conjuntos *fuzzy* que são conectados por conjunção para formar o antecedente da regra integrada $T_{ji'}$. Para cada j e i' fixos, $1 \leq z_{l'} \leq 3m$ e $f^j_{i'l'}$ é o l'-ésimo literal ou conjunto *fuzzy* (*baixo, médio* ou *alto*) do antecedente da regra integrada i com conseqüente c_j.

Para cada $j = 1, 2, \ldots, n$:

- Criam-se $n2_j$, neurônio ocultos.
- Cria-se uma conexão entre cada neurônio oculto com o neurônio de saída correspondente à classe c_j expressa no conseqüente de $\Re_j$. Os pesos dessas conexões são inicializados com $w_{ji'} = \frac{1}{n2_j}$, $i' = 1, \ldots, n2_j$.
- Para cada neurônio correspondente a $T_{ji'}$, criam-se conexões com os neurônios de entrada correspondentes a cada literal $f^j_{i'l'}$ de $T_{j'i'}$. Os pesos dessas conexões são inicializados com $w_{i'l'} = \frac{w_{ji'}}{z_{l'}}$.

4. Dado que o SNFDI é uma RNA completamente conectada, são criadas as conexões ausentes, e seus pesos são inicializados com valores maiores e próximos a -1, significando que tais conexões possuem pouca influência.

Lembre-se que o número de classes do domínio é igual ao número de neurônios da camada de saída, i.e., $n3$. O total de conjuntos *fuzzy* definidos para os atributos do conjunto de dados é igual ao número de neurônios da camada de entrada, i.e., $n1$. O número de neurônios da camada intermediária depende do número de componentes (da FND) existentes nos antecedentes de todas as regras a serem mapeadas.

Para ilustrar o mapeamento de regras ao SNFDI, far-se-á a suposição de que foram geradas as seguintes regras $\Re$, para o conjunto Iris:

$$\begin{aligned} \Re_1 &: A_2 \vee (M_2 \wedge B_1) \rightarrow \text{SE}, \\ \Re_2 &: B_4 \vee (M_2 \wedge M_1) \rightarrow \text{VE}, \\ \Re_3 &: A_1 \vee (A_2 \wedge B_1) \rightarrow \text{VI}. \end{aligned}$$

São criados 3 neurônios de saída, um para cada espécie (classe) de Iris, e são criados 12 neurônios de entrada, pois para cada um dos 4 atributos do conjunto Iris, foram definidos três conjuntos *fuzzy*, *baixo, médio,* e *alto*, dando um total de 12 conjuntos *fuzzy*.

Para a regra $\Re_1$, $A_2 \vee (M_2 \wedge B_1)$ →SE: Dado que o antecedente da mesma tem dois componentes (i.e., $n2_1 = 2$), são criados dois neurônios ocultos, um correspondente ao componente A_2, e outro correspondente a $M_2 \wedge B_1$. Estes são conectados ao neurônio de saída correspondente a SE. As conexões entre os dois neurônios ocultos, previamente criados, e o neurônio de saída SE possuem pesos iguais a 0.5, pois $\frac{1}{n2_1} = \frac{1}{2} = 0.5 = w_{11} = w_{12}$.

Posteriormente, para o neurônio oculto correspondente ao componente A_2 é criada uma conexão com o neurônio de entrada correspondente ao atributo A_2. O seu peso é inicializado com 0.5 ($\frac{w_{11}}{z_1} = \frac{0.5}{1} = 0.5$). Similarmente, dado que o componente $M_2 \wedge B_1$ tem dois atributos (i.e., $z_2 = 2$), duas conexões são criadas: Uma entre o neurônio de entrada correspondente a M_2 e o neurônio oculto $M_2 \wedge B_1$; e outra entre o neurônio de entrada B_1 e o mesmo neurônio oculto. Ambas as conexões têm pesos iguais a 0.25, pois $\frac{w_{12}}{z_2} = \frac{0.5}{2} = 0.25$.

Depois de aplicar o mesmo processo às regras $\Re_2$ e $\Re_3$, a arquitetura do SNFDI é a mostrada na Figura 4.7. Nesta figura, apenas foram desenhadas as conexões geradas a partir das regras disponíveis; no entanto, já que o SNFDI é uma RNA completamente conectada, as conexões ausentes, também, devem ser criadas. O valor dos seus pesos são inicializados com valores próximos e maiores a -1.

Nesta metodologia, cada componente de uma expressão na FND, é uma regra integrada, (ver Seção 4.3.1). No exemplo, cada regra $\Re$ possui dois componentes. Isto equivale a dizer, que 6 regras integradas (duas para cada classe de Iris), foram consideradas no processo de mapeamento, conseqüentemente, o número de neurônios da camada intermediária da arquitetura gerada é 6.

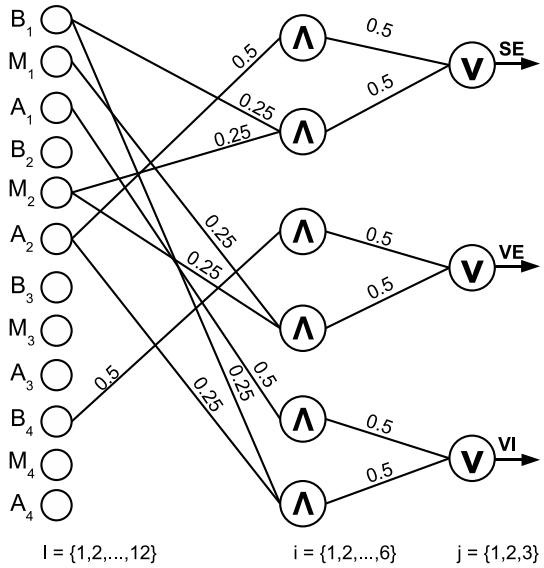


Figura 4.7: Mapeamento das regras, $A_2 \vee (M_2 \wedge B_1) \rightarrow$ SE, $B_4 \vee (M_2 \wedge M_1) \rightarrow$VE e $A_1 \vee (A_2 \wedge B_1) \rightarrow$VI ao SNFDI.

4.3.8 Avaliação da População

Para determinar os indivíduos mais aptos da população intermediária, e conseqüentemente, aqueles que farão parte da seguinte população, é realizado um processo de avaliação, em que para cada indivíduo i de uma população intermediária, é calculada sua função de aptidão, f_i, definida como,

$$f_i = ta_i + \frac{1}{n2_i \times 100} + 1.5kp_i, \tag{4.3}$$

em que ta_i representa a taxa de acerto do SNFDI que se origina do mapeamento do indivíduo i, kp_i representa seu coeficiente *kappa* e $n2_i$ representa o número de neurônios da sua camada intermediária.

O treinamento de cada SNFDI da população é realizado, usando o algoritmo retropropagação descrito na Seção 2.5.6. Depois de um certo número de épocas, faz-se o teste, e calcula-se a taxa de acerto e o coeficiente *kappa*, para posteriormente, calcular seu índice de aptidão mediante a Equação 4.3. Assim, a cada avaliação da população intermediária haverá tantos treinamentos e testes quantos forem os indivíduos da população intermediária.

4.3.9 Seleção da Nova População

Foi visto até a seção anterior que a partir de uma população inicial é criada outra, chamada população intermediária, conformada pelos indivíduos progenitores, seus filhos e os indivíduos criados pela mutação. Estes são avaliados por seu índice de aptidão para o qual são mapeados, treinados e testados.

Para determinar os indivíduos da seguinte geração é levado em conta o índice de aptidão dos indivíduos da população intermediária. Dado que o número de indivíduos ρ de uma população deve permanecer fixo, somente os ρ melhores indivíduos da população intermediária constituirão a população da seguinte geração.

Nota-se então que os indivíduos mais aptos são os que transmitirão seus genes às seguintes gerações. Assim, mediante um processo repetitivo de seleção (dos indivíduos que irão se reproduzir), cruzamento, mutação, mapeamento, avaliação e seleção da nova população, chega-se a uma população final que contém os melhores indivíduos, que ao serem mapeados determinam SNFDIs com arquiteturas e pesos particulares, que resultam em uma boa generalização.

Após a geração da última população do AG, é realizado, para cada um dos seus indivíduos, um último treinamento e teste. Nos treinamentos realizados no processo evolutivo do AG, é empregado um número pequeno de épocas; e no último treinamento é utilizado um número maior. A idéia por trás destes dois limites é, no primeiro caso, diminuir o custo computacional, em termos de tempo de treinamento, e conseqüentemente do AG; e no segundo, dar um maior ajuste aos pesos e assim conseguir uma melhor generalização.

Ambos os limites do número de épocas, são estabelecidos antes do treinamento. Nos experimentos da metodologia, cujos resultados são apresentados no Capítulo 5, estes limites foram os mesmos para todos os experimentos realizados usando o mesmo conjunto de dados.

4.3.10 Algoritmo Genético Proposto

O AG busca num espaço de regras, aquelas que ao serem combinadas, e posteriormente mapeadas, determinem a arquitetura e os pesos de um SNFDI que possua uma boa capacidade de generalização, e que esta seja atingida com o menor número possível de neurônios.

Com a introdução do AG, a busca dos melhores grupos (combinações) de regras deixa de ser um problema combinatório, como o era na metodologia de Banerjee et al. [BMP98].

A seguir, dar-se-á o pseudocódigo da avaliação dos indivíduos de uma população.

Algoritmo 3 Avaliar

Entrada: População P sem índice de aptidão
Saída: População P com índice de aptidão
1: **Para todo** SNFDI i de P **faça**
2: Treinar SNFDI i
3: Testar SNFDI i
4: Calcular a taxa de acerto e *kappa* do SNFDI i
5: Calcular o índice de aptidão do indivíduo i
6: **Fim para**
7: **Retornar** população P

O algoritmo recebe como entrada uma população P para cujos indivíduos não é conhecida sua aptidão. Por meio de um processo de treinamento e teste aplicado a cada um dos indivíduos de P, são calculadas suas taxas de acerto e *kappa*, com base nas quais, posteriormente, são determinados seus índices de aptidão.

Este procedimento intervém dentro do AG, cujo pseudocódigo é dado a seguir:

Algoritmo 4 AG SNFDI

Entrada: ϵ é o número de evoluções que serão executadas, ρ é o tamanho da população.
Saída: Melhor indivíduo
1: $t \leftarrow 0$
2: Criação da população $P(t)$
3: **Enquanto** $t < \epsilon$ **faça**
4: Selecionar os indivíduos reprodutivos de $P(t)$, i.e., adicionar $P(t)$ a $PI(t)$
5: Cruzar $PI(t)$ e adicionar os filhos a $PI(t)$
6: Mutar $PI(t)$ e adicionar os mutados a $PI(t)$
7: Mapear $PI(t)$
8: Avaliar $PI(t)$
9: Selecionar $P(t+1)$ a partir $PI(t)$ usando ρ
10: $t \leftarrow t+1$
11: **Fim enquanto**
12: Treinar $P(t)$
13: Testar $P(t)$
14: Calcular a taxa de acertos para cada indivíduo de $P(t)$
15: **Retornar** indivíduo de $P(t)$ com a maior taxa de acerto

No algoritmo, t representa a geração atual, $P(t)$ é a população da geração t, ρ é o número de indivíduos para toda população do AG, ϵ é o número máximo de evoluções que o algoritmo executa, que também é a condição de parada do algoritmo, e $PI(t)$ é a população intermediária da geração t. O ϵ e o ρ são as entradas do AG.

Na linha 4, os indivíduos de $P(t)$ são adicionados a $PI(t)$, pois todos os indivíduos têm direito a se reproduzirem. Nas linhas 5 e 6 são adicionados a $PI(t)$ os indivíduos filhos e os mutados. Na linha 7 os indivíduos de $PI(t)$ são mapeados a seus respectivos SNFDIs, e posteriormente na linha 8, são avaliados usando o Algoritmo 4. Na linha 9 são selecionados os ρ melhores indivíduos de $P(t)$, os quais passam a formar parte de $P(t+1)$. O treinamento, feito na linha 12, é realizado com um número maior de épocas, o que permite melhorar a precisão dos indivíduos. Da última população escolhe-se o indivíduo que fornece a melhor taxa de acerto. Esse indivíduo constitui a resposta do AG que é retornada na linha 15.

Para escolher o melhor indivíduo da última população não se usa a função de aptidão, pois todos os indivíduos na população $P(\epsilon)$ já foram avaliados usando essa função.

4.4 Considerações Finais

Neste capítulo foi descrita a metodologia de inicialização proposta para o SNFDI, a qual pode ser resumida mediante o diagrama de blocos apresentado na Figura 4.8. As Etapas 1, 2 e 3 da figura, correspondem às etapas descritas nas Seções 4.1, 4.2 e 4.3. As etapas são sucintamente expostas a seguir:

- Etapa 1: Denominada "Pré-processamento dos dados" (ver Seção 4.1), prepara os exemplos de treinamento para que possam ser utilizados nas etapas posteriores.
- Etapa 2: Denominada "Geração de regras de dependência" (ver Seção 4.2), gera dois tipos de regras de dependência: regras que distinguem qualquer um representante de qualquer uma classe de outro representante de uma classe diferente, chamadas regras RSD, e regras que distinguem os diferentes representantes de cada classe, chamadas de regras R.
- Etapa 3: Denominada "Execução do Algoritmo Genético" (ver Seção 4.3), determina a combinação de regras integradas (regras formadas por uma regra R e uma regra RSD mediante um processo denominado Integração, que na Figura 4.8 é realizado na criação da população inicial) que ao ser mapeada no SNFDI maximize a função de aptidão do AG proposto. Isto equivale a encontrar o SNFDI com o menor número de neurônios na camada intermediária e a maior taxa de acerto. Portanto, no processo de avaliação dos indivíduos está envolvido o treinamento e teste do SNFDI.

  Por meio de um processo evolutivo que realiza iterativamente a seleção de indivíduos para se reproduzirem, o cruzamento, a mutação, o mapeamento, e a avaliação dos mesmos, é encontrada uma população final, cujos indivíduos possuem os melhores índices de aptidão. A fim de encontrar uma melhor generalização para tais indivíduos é realizado um último treinamento e teste com um maior número de épocas. Concluído este processo, é escolhido o indivíduo com a melhor taxa de acerto, ou seja, aquele correspondente a um SNFDI com uma arquitetura e pesos que fornecem uma boa generalização.

O processo de fuzzificação que aparece na Figura 4.8 não forma parte da proposta, mas é necessário para o funcionamento de SNFDI. Note-se que o conjunto de dados inteiro é fuzzificado, mas só os exemplos de treinamento passam pelas Etapas 1 e 2, para, com base neles, se gerar regras de dependência. Os exemplos de treinamento fuzzificados também são utilizados nos treinamentos do processo evolutivo do AG e no treinamento final.

No seguinte capítulo serão apresentados os experimentos e os resultados obtidos da implementação da metodologia aqui exposta.

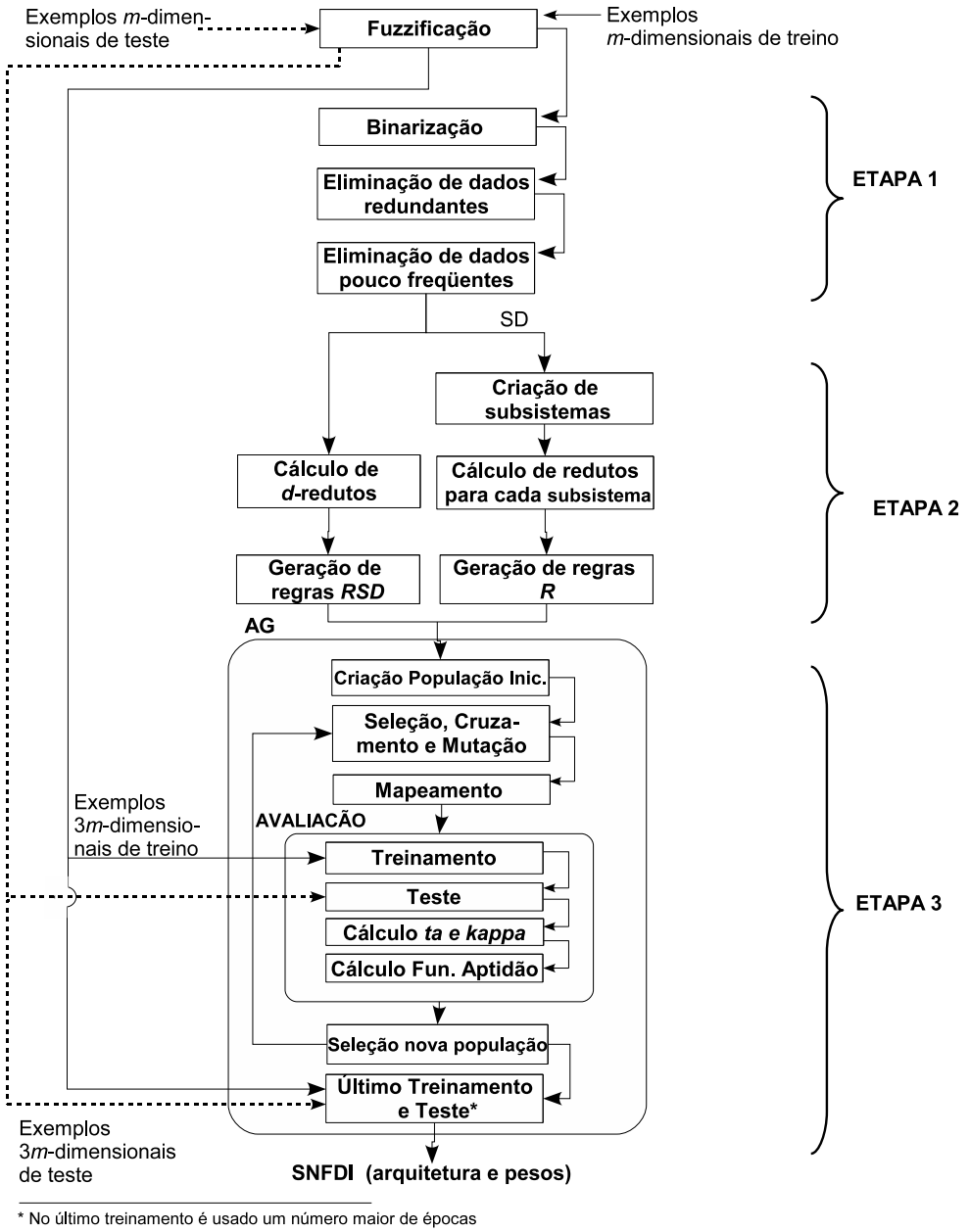


Figura 4.8: Etapas da metodologia de inicialização para o SNFDI.

Capítulo 5

Experimentação e Resultados

5.1 Considerações Iniciais

Neste capítulo são apresentados os resultados da experimentação da metodologia de inicialização sobre três conjuntos de dados: Diabetes, Coração e Vogal[1]. Na Seção 5.2 faz-se uma sucinta introdução às medidas de precisão da classificação utilizadas neste trabalho. Na Seção 5.3 são descritos os experimentos realizados, baseados na metodologia de avaliação de algoritmos de aprendizado, denominada validação cruzada. Na Seção 5.4, são apresentados o conjunto Diabetes e seus resultados de classificação, comparando o desempenho do SNFDI, obtido mediante a aplicação da metodologia de inicialização, com o obtido sem o auxílio da metodologia em questão. Esses resultados também são comparados com os de outros classificadores reconhecidos da literatura, conforme publicados por Michie et al. [MST94]. Também são apresentados alguns resultados parciais da evolução do AG, incluídos para complementar o entendimento do AG. Nas Seções 5.5 e 5.6, são apresentados o conjunto de dados Coração e Vogal e seus respectivos resultados, seguindo o mesmo esquema utilizado para o conjunto Diabetes. Para o conjunto Vogal, foram apresentados os resultados publicados por Mitra et al. [MP94], Banerjee et al. [BMP98] e Oliveira [Oli06].

A metodologia de inicialização foi implementada na linguagem C++ devido a sua característica de gerar código executável rápido.

5.2 Medidas de Precisão da Classificação

Um método para se avaliar o desempenho de um algoritmo de classificação consiste em comparar seus resultados de classificação com os resultados obtidos por outros classificadores. A avaliação da precisão pode ser feita por meio de uma matriz de confusão, na qual é mostrado o percentual de classificações corretas preditas pelo classificador em oposição ao percentual de classificações preditas por um especialista. Na Tabela 5.2, é

[1] Um resumo dos conjuntos de dados utilizados é apresentado no Apêndice C.

mostrada uma matriz de confusão, considerando um problema de classificação com duas classes, em que A é o percentual de exemplos corretamente classificados como sendo da classe c_1, B é o percentual de exemplos da classe c_2 erroneamente classificados como sendo da classe c_1, E é o percentual de exemplos corretamente classificados como sendo da classe c_2 e D é o percentual de exemplos da classe c_1 erroneamente classificados como sendo da classe c_2. A diagonal principal da matriz de confusão indica a concordância entre o classificador e o especialista.

	Classificador		
Especialista	c_1	c_2	Total
c_1	A	D	$A+D$
c_2	B	E	$B+E$
Total	$A+B$	$D+E$	1.0

Tabela 5.1: Matriz de confusão genérica.

Com base na matriz de confusão, várias medidas de precisão podem ser calculadas, entre elas, a taxa de acerto e o coeficiente *kappa*. A soma da diagonal da matriz é a taxa de acerto. O coeficiente *kappa* é uma medida que considera todos os elementos da matriz de confusão, e é definido por,

$$kp = \frac{p_o - p_c}{1 - p_c} \tag{5.1}$$

em que, $p_o = A + B$ e $p_c = (A+B)(A+D) + (D+E)(B+E)$ indicam, respectivamente, a proporção de unidades que concordam e a proporção de unidades para a concordância esperada.

5.3 Descrição dos Experimentos

Neste trabalho, a validação cruzada, foi utilizada para encontrar os valores dos parâmetros que intervém na metodologia, e assim determinar os melhores SNFDIs. Também foi utilizada para avaliar a metodologia proposta, possibilitando, a comparação com o SNFDI inicializado sem a aplicação da mesma, e com outros classificadores na literatura. Os propósitos para os quais é utilizada a validação cruzada são cumpridos, realizando dois experimentos que serão descritos a seguir.

5.3.1 Experimento 1

Este experimento aplica a validação cruzada para determinar os valores dos parâmetros de inicialização, mostrados na Tabela 5.2, e assim, mediante a aplicação da metodologia proposta, encontrar os melhores SNFDIs, i.e., aqueles com as maiores taxas de acerto.

A taxa de aprendizagem (η) e de momento (α) também constituem parâmetros de inicialização para o treinamento do SNFDI; porém, este foi implementado para que os valores

Parâmetro	Descrição
s	Sobreposição das funções de pertinência usada na fuzzificação dos dados (ver Seção 2.5.1).
pc	Ponto de corte utilizado para a binarização dos exemplos fuzzificados de treinamento (ver Seção 4.1.1).
ϵ	O número máximo de evoluções que o AG vai executar.
ρ	O número de indivíduos da população do AG.

Tabela 5.2: Parâmetros de inicialização.

de ambas as taxas diminuíssem no decorrer das épocas. Por esse motivo, estas não são consideradas na inicialização. Também não é considerada a taxa mutação, a qual em todos os experimentos foi de 10%. Este valor, que é maior que os usados comumente na literatura, foi escolhido, visando garantir a diversidade da população. Com o propósito de conservar o melhor indivíduo, o operador de mutação deste AG não substitui o indivíduo a ser mutado. Na verdade, este operador clona o indivíduo, e posteriormente, realiza a mutação sobre o clone criado; por conseguinte, apesar de se usar uma alta taxa de mutação, esta não impede a convergência do AG.

Antes de expor a forma em que a validação cruzada foi aplicada neste experimento, levem-se em conta as seguintes considerações:

- O SNFDI que é inicializado usando a metodologia proposta, será chamado de SNFDI evoluído.
- O conjunto de dados está dividido em k *folds* (subconjuntos).
- Neste experimento, quando se fizer menção a um determinado número de *fold*, estar-se-á indicando que esse *fold* é usado em todos os testes realizados no AG, inclusive no teste final; e a união dos restantes, na geração de regras de dependência e nos treinamentos de AG, inclusive no treinamento final, conforme exibido na Figura 4.8, correspondente às etapas da metodologia.
- Nos experimentos, os valores tanto para ϵ quanto para ρ foram 10, 15, 20. Os valores para s e pc foram 0.2, 0.4, 0.6, 0.8, salvo para o conjunto Vogal que para pc usou os valores 0.3, 0.5, 0.7, 0.9.

No Experimento 1, aplica-se a validação cruzada como descrito a seguir:

Começando com ϵ e ρ fixos faça o seguinte:

1. Para cada combinação de valores de s e pc:
  1.1 Para cada *fold*
    Aplicar metodologia de inicialização.
  1.2 Calcular *ta* média de todos *folds*.
2. Determinar a maior *ta* média.

3. Fixar s e pc correspondente ao maior ta e para cada combinação de valores de ϵ e ρ, repetir 1.1 e 1.2 e 2, que dará como resultado ϵ e ρ correspondente ao maior ta.

Em resumo, obtém-se para os parâmetros citados, os valores que conduzem à obtenção da maior ta média sobre todos os *folds*. Os k sistemas que contribuíram para a obtenção dessa média, serão chamados *SNFDI evoluídos*, e possuem arquiteturas particulares determinadas pela metodologia de inicialização. As informações dessas arquiteturas, bem como seus pesos iniciais com os quais iniciaram seu treinamento são armazenados para ser usados no Experimento 2.

5.3.2 Experimento 2

Neste experimento avalia-se as arquiteturas obtidas no experimento anterior a fim de apreciar como se comportam em média, e assim poder comparar seus resultados com os obtidos por outros classificadores, entre eles, o próprio SNFDI inicializado sem a aplicação da metodologia proposta.

Nem todas as arquiteturas são avaliadas e somente são escolhidas aquelas que dão a diversidade suficiente a este experimento. Deste modo, das k arquiteturas obtidas no experimento anterior, uma determinada arquitetura é escolhida desde que possua um número único de neurônios na camada intermediária. Das arquiteturas que possuem um mesmo número de neurônios é escolhida aquela que fornece a maior taxa de acerto.

Posteriormente, para cada uma das arquiteturas escolhidas, e usando os pesos iniciais com os quais estas começaram seu treinamento, aplica-se a validação cruzada, usando os *folds* determinados no Experimento 1, como descrito a seguir:

Para cada arquitetura escolhida realiza-se o treinamento com a união de todos os *folds*, exceto um. A sua precisão da classificação (taxa de acerto) é medida, testando-a com o *fold* que não é usado no treinamento. Este procedimento (treinamento e teste) é repetido k vezes, cada vez tomando $k-1$ *folds* diferentes para o treinamento. Por último, calcula-se a média da taxa de acerto sobre todos os *folds*. Este valor constitui uma média do desempenho do SNFDI para uma arquitetura, em particular, encontrada pela metodologia de inicialização no Experimento 1. Assim, o propósito desta primeira parte do Experimento 2, é encontrar taxa média de acerto do SNFDI evoluído, para as arquiteturas que conduziram à obtenção da maior taxa média de acerto no Experimento 1.

A validação cruzada é reaplicada usando os *folds* já determinados. Desta vez, inicializa-se as arquiteturas escolhidas para determinar as taxas de acerto do SNFDI evoluído, (i.e., as escolhidas na primeira parte do Experimento 2), com pesos aleatórios. Finalmente, para cada arquitetura é calculada sua respectiva taxa média de acerto sobre todos os *folds*. O SNFDI, como descrito neste parágrafo, será chamado de *SNFDI base*.

Este experimento tem como objetivo, além de comparar o SNFDI com outros classificadores, comparar a taxa média de acerto, obtidas pelo SNFDI base e o evoluído. Para isso, usam-se as arquiteturas com as quais se obteve a maior taxa média de acerto no Ex-

perimento 1. Para o SNFDI base, inicializam-se os pesos aleatoriamente, e para o SNFDI evoluído, usam-se os pesos obtidos pela metodologia proposta, mais especificamente, os pesos iniciais com os quais as arquiteturas em questão, começaram seu treinamento.

Nas Tabelas 5.6, 5.11 e 5.16 são apresentados os resultados deste experimento para os conjuntos de dados Diabetes, Coração e Vogal, respectivamente.

5.4 Conjunto de Dados Diabetes

O conjunto de dados Diabetes foi doado por Vincent Sigillito do Applied Physics Laboratory (Johns Hopkins University, Laurel, Maryland, USA) ao UCI Machine Learning Repository[2]. Seus exemplos constituíram-se de uma seleção de casos do banco de dados do National Institute of Diabetes and Digestive and Kidney Diseases (Bethesda, Maryland, USA), e possuem 1 atributo de classe que indica a presença da doença e 8 atributos condicionais que correspondem a medidas fisiológicas e a resultados de exames médicos para diagnosticar a diabetes. Este conjunto possui 768 exemplos dos quais 500 correspondem à classe 1 (interpretada como "diagnóstico negativo para a diabetes") e 268 à classe 2 (interpretada como "diagnóstico positivo").

5.4.1 Resultados Publicados por Michie et al.

Michie et al. [MST94] forneceram os resultados de classificação de 22 classificadores, entre os quais estão diversas árvores de decisão, o PMC, o *Naive Bayes*, etc. Na Tabela C.1, são mostrados os 5 classificadores[3] que forneceram os melhores resultados, a saber, Logdisc (discriminador logístico), DIPOL92 (classificador híbrido que integra um discriminador logístico e outros métodos estatísticos não paramétricos), Discrim (discriminador linear), SMART (classificador por busca de projeção) e RBF (RNA de função de base radial). O classificador que forneceu os piores resultados foi o k-NN (k- vizinhos mais próximos) com uma taxa de acerto de 0.6760. Nesses experimentos foi usada a validação cruzada de 12 *folds*.

Michie et al. [MST94] manifestaram que este conjunto de dados é difícil de se classificar, e que nenhum dos algoritmos funciona excepcionalmente bem. Eles atribuíram essa dificuldade ao fato de que o atributo de decisão é realmente uma forma binarizada de outro atributo, que é um forte indicativo de certos tipos de diabetes, porém, não tem uma correspondência unívoca com a condição médica da presença da doença. Finalmente, concluíram que é razoável afirmar que os atributos não predizem corretamente o diagnóstico da diabetes. Essa dificuldade na classificação foi a principal motivação para considerá-la nos testes deste trabalho.

[2]O conjunto Diabetes está disponível no UCI Machine Learning Repository, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[3]Informações mais detalhadas sobre os classificadores apresentados podem ser encontradas em [MST94].

Ranking	Classificador	Taxa de acerto (*ta*)
1	Logdisc	0.7770
2	DIPOL92	0.7760
3	Discrim	0.7750
4	SMART	0.7680
5	RBF	0.7570

Tabela 5.3: Resultados publicados por Michie et al. [MST94] para o conjunto Diabetes.

5.4.2 Resultados da Metodologia e Comparação

Os experimentos com este conjunto de dados foram realizados com 12 *folds*, pois um dos objetivos desta seção é viabilizar a comparação com os resultados apresentados por Michie et al. [MST94].

Experimento 1

Foram realizados vários experimentos mudando os valores de pc, s, ϵ e ρ, conforme descrito na Seção 5.3.1. Na Tabela 5.4, são apenas mostrados os resultados mais significativos. Estes foram obtidos com $\epsilon = 10$, $\rho = 20$. Utilizaram-se 2000 épocas para o treinamento final, i.e., para o treinamento de todos os indivíduos da população final do AG; e 300 épocas para os treinamentos no processo de evolução do AG.

s	pc	*Fold* 1		*Fold* 2		*Fold* 3		*Fold* 4		
		$n2$	ta	$n2$	ta	$n2$	ta	$n2$	ta	
0.4	0.4	8	0.7619	6	0.7538	8	0.7692	6	0.7384	
	0.6	5	0.7778	5	0.7230	5	0.7538	4	0.7538	
	0.8	6	0.7460	8	0.7538	4	0.7230	9	0.6923	
0.6	0.4	4	0.7142	6	0.7230	5	0.7076	5	0.7076	
	0.6	**6**	**0.7619**	**5**	**0.7692**	**8**	**0.7891**	**7**	**0.7538**	
		Fold 5		*Fold* 6		*Fold* 7		*Fold* 8		
0.4	0.4	7	0.7846	5	0.7813	5	0.7500	4	0.7813	
	0.6	7	0.8153	3	0.7969	6	0.7344	6	0.8125	
	0.8	8	0.7230	5	0.7188	5	0.7031	5	0.7344	
0.6	0.4	5	0.7538	4	0.7031	6	0.7500	9	0.7656	
	0.6	**6**	**0.8308**	**7**	**0.8281**	**6**	**0.7656**	**5**	**0.8281**	
		Fold 9		*Fold* 10		*Fold* 11		*Fold* 12		ta média
0.4	0.4	10	0.7500	5	0.7142	10	0.7936	4	0.7460	0.7604
	0.6	8	0.7656	5	0.7142	9	0.8095	7	0.7619	0.7682
	0.8	4	0.7344	4	0.6984	9	0.7301	5	0.7460	0.7253
0.6	0.4	6	0.7344	5	0.7460	3	0.7301	7	0.7142	0.7291
	0.6	**6**	**0.7969**	**6**	**0.7302**	**5**	**0.8413**	**6**	**0.7778**	**0.7894**

Tabela 5.4: Resultados para o conjunto Diabetes, usando o SNFDI evoluído.

Na Tabela 5.4, a primeira e segunda coluna correspondem aos valores da sobreposição (s) e do ponto de corte (pc), utilizados nos experimentos. Nas colunas subseqüentes são mostrados o número encontrado de neurônios ($n2$) e a taxa de acerto atingida (ta), para cada *fold*. Na última coluna, é mostrada a média da taxa de acerto sobre os 12 *folds*. Por exemplo, para o *fold* 1, usando $s = 0.6$, $pc = 0.6$, $\epsilon = 10$, $\rho = 20$, 300 épocas no AG e 2000 épocas no treinamento final, foi encontrada uma arquitetura de RNA com 6 neurônios na camada intermediária, atingindo-se uma taxa de acerto de 0.7619.

Apesar destes resultados terem sido feitos para determinar os valores dos parâmetros, pode-se analisar seus resultados.

Observa-se que os melhores resultados, que aparecem em destaque na Tabela 5.4, foram encontrados com $pc = 0.6$ e $s = 0.6$. Obteve-se um número de neurônios $n2$ flutuante entre 5, 6, 7 e 8, atingido-se uma ta média de 0.7894. Nota-se que este resultado supera em precisão aos publicados por Michie et al. [MST94] (ver Tabela C.1), nos que Logdisc obteve uma ta de 0.7770. Lembre-se que este valor foi a melhor taxa de acerto dos 22 classificadores apresentados. Para o SNFDI evoluído, outras taxas de acerto significativas, foram obtidas com $s = 0.4$ e $pc = 0.4$ e com $s = 0.4$ e $pc = 0.6$.

Resultados Parciais do AG : A Tabela 5.4 apresenta o melhor resultado obtido, entre todos os realizados, (i.e., o atingido usando $s = 0.6$, $pc = 0.6$, $\epsilon = 10$ e $\rho = 20$), junto a outros resultados significativos obtidos ao variar pc e s. O objetivo, nesta seção, é apresentar alguns resultados parciais da evolução do AG, obtidos no Experimento 1, visando mostrar o efeito das variações de ϵ e ρ.

Assim, na Tabela 5.5, mostram-se as taxas médias de acerto sobre os 12 *folds*, variando os parâmetros ϵ e ρ e fixando $s = 0.6$ e $pc = 0.6$. Escolheu-se apresentar estes resultados, i.e., usando $s = 0.6$ e $pc = 0.6$ fixos, pois com esses valores atingiu-se as melhores taxas médias de acerto, entre elas, a mostrada na coluna "ta média", que aparece em destaque na Tabela 5.4.

ϵ	ρ	ta média	ϵ	ρ	ta média	ϵ	ρ	ta média
10	10	0.7651	15	10	0.7785	20	10	0.7792
10	15	0.7701	15	15	0.7810	20	15	0.7889
10	**20**	**0.7894**	15	20	0.7894	20	20	0.7893

Tabela 5.5: Resultados do SNFDI evoluído para o conjunto Diabetes, mudando ϵ e ρ ($pc = 0.6$, $s = 0.6$).

De maneira geral, observa-se que, para um número fixo de indivíduos da população, houveram melhoras na taxa de acerto ao aumentar o número de evoluções. Também, pode-se notar que para um número fixo de evoluções, a taxa média de acerto melhorou ao aumentar o tamanho da população. Contudo, embora existam melhoras nas taxas de acerto atingidas ao mudar ϵ e ρ, vale frisar, que estas não foram tão marcantes quanto as observadas ao variar pc e s[4].

[4]Ver a Tabela 5.4 para observar ditas variações.

Para os experimentos executados com $s = 0.6$, $pc = 0.6$, $\epsilon = 10$, e $\rho = 20$, e cujos resultados foram mostrados na linha em destaque da Tabela 5.4, é apresentado, na Figura 5.1, o comportamento do *fitness* médio dos indivíduos de uma mesma geração, para cada um dos 12 *folds*. Os eixos horizontais das Figuras 5.1(a) a 5.1(l), representam as seqüências temporais de 10 gerações, e os eixos verticais, os *fitness* médios dos indivíduos da população para uma mesma geração.

Na Figura 5.1(a), é mostrada a variação do *fitness* médio no decorrer do processo de evolução, até se obter a arquitetura de 6 neurônios que forneceu a taxa de acerto de 0.7619, conforme mostrado na coluna *Fold* 1 da Tabela 5.4.

Nota-se que para todos os *folds*, o *fitness* melhorou à medida que as gerações aumentaram, convergindo, na maioria dos casos, a partir da oitava geração.

Experimento 2

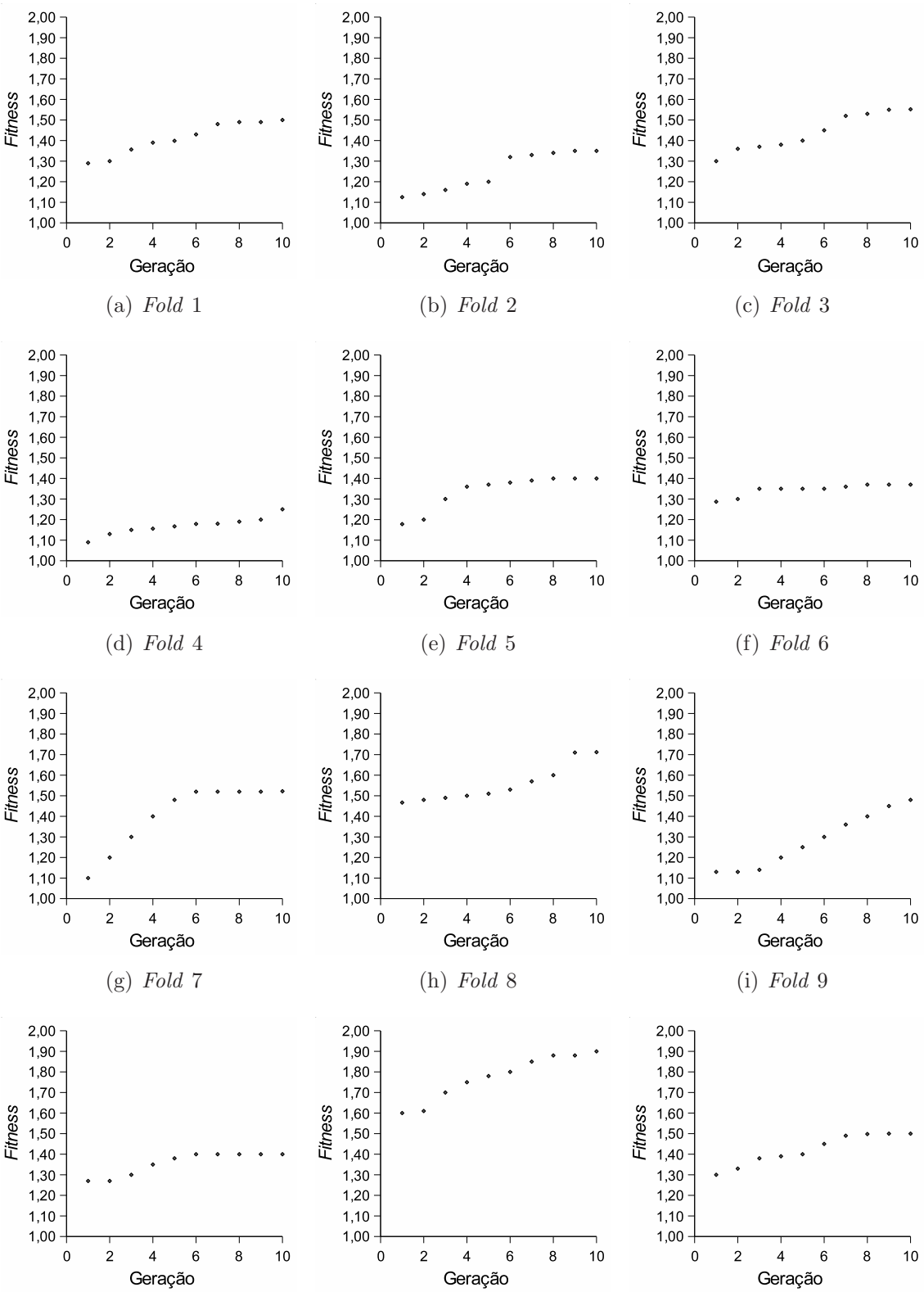
Neste experimento busca-se avaliar (determinar a capacidade média de generalização) algumas das arquiteturas encontras no Experimento 1. Na Tabela 5.4, nota-se que algumas das arquiteturas correspondentes à maior taxa média de acerto, possuem o mesmo número de neurônios na camada intermediária, como visto nos *folds* 1, 5, 7, 9, 10 e 12. Nesse caso, foi escolhida aquela que forneceu a maior taxa de acerto, ou seja, a arquitetura do *fold* 5, cuja taxa de acerto foi maior que as das outras com 6 neurônios. A mesma situação ocorre para os *folds* 2, 8 e 11, e para os *folds* 4 e 6, nos quais suas arquiteturas possuem 5 e 7 neurônios, respectivamente. Nesses casos, escolheram-se os *folds* 11 e 6, pois suas arquiteturas atingiram as maiores taxas de acerto. Para o *fold* 3, tomou-se a arquitetura com 8 neurônios, cuja taxa de acerto foi 0.7891.

Posteriormente, para cada arquitetura escolhida, que possui o número de neurônios determinados pela metodologia de inicialização, foi aplicada a validação cruzada, conforme descrito na Seção 5.3.2, usando para o treinamento os pesos iniciais com os que a arquitetura escolhida iniciou seu treinamento, no Experimento 1. A taxa média de acerto sobre os 12 *folds* foi calculado para cada arquitetura. Estes resultados aparecem na coluna do SNFDI evoluído na Tabela 5.6. Posteriormente, para as mesmas arquiteturas, mas fazendo os pesos iniciais aleatórios foi reaplicada, a validação cruzada, obtendo-se as taxas de acerto apresentadas na coluna do SNFDI base da Tabela 5.6.

Fold	$n2$	SNFDI base *ta* média	SNFDI evoluído *ta* média
11	5	0.7616 ± 0.032	0.8034 ± 0.048
5	6	0.7585 ± 0.035	0.7982 ± 0.062
6	7	0.7697 ± 0.056	0.8008 ± 0.056
3	8	0.7616 ± 0.027	0.8034 ± 0.054

Tabela 5.6: Comparação entre o SNFDI base e evoluído para o conjunto Diabetes.

O Experimento 2 compara por meio da Tabela 5.6, o desempenho do SNFDI base (treinado



(a) *Fold* 1 (b) *Fold* 2 (c) *Fold* 3

(d) *Fold* 4 (e) *Fold* 5 (f) *Fold* 6

(g) *Fold* 7 (h) *Fold* 8 (i) *Fold* 9

(j) *Fold* 10 (k) *Fold* 11 (l) *Fold* 12

Figura 5.1: Evolução dos indivíduos do AG para conjunto Diabetes.

com uma arquitetura cujo número de neurônios foi especificado pela aplicação da metodologia de inicialização, mas com os pesos inicializados aleatoriamente) e do SNFDI evoluído (com arquitetura e pesos iniciais determinados pela metodologia em estudo).

Observa-se que a inicialização dos pesos do SNFDI, obtida pela aplicação da teoria de *rough sets* e de AGs, o que caracteriza o SNFDI evoluído, obteve melhores taxas de acerto em todas as arquiteturas mostradas na Tabela 5.6. Houve, em média, uma melhoria de 3.5% na taxa de acerto. Ao comparar a Tabela C.1 dos resultados de Michie [MST94], também se observa melhores taxas de acerto para o SNFDI base, e ainda melhores, para o SNFDI evoluído.

Pode ser observado que a maior *ta* média para o SNFDI base, foi 0.7697 para $n2 = 7$. O melhor resultado para o SNFDI evoluído foi obtido com um número menor de neurônios (5 neurônios), atingindo-se uma *ta* média de 0.8034. O número de épocas estabelecido para todos os treinamentos neste experimento, foi de 2000, e o valor de s para a fuzzificação foi de 0.6, ou seja, utilizaram-se o mesmo número de épocas estabelecidas, e o mesmo o valor de s encontrado no Experimento 1.

5.5 Conjunto de Dados Coração

Este conjunto de dados foi doado por Robert Detrano do V.A. Medical Center (Long Beach, California) ao UCI Machine Learning Repository[5], e é parte de uma coleção de bancos de dados da University of California. Seus exemplos possuem 13 atributos condicionais que provém de várias medidas fisiológicas e do resultado de diversos exames para diagnosticar uma doença cardíaca; e 1 atributo de classe que indica a ausência ou presença de uma doença cardíaca. Este conjunto de dados possui 270 exemplos, dos quais 150 pertencem à classe 1 (diagnóstico negativo) e 120 pertencem à classe 2 (diagnóstico positivo).

5.5.1 Resultados Publicados por Michie et al.

O conjunto Coração é utilizado junto com uma matriz de custo de erro da classificação, que é mostrada na Tabela 5.7. Esta matriz define os custos de classificações errôneas, i.e., penalidades por falhas incorridas por um modelo (classificador). Assim, o propósito desta matriz no conjunto Coração, é dar um maior custo aos falsos negativos; i.e., aos exemplos de teste que pertencendo à classe 2 (presença da doença), foram classificados, pelo modelo, como sendo da classe 1 (ausência da doença).

A matriz de custo é usada da seguinte maneira: Após o treinamento, é calculada sua matriz de confusão, que é multiplicada ponto a ponto pela matriz de custo; ou seja, são multiplicados os valores das entradas (1,1) de ambas as matrizes; depois, são multiplicadas as entradas (1,2) de ambas as matrizes, e assim por diante. Os resultados parciais são

[5]O conjunto Coração está disponível no UCI Machine Learning Repository, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

	Classificador	
Referência	Ausência	Presença
Ausência	0	1
Presença	5	0

Tabela 5.7: Matriz de custo do erro para o conjunto Coração.

somados, encontrando-se assim, o custo associado ao erro da classificação.

Nos testes de Michie et al. [MST94] foram avaliados classificadores que consideraram o custo de erro, e também foram avaliados outros que não o consideraram. Dado que o SNFDI pertence aos últimos, na Tabela 5.8[6] são apresentados os melhores 5 classificadores entre aqueles que não levaram em conta dito custo, sendo o PCM (Perceptron de Múltiplas Camadas) aquele que forneceu os melhores resultados (menor custo), seguido de LQV (*Learning Vector Quantization*), Kohonen (rede neural de Kohonen), AC^2 (tipo de árvore de decisão) e finalmente, de CN2 (modelo baseado em regras)[7].

Ranking	Classificador	Custo
1	PCM	0.574
2	LQV	0.600
3	Kohonen	0.693
4	AC^2	0.744
5	CN2	0.767

Tabela 5.8: Resultados publicados por Michie et al. [MST94] para o conjunto Coração.

Para os experimentos foram usados 9 *folds*.

5.5.2 Resultados da Metodologia e Comparação

Foram usados 9 *folds*, visando possibilitar a comparação com os resultados mostrados por Michie et al. [MST94].

Experimento 1

Consoante ao descrito na Seção 5.3.1, foram realizados vários experimentos mudando os parâmetros s, pc, ρ e ϵ. Os resultados mais relevantes são mostrados na Tabela 5.9. Estes

[6]Nos experimentos Michie et al., os classificadores que consideraram o custo de erro forneceram os melhores resultados, aparecendo o PCM na sexta posição. O *ranking* da Tabela 5.8 foi realizado neste texto com base no custo do erro fornecido por Michie et al.

[7]Informações mais detalhadas sobre os classificadores apresentados podem ser encontradas em [MST94].

foram alcançados com $\epsilon = 15$ e $\rho = 20$. O número de épocas empregado foi de 200 para os treinamentos no AG, e de 2000 para o treinamento final. Observa-se, por sua vez, que para $pc = 0.6$ e $s = 0.6$, atingiu-se uma taxa média de acerto igual a 0.853, e um custo médio igual a 0.4193. Comparando esse resultado com os apresentados por Michie et al. [MST94] e resumidos na Tabela 5.8, observa-se que o SNFDI evoluído é *rankeado* na primeira posição.

s	pc	*Fold* 1			*Fold* 2			*Fold* 3		
		$n2$	ta	Custo	$n2$	ta	Custo	$n2$	ta	Custo
0.6	0.4	6	0.8387	0.6774	7	0.8387	0.5484	5	0.7419	0.7742
	0.6	**5**	**0.9032**	**0.3548**	**10**	**0.8387**	**0.5484**	**3**	**0.7742**	**0.3548**
		Fold 4			*Fold* 5			*Fold* 6		
s	pc	$n2$	ta	Custo	$n2$	ta	Custo	$n2$	ta	Custo
0.6	0.4	4	0.8710	0.3871	3	0.8710	0.2580	10	0.7097	0.9355
	0.6	**4**	**0.9355**	**0.3226**	**2**	**0.8710**	**0.2580**	**3**	**0.7419**	**1.0323**
		Fold 7			*Fold* 8			*Fold* 9			Média	
s	pc	$n2$	ta	Custo	$n2$	ta	Custo	$n2$	ta	Custo	ta	Custo
0.6	0.4	5	0.8065	0.7097	5	0.8710	0.3871	3	0.8710	0.3871	0.8244	0.5627
	0.6	**3**	**0.8065**	**0.3226**	**6**	**0.9032**	**0.2258**	**3**	**0.9032**	**0.3548**	**0.8530**	**0.4193**

Tabela 5.9: Resultados para o conjunto Coração, usando o SNFDI evoluído.

Considerando os classificadores que levaram em conta o custo do erro, o SNFDI é *rankeado* na quinta posição; contudo, destaca-se o fato de que o SNFDI evoluído tenha fornecido os melhores resultados entre os classificadores que não levaram em conta, nos seus mecanismos de aprendizado, o custo do erro da classificação.

Resultados Parciais do AG: As informações apresentadas nesta seção correspondem a alguns resultados parciais do Experimento 1, visando mostrar os efeitos ao variar ϵ e ρ.

Mudando os parâmetros ϵ e ρ, e fixando $s = 0.6$ e $pc = 0.6$, obteve-se os resultados mostrados na Tabela 5.10. O valor que aparece em destaque corresponde à melhor taxa média de acerto obtida, que também é mostrada na Tabela 5.9. Foram escolhidos $s = 0.6$ e $pc = 0.6$, pois com estes valores, obtiveram-se os melhores resultados de classificação.

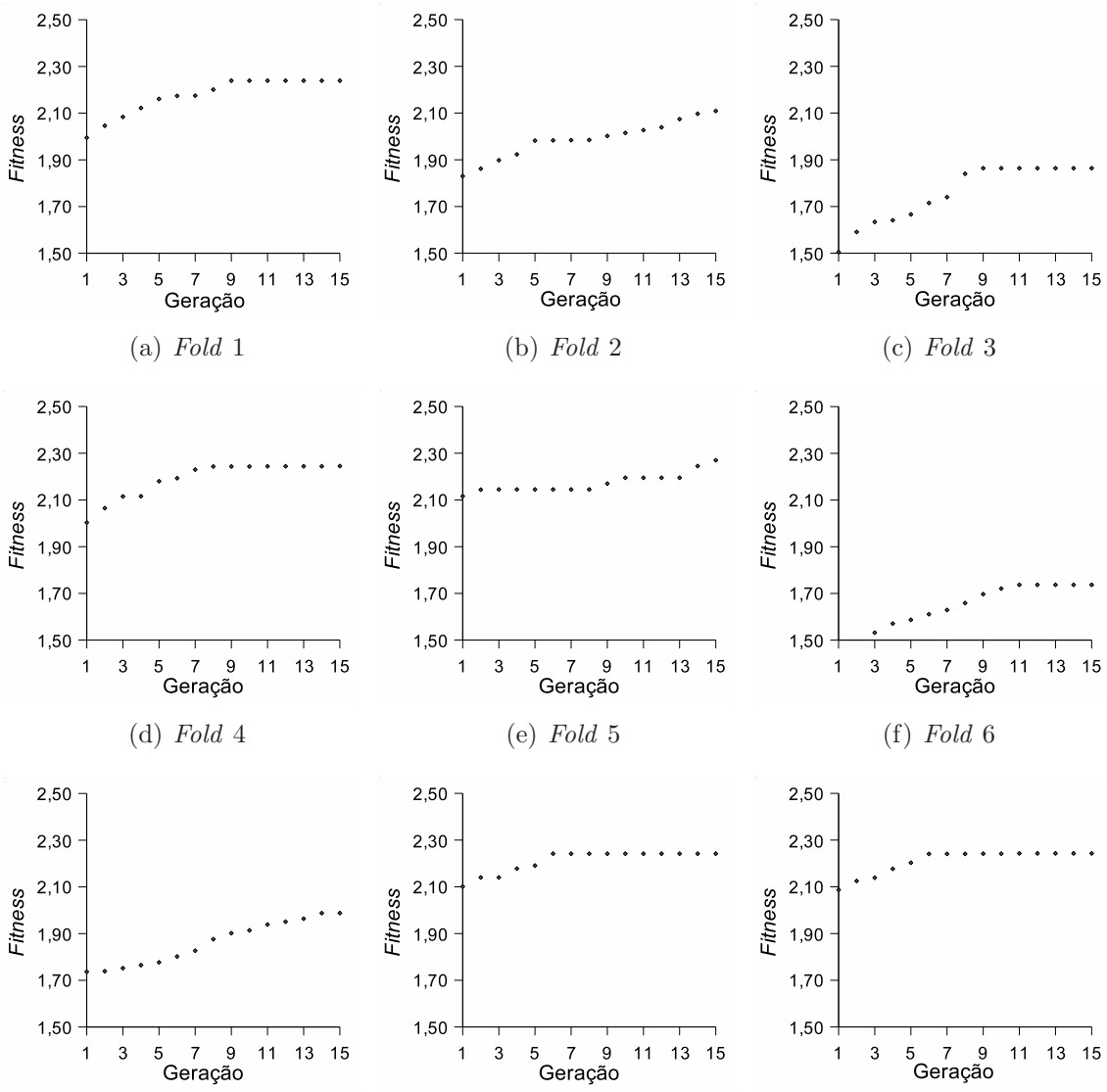
Um aspecto que merece ser mencionado foi que, mudando o número de evoluções do AG e o tamanho da população, as variações na taxa acertos não foram tão marcantes quanto as obtidas mudando os valores da sobreposição e do ponto de corte. Isso leva afirmar que estes parâmetros são mais influentes que ϵ e ρ, na metodologia proposta.

ϵ	ρ	ta	ϵ	ρ	ta média	ϵ	ρ	ta média
10	10	0.8065	15	10	0.8280	20	10	0.8351
10	15	0.8387	15	15	0.8528	20	15	0.8499
10	20	0.8315	15	**20**	**0.8530**	20	20	0.8527

Tabela 5.10: Resultados do SNFDI evoluído para o conjunto Coração, mudando ϵ e ρ ($pc = 0.6$, $s = 0.6$).

Na Figura 5.2, para cada um dos 9 *folds*, são mostrados os gráficos da evolução, os quais correspondem aos resultados obtidos fixando $s = 0.6$, $pc = 0.6$, $\epsilon = 15$ e $\rho = 20$. Tais

resultados foram apresentados na linha em destaque da Tabela 5.9. Cada valor no eixo *Fitness* é a média do *fitness* obtida sobre todos os indivíduos da mesma geração.



(a) *Fold* 1 (b) *Fold* 2 (c) *Fold* 3

(d) *Fold* 4 (e) *Fold* 5 (f) *Fold* 6

(g) *Fold* 7 (h) *Fold* 8 (i) *Fold* 9

Figura 5.2: Evolução dos indivíduos do AG para conjunto Coração.

Observa-se que para todos os *folds*, o índice de aptidão melhora com o decorrer das gerações. Pode ser observado, também, que este se estabiliza a partir da décima primeira geração.

Experimento 2

Na Tabela 5.11, são mostrados os resultados, tanto para o SNFDI base quanto para o SNFDI evoluído, obtidos seguindo a descrição do Experimento 2. Estabeleceram-se 2000 épocas para todos os treinamentos realizados neste experimento e usou-se $s = 0.6$ para a

fuzzificação.

Fold	$n2$	SNFDI base		SNFDI evoluído	
		ta média	Custo médio	*ta* média	Custo médio
5	2	0.8136 ± 0.040	0.5018 ± 0.022	0.8280 ± 0.033	0.4875 ± 0.029
6	3	0.8172 ±0.039	0.5269 ± 0.038	0.8172 ± 0.039	0.5412 ± 0.040
4	4	0.7993 ± 0.036	0.6022 ± 0.029	0.8029 ± 0.031	0.5269 ± 0.038
1	5	0.7921 ± 0.036	0.5520 ± 0.024	0.8280 ± 0.033	0.4731 ± 0.027
8	6	0.7921 ± 0.051	0.5806 ± 0.039	0.8315 ± 0.040	0.4839 ± 0.045
2	10	0.8186 ± 0.046	0.4875 ± 0.026	0.8423 ± 0.021	0.4731 ± 0.032

Tabela 5.11: Comparação entre o SNFDI base e evoluído para o conjunto Coração.

Observa-se que o SNFDI inicializado, usando a metodologia proposta, atingiu maiores taxas de acerto que o SNFDI base. Em média, essas melhorias contribuíram para aumentar a precisão do SNFDI em 2%. No que diz respeito ao custo do erro, em geral, o SNFDI evoluído incorreu em custos menores que os do SNFDI base. No entanto, vale mencionar que essa diminuição no custo, é, estritamente, uma conseqüência das melhores taxas de acerto do evoluído.

No caso da arquitetura com 3 neurônios (ver linha 2 da Tabela 5.11), tanto o SNFDI evoluído quanto o base tiveram a mesma taxa de acerto, mas diferentes custos, sendo menor, o do SNFDI base. Isto equivale a dizer que apesar de ambos terem, em média, o mesmo número de exemplos classificados erroneamente, o número de falsos negativos diagnosticados pelo SNFDI base foi menor que o do evoluído.

A maior taxa de acerto para o SNFDI base foi de 0.8186, com uma arquitetura com 10 neurônios. Por sua vez, o SNFDI evoluído atingiu uma taxa de 0.8423 e 0.8315 com 10 e 6 neurônios, respectivamente. Pode-se ver, então, que o SNFDI evoluído gera arquiteturas que fornecem maior precisão, com um número menor de neurônios que o SNFDI base.

5.6 Conjunto de Dados Vogal

A classificação do conjunto de dados Vogal[8] foi explorado em diversos trabalhos de Mitra et al. [MP94] [MP92], e Banerjee et al. [BMP98]. A sua escolha foi motivada, visando a comparação com os resultados de classificação, obtidos nessas pesquisas, bem como, os obtidos por Oliveira [Oli06].

Esse conjunto de dados foi construído da seguinte forma: Amostras de voz de 6 vogais de um dialeto indiano chamado *Telugu* foram tomadas de um grupo de locutores masculinos com idades na faixa de 30 a 35 anos. Posteriormente, 3 atributos foram obtidos por meio de análise espectral dos sinais de voz. O conjunto de dados é formado por 871 exemplos e 4 atributos. Os primeiros 3 atributos correspondem às características espectrais do sinal

[8]O conjunto de dados Vogal foi criado pelo Machine Intelligence Unit no Indian Statistical Institute. Os dados podem ser encontrado no endereço http://www.isical.ac.in/∼miu.

de voz, F_1, F_2, e F_3; e o último, corresponde ao atributo de classe, que pode ser uma das 6 vogais do dialeto, a saber, ∂, a, i, u, e ou o. O número de exemplos por classe aparece na Tabela 5.12.

Classe	No. Exemplos	Classe	No. Exemplos
Classe 1 (∂)	72	Classe 4 (u)	151
Classe 2 (a)	89	Classe 5 (e)	207
Classe 3 (i)	172	Classe 6 (o)	180

Tabela 5.12: Número de exemplos do conjunto Vogal.

Na Figura 5.3, é mostrada a projeção bidimensional dos atributos F_1 e F_2, que são os que melhor discriminam as classes deste conjunto.

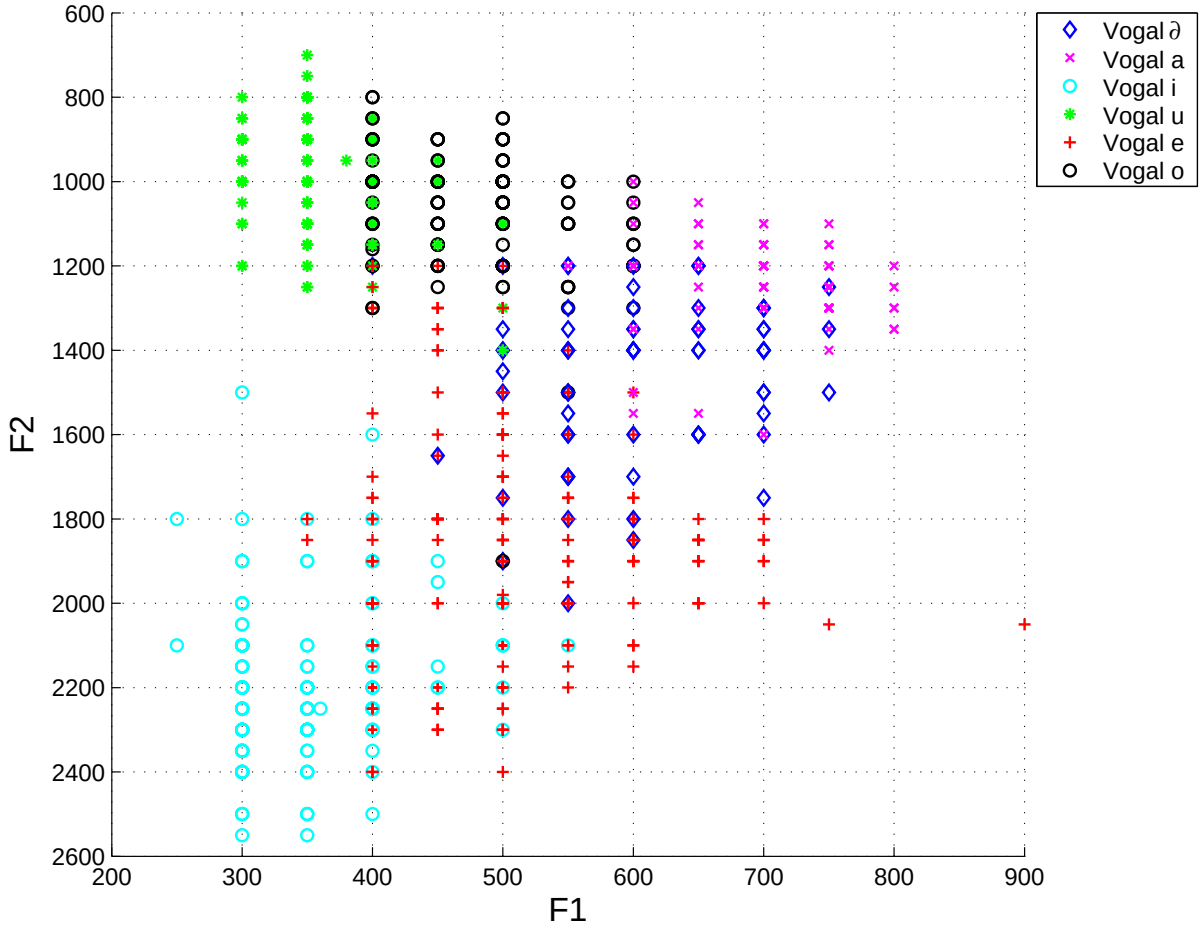


Figura 5.3: Conjunto de dados Vogal.

5.6.1 Resultados Publicados

Na Tabela 5.13, apresentam-se os resultados de classificação, publicados por Mitra [MP94], Banerjee et al. [BMP98] e Oliveira [Oli06] para este conjunto de dados.

Classificador	Taxa de acerto
Fuzzy-O	0.801
Logical-P	0.778
Logical-P2	0.861
Fuzzy-MLP	0.844
Rough-Fuzzy-MLP	0.860
SNFDI	0.801

Tabela 5.13: Resultados publicados para o conjunto Vogal.

Os primeiros quatro classificadores correspondem aos sistemas neuro-*fuzzy* propostos por Mitra et al. [MP94] [MP92]. O classificador *Fuzzy*-O emprega somas ponderadas e funções sigmóides no lugar das t-normas. *Logical*-P usa como t-norma e t-conorma, o produto e a soma probabilística, respectivamente. *Logical*-P2 é uma versão melhorada do *Logical*-P, e fuzzifica o atributo de classe do conjunto de dados tratado. *Fuzzy*-MLP, igual ao anterior modelo, realiza a fuzzificação do atributo de classe. *Rough-Fuzzy*-MLP é a versão *Fuzzy*-MLP que inclui a metodologia de inicialização de Banerjee [BMP98]. Nota-se que a melhora sobre a taxa de acerto do *Fuzzy*-MLP usando essa metodologia foi de 1.5%.

Entre os classificadores que não fuzzificam o atributo de classe, o *Fuzzy*-O obteve a maior taxa de acerto (0.801).

Na última linha, mostra-se a maior taxa de acerto, encontrada por Oliveira [Oli06] para o SNFDI. Nesse resultado, o SNFDI foi inicializado sem o auxílio da metodologia proposta, obtendo-se uma taxa de acerto de 0.800, com uma arquitetura de 17 neurônios na camada intermediária.

5.6.2 Resultados da Metodologia e Comparação

Nos experimentos deste conjunto de dados, usou-se a validação cruzada com 3 *folds*.

Experimento 1

Na Tabela 5.14, são mostrados os melhores resultados, obtidos com $\epsilon = 15$ e $\rho = 15$. Nota-se que a maior taxa média de acerto, 0.8090, foi atingida com $s = 0.8$ e $pc = 0.7$. Assim mesmo, o número de neurônios na camada intermediária, variou de 21 a 23 neurônios. O número de épocas utilizado nos experimentos foi de 550 para os treinamentos dentro do AG, e de 4500 para o treinamento final.

s	pc	*Fold* 1		*Fold* 2		*Fold* 3		Média ta
		$n2$	ta	$n2$	ta	$n2$	ta	
0.8	0.3	23	0.7705	19	0.6849	22	0.7671	0.7408
	0.5	25	0.7739	20	0.7671	20	0.7945	0.7785
	0.7	**23**	**0.8322**	**21**	**0.7808**	**22**	**0.8139**	**0.8090**
	0.9	24	0.7534	18	0.6369	23	0.7602	0.7168

Tabela 5.14: Resultados para o conjunto Vogal, usando o SNFDI evoluído.

A boa generalização, usando $s = 0.8$, se explica pelo fato de que o conjunto de dados Vogal tem um alto grau de sobreposição entre as suas classes. Portanto, atribuindo-se pequenos valores para s, não é representada a natureza deste conjunto. A sobreposição entre as classes de vogal pode ser apreciada na Figura 5.3, onde é mostrada a plotagem dos dados usando os atributos que mais discernem as classes.

Resultados Parciais do AG: As informações apresentadas nesta seção, correspondem a alguns resultados parciais do Experimento 1.

Vários experimentos foram realizados, fixando $pc = 0.7$ e $s = 0.8$, e variando ϵ e ρ. Estes forneceram os resultados mostrados na Tabela 5.15, nos quais, em geral, pode-se observar que a taxa média melhorou à medida que o número de evoluções foi acrescentado.

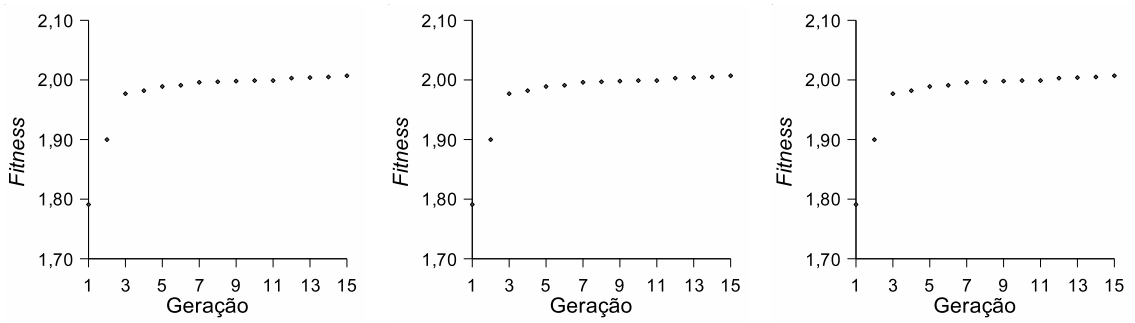
Os valores de $pc = 0.7$ e $s = 0.8$ foram escolhidos pois, como visto na Tabela 5.14, usando-os foram atingidos os melhores resultados de classificação.

Não existiram melhoras na taxa média para valores de ϵ e ρ maiores que 15, salvo no caso de $\epsilon = 20$ e $\rho = 20$; contudo, dado que o aumento na taxa de acerto não foi significativo, mas custoso, considerou-se este resultado (i.e., $ta = 0.8090$ com $\epsilon = 15$ e $\rho = 15$) como o melhor.

ϵ	ρ	ta	ϵ	ρ	ta	ϵ	ρ	ta
10	10	0.7979	15	10	0.8059	20	10	0.8013
10	15	0.7999	15	15	**0.8090**	20	15	0.8087
10	20	0.8082	15	20	0.8085	20	20	0.8092

Tabela 5.15: Resultados do SNFDI evoluído para o conjunto Vogal, mudando ϵ e ρ ($pc = 0.7$, $s = 0.8$).

Por outro lado, na Figura 5.4, é mostrada a evolução dos indivíduos ao longo de 15 gerações. As Figuras 5.4(a), 5.4(b) e 5.4(c) correspondem aos experimentos executados, usando $s = 0.8$, $pc = 0.7$, $\epsilon = 15$ e $\rho = 15$, mostrados na linha destacada da Tabela 5.14. Pode-se observar nas três figuras, que o *fitness* médio melhorou à medida que as gerações aumentaram. Também nota-se que as curvas do comportamento do *fitness* têm a mesma trajetória, i.e., a ascensão das mesmas é bem marcante nas primeiras gerações, e suave, nas últimas.



(a) *Fold* 1 (b) *Fold* 2 (c) *Fold* 3

Figura 5.4: Evolução dos indivíduos do AG para conjunto Vogal.

Experimento 2

Na Tabela 5.16, mostram-se as médias da taxa de acerto, atingidas pelo SNFDI base e evoluído. Para os treinamentos realizados, foram usadas 4500 épocas, e usou-se $s = 0.8$ para a fuzzificação.

Fold	$n2$	SNFDI base *ta* média	SNFDI evoluído *ta* média
2	21	0.7797 ± 0.039	0.8162 ± 0.020
3	22	0.7934 ± 0.036	0.8139 ± 0.025
1	23	0.8071 ± 0.036	0.8151 ± 0.020

Tabela 5.16: Comparação entre o SNFDI base e evoluído para o conjunto Vogal.

É importante notar que, as médias das taxas de acerto, atingidas executando o SNFDI evoluído, foram sempre melhores que as encontradas pelo SNFDI base, sendo a taxa de acerto, em média, 2% superior à obtida pelo SNFDI base.

Embora Banerjee et al. [BMP98] tenham conseguido uma taxa de acerto de 0.860, lembre-se que eles utilizaram como base, o *Fuzzy*-MLP, cuja taxa de acerto foi de 0.844, antes da aplicação da sua metodologia. Isto significa que a melhoria na precisão da classificação, foi de 1.5%, aproximadamente. Nesse sentido, pode-se dizer que os resultados de ambas as metodologias de inicialização são comparáveis.

Entre os classificadores propostos por Mitra et al. [MP94], o *Logical*-P2 atingiu a máxima taxa média, $ta = 0.8610$, superando ao SNFDI evoluído. No entanto, esse modelo e os outros que também o superam, entre eles, o *Fuzzy*-MLP, realizam um processo de fuzzificação do atributo de classe, situação, que o SNFDI ainda não explora. Porém, vale a pena frisar que a fuzzificação do atributo de classe é um dos aperfeiçoamentos planejados para o SNFDI.

O melhor resultado obtido por Oliveira[Oli06] se refere a uma arquitetura com 17 neurônios na camada intermediária, para a qual foi atingida uma taxa de acerto de 0.8000. Nota-se

que esta taxa de acerto foi menor que qualquer uma das taxas atingidas pelo SNFDI evoluído.

No entanto, as arquiteturas encontradas pela metodologia de inicialização, possuem um número maior de neurônios que a encontrada por Oliveira[Oli06]. Por conseguinte, pode-se afirmar que o custo computacional das primeiras é maior que o da segunda, pois no caso das arquiteturas encontradas pela metodologia proposta, existem mais parâmetros livres que devem ser calculados.

5.7 Considerações Finais

Neste capítulo, para cada um dos conjuntos de dados testados, foram apresentados os resultados mais significativos, com base nos quais, pode-se afirmar que, de maneira geral, a metodologia de inicialização proposta melhora a capacidade de generalização do SNFDI. Assim, estes resultados se mostraram motivadores para continuar as pesquisas nesta linha; porém, não se pode deixar de mencionar que é importante realizar mais comprovações empíricas com outros conjuntos de dados, para que o SNFDI, junto a metodologia proposta, seja efetivamente utilizado como uma ferramenta de aprendizado de máquina.

No seguinte capítulo serão apresentados as conclusões deste trabalho.

Capítulo 6

Conclusões

6.1 Considerações Finais

Os esforços nas pesquisas na área de SIHs, em que por meio da integração de diferentes técnicas se potencializam as vantagens de cada técnica e se mitigam suas limitações, têm conduzido ao desenvolvimento de sistemas mais robustos, capazes de dar solução a problemas reais.

O SNFDI é um sistema híbrido que possui uma arquitetura de 3 camadas, cujos neurônios da camada intermediária realizam a operação de conjunção de conjuntos *fuzzy*, e os da camada de saída realizam disjunção. As entradas são graus de pertinência a conjuntos *fuzzy* obtidos mediante um processo de fuzzificação de dados. Seu algoritmo de treinamento é uma versão modificada do algoritmo de retropopagação que inclui as t-normas definidas por Zanusso [Zan97]. Antes da aplicação da metodologia proposta, a determinação de seus parâmetros de inicialização e de seus pesos era feita aleatoriamente, e a determinação do número adequado de neurônios da camada intermediária, mediante um processo de tentativa e erro, fazendo-se necessária, conseqüentemente, uma metodologia de inicialização para este sistema.

Baseando-se no fato de que, em RNAs baseadas no conhecimento é possível determinar, antecipadamente, aspectos da sua arquitetura relevantes para seu desempenho (e.g., número de camadas, número de neurônios e pesos das conexões), e sabendo que a teoria de *rough sets* possui a capacidade de gerar regras de dependência a partir de um determinado conjunto de dados; foi desenvolvida uma metodologia que automatiza a determinação do número de neurônios da camada intermediária e o valor dos pesos das conexões entre os neurônios do SNFDI. Além da teoria de *rough sets*, foram utilizados AGs para determinar os grupos de regras de dependência que conduzem a uma melhor generalização.

Foram utilizados os conjuntos de dados Diabetes, Coração e Vogal para realizar os experimentos e avaliar a metodologia de inicialização proposta. A escolha dos conjuntos de dados Diabetes e Coração foi motivada, principalmente, por duas razões. A primeira, foi a existência de resultados de classificação já conhecidos, em particular, os mostrados por Michie [MST94]. A segunda, foram as baixas taxas de acerto que ambos os conjuntos

possuem em comparação com outros conjuntos disponíveis. Por outro lado, no caso do conjunto Vogal, sua escolha procurou viabilizar a comparação com os resultados obtidos por Oliveira [Oli06] e Mitra et al. [MP94].

Os resultados usando esses três conjuntos de dados mostraram-se promissores, pois além de se automatizar, em parte, o processo de inicialização, mostrou-se que a capacidade de generalização do SNFDI melhora, em diferentes graus, aplicando a metodologia proposta. Ressalta-se o fato de que a automatização é parcial, pois esta ainda depende dos parâmetros de sobreposição e ponto de corte, bem como, dos parâmetros próprios de AGs, como o tamanho da população e o número de evoluções executadas.

Diferentemente à proposta de Banerjee et al. [BMP98], a metodologia de inicialização apresentada neste trabalho introduzi a integração de regras que permite considerar a mono e multirepresentatividade simultaneamente. Também utiliza um algoritmo genético como um mecanismo para encontrar a melhor combinação de regras, cujo mapeamento a um SNFDI determina a arquitetura e pesos que produzem uma boa generalização. Além disso possibilita que arquiteturas com diferentes números de neurônios na camada intermediária sejam testadas ao longo do processo iterativo.

Nos experimentos realizados, percebeu-se que os parâmetros, sobreposição (s) e ponto de corte (pc), usados nos processos de fuzzificação e de binarização, respectivamente, influem consideravelmente nos resultados da metodologia. Assim, por exemplo, no conjunto de dados Vogal, estabelecendo $s = 0.8$ e $pc = 0.7$, é atingida uma taxa média de acerto de 0.8090 para um número variável de neurônios; enquanto que, estabelecendo $s = 0.8$ e $pc = 0.9$ foi atingida uma taxa média de acerto de 0.7168. Contrariamente ao percebido para s e pc, as mudanças feitas em ϵ e ρ não causaram variações tão marcantes nas porcentagens de acerto.

Um aspecto que vale mencionar é que o número de indivíduos na população inicial e o número de iterações executadas no AG (em geral, para os três conjuntos de dados), foi relativamente pequeno, e ainda assim, conseguiu-se resultados desejáveis. Este fato não só reforça a idéia que a inicialização dos parâmetros livres de uma RNA são determinantes para seu funcionamento, mas também, explica a significação de s e pc na metodologia proposta. Estes dois parâmetros determinam diretamente o SD a partir do qual são geradas a regras de dependência. Assim, se a escolha de s e pc for a adequada, as regras geradas e, conseqüentemente, os indivíduos criados com base nas regras, constituirão um bom começo para o AG, precisando-se de um número menor de evoluções para encontrar a arquitetura e os pesos que produzam uma boa generalização.

Para o conjunto de dados Diabetes, atingiu-se uma taxa média de acerto de 0.8034 com 5 neurônios. Esse resultado foi melhor que os publicados por Michie et al. [MST94], nos que a máxima taxa média de acerto foi de 0.7770, atingida pelo LogDisc (Discriminador Logístico). Também, mostrou-se que a capacidade de generalização aumentou em aproximadamente 3.5% com relação à taxa de acerto obtida pelo SNFDI base, cuja melhor taxa foi de 0.7616 com 5 neurônios. Esta taxa situa o SNFDI base na quarta posição do *ranking* dos 22 classificadores.

Para o conjunto de dados Coração, a melhor taxa média de acerto (0.8423) foi atingida com 10 neurônios. O aumento na taxa média foi de, aproximadamente, 2%, em relação

ao SNFDI base. O menor custo médio do SNFDI evoluído foi 0.4731 superando ao PCM que teve um custo de 0.5740. Lembre-se que o PCM incorreu no menor custo entre os classificadores cujos mecanismos de aprendizado não incluíram custos de erro da classificação.

Para o conjunto de dados Vogal, foi atingida uma taxa média de acerto de 0.8162 com uma arquitetura de 21 neurônios. O aumento na taxa de acerto foi de, aproximadamente, 2% sobre a taxa de acerto obtida usando o SNFDI base. Esta taxa de acerto foi maior que o do modelo *Fuzzy*-O, que obteve uma $ta = 0.8010$, e também foi maior que o resultado publicado por Oliveira [Oli06] quem com 17 neurônios atingiu uma $ta = 0.8000$. O *Logical*-P2 atingiu uma $ta = 0.8610$, superando ao SNFDI evoluído. No entanto, esse modelo inclui a fuzzificação do atributo de classe, que o SNFDI ainda não explora. Vale a pena frisar que a fuzzificação do atributo de classe é um dos aperfeiçoamentos planejados para o SNFDI.

É importante mencionar que as arquiteturas encontradas pela metodologia de inicialização possuem um número maior de neurônios que a encontrada por Oliveira[Oli06]; conseqüentemente, poderia se afirmar que o custo computacional das primeiras é maior que o da arquitetura de 17 neurônios encontrada por Oliveira[Oli06], pois esta última possui um menor número de parâmetros livres a ser determinados.

Este fato traz a discussão sobre o *trade-off* entre o custo computacional e capacidade de generalização. Em alguns contextos, soluções com baixo custo são suficientes, e conseqüentemente, aceitáveis. Contrariamente, em casos em que vidas humanas estão em jogo, a capacidade de generalização e o custo do erro da classificação têm maior influência no momento de escolher um classificador. Contudo, nesta pesquisa, deu-se maior importância à capacidade de generalização, pois considera-se que a capacidade de classificação do SNFDI ainda se encontra em fase de avaliação.

Os resultados apresentados não só mostram que a metodologia proposta auxilia à automação da inicialização do SNFDI, mas também, que melhora a capacidade de generalização do SNFDI para os conjuntos de dados testados. Isto último, constitui um aspecto destacável, pois, como é conhecido, obter um alto nível de generalização é um dos objetivos mais importantes em problemas de classificação.

6.2 Limitações

As arquiteturas encontradas pela metodologia de inicialização, por estar baseadas no conhecimento de um determinado domínio, possivelmente, encontrem-se próximas à arquitetura ideal e seus pesos, próximos àqueles que minimizam o EQM, precisando, conseqüentemente, de tempos menores de treinamento. Neste sentido, uma limitação que se teve na execução dos experimentos é a referida ao uso de computadores com diferentes características de *hardware* e de *software*. Este fato fez com que não fosse possível verificar se os tempos incorridos para o treinamento das arquiteturas com pesos e número de neurônios determinados pela metodologia proposta, foram menores que os incorridos para o treinamento de arquiteturas com pesos gerados aleatoriamente.

A metodologia de inicialização depende de certos parâmetros que devem ser calculados para garantir o seu bom funcionamento. Entre os parâmetros a serem determinados encontram-se: o ponto de corte, a sobreposição, o número de evoluções do AG e o tamanho da população. Assim sendo, faz-se necessária a intervenção do usuário para determinar tais parâmetros, podendo-se dizer, que esta metodologia, automatiza em parte, o processo de inicialização do SNFDI.

6.3 Contribuições

Este texto expôs os conceitos gerais dos sistemas híbridos e da teoria de *rough set*, e apresentou, desde uma perspectiva prática, a aplicação de tais conceitos. Assim sendo, este trabalho contribui como uma referência bibliográfica de aplicação da teoria de *rough sets*, e em geral, de sistemas híbridos inteligentes. Assim mesmo, contribui como base para pesquisas subseqüentes, que abordarão outros aspectos da metodologia proposta e do sistema neuro-*fuzzy* diferenciável e interativo, na consecução do sistema híbrido em desenvolvimento.

A introdução de uma metodologia de inicialização para os pesos das conexões e o número de neurônios da camada intermediária do SNFDI constitui uma contribuição para o SNFDI, pois melhorou seu desempenho no sentido de aumentar a capacidade de generalização, e automatizou, em parte, seu processo de inicialização.

6.4 Trabalhos Futuros

O desenvolvimento da metodologia de inicialização, e o entendimento do SNFDI, abriram a possibilidade de novas linhas de pesquisa, entre as quais encontram-se:

A respeito da Metodologia

- Introdução de alguma técnica que guie a escolha do valor do ponto de corte utilizado na binarização.
- Implementação de outras heurísticas para determinar redutos como a apresentada por Wang et al. [WYT+07].
- Introdução de outros métodos de discretização para a criação do sistema de decisão. Por exemplo, ao invés de usar um ponto corte, como feito na binarização, usar dois ou três pontos de corte.
- Comparação, através de experimentos com dados reais, da metodologia proposta com outras metodologias de inicialização de RNAs (e.g., poda, AGs exclusivamente, etc.).

- Em vez de gerar regras que distinguam qualquer um representante de uma classe dos outros representantes da mesma, como feito neste trabalho; gerar regras que distinguam um determinado representante do resto dos representantes de uma mesma classe. Isso pode ser realizado usando o conceito de reduto relativo a um objeto.

A respeito do SNFDI

- Introdução de alguma técnica, que leve em conta o custo de erro da classificação durante o treinamento do SNFDI.
- Aperfeiçoamento do módulo de explanação (geração de regras de produção *fuzzy*), para possibilitar que este módulo funcione como um sistema classificador. Nesse caso o SNFDI proporcionaria o conjunto de regras para a construção da base de conhecimento.
- Determinação de funções de pertinência para atributos qualitativos para assim testar o SNFDI com conjuntos de dados que possuam o tipo de atributos mencionado.
- Utilização de funções de pertinência diferentes às propostas por Pal e Mitra[MP94] e utilizadas neste trabalho. Por exemplo, poder-se-ia utilizar funções trapezoidais e triangulares que são bastante utilizadas na literatura e outras como a apresentada por Donato et al. [DB95].
- Fuzzificação dos atributos de decisão, usando funções diferentes das propostas por Pal e Mitra [MP94].
- Criação de uma interface gráfica de fácil uso, para o SNFDI.
- Criação de um componente SNFDI que seja acoplável a um sistema de banco de dados qualquer.

Embora os resultados conseguidos com a metodologia de inicialização sejam significativos, merece ser destacada a necessidade de mais comprovações empíricas com outros conjuntos de dados, e da realização de posteriores pesquisas para que o SNFDI e a metodologia de inicialização proposta, constituam um sistema inteligente híbrido, efetivamente utilizado como uma ferramenta de aprendizado de máquina.

Apêndice A

Computação Granular

A.1 Introdução

A formação de grânulos de informação é um processo humano realizado de forma natural no intento de abstrair a realidade. Por exemplo, no contexto da visão, os humanos de forma natural, interpretam uma imagem buscando algum significado semântico. A imagem interpretada desta forma não é analisada pixel a pixel, mas é analisada por grupos com algum significado. Estes grupos, geralmente, envolvem regiões conformadas por vários pixels que possuem características comuns como, similaridade de cor, similaridade de textura, etc. [Ped05] [Zad97].

Assim, os grânulos de informação, como seu nome mesmo indica, são uma coleção de entidades agrupadas seguindo um critério de similaridade, indistingüibilidade, proximidade ou funcionalidade de suas entidades. Os grânulos de informação, como toda abstração da realidade, visam construir representações corretas do mundo e suportar e facilitar a percepção humana da realidade física e virtual [BP03].

Neste Apêndice serão expostas as noções básicas de grânulos de informação. Também expor-se-á como a aplicação destas noções na solução de problemas deu origem a um novo termo denominado *computação granular*.

A.2 Níveis de Granularidade

Os grânulos de informação estão organizados em vários níveis seguindo uma estrutura hierárquica. Por exemplo, visando tratar o tempo, o ser humano tem visto a necessidade de organizá-lo em anos, meses, dias, horas, minutos, etc.; i.e., tem dividido o tempo em vários níveis de granularidade [Ped05].

Assim sendo, dependendo do problema a ser tratado, a eleição de um nível de granularidade (tamanho dos grânulos de informação), pode variar drasticamente. Por exemplo, o gerente de vendas de uma empresa, provavelmente, esteja interessado em comparar as

ganâncias trimestrais de algum de seus produtos. Assim, dividindo o tempo em períodos de três meses solucionar-se-á seu problema. No entanto, para um engenheiro eletrônico que deseja testar a rapidez de um sistema digital de alta velocidade, os grânulos de interesse não vão ser maiores do nível de nanosegundos [Ped05].

A.3 Computação Granular: Surgimento

Ao longo de vida da inteligência artificial, diversas paradigmas que operam sobre grânulos de informação para a solução de problemas surgiram de maneira independente; e.g., o análise de intervalos, a teoria de conjuntos *fuzzy*, a teoria de *rough sets*, etc. No entanto, tais paradigmas não foram vistas em conjunto; i.e., não existia um modelo unificado nem uma base comum para todas elas. Esse fato não só tem provocado que o potencial da aplicação da computação baseada grânulos não seja percebido em toda sua magnitude; mas também, tem dificultado adquirir um entendimento mais claro da interação entre os paradigmas e uma visualização macroscópica da maneira que estes podem se comunicar [Ped05] [Yao05]. A necessidade de um marco de trabalho onde as teorias, metodologias e técnicas que utilizam grânulos possam ser formalmente desenvolvidas, motivou o surgimento de um novo paradigma chamado *computação granular*. Este foi, inicialmente, introduzido por Lin em 1997 [Lin97] [Ped05] [PM04] [Yao05].

Existem várias razões para o estudo da computação granular. Yao [Yao05] cita as seguintes:

1. O mundo real está organizado em níveis; razão pela qual a computação granular pode fornecer uma representação mais adequada desse mundo;

2. A computação granular é coerente com a forma humana de pensar e solucionar problemas, no sentido de decompor o tudo nas suas partes;

3. A computação granular permite a simplificação de problemas complexos. Omitindo detalhes irrelevantes e desnecessários e focando-se no nível de abstração correto (o mais adequado para uma situação) pode-se simplificar um problema e solucioná-lo de maneira mas simples;

4. A computação granular fornece soluções econômicas e de baixo custo. Considerando um mesmo problema em diversos níveis de granularidade, ignoram-se alguns detalhes. Isso pode conduzir a encontrar soluções apenas aproximadas, mas suficientes em alguns contextos. O benefício de tais soluções deve-se a que use uma menor quantidade de recursos; e portanto implicam um menor custo.

A.4 Grânulos *Crisp* e Grânulos *Fuzzy*

Por outra parte, dependendo da natureza da fronteira entre os grânulos de um determinado problema; esses podem ser divididos em grânulos *crisp* ou em grânulos *fuzzy*. Nos primeiros, a fronteira entre os grânulos é bem definida. Os grânulos nesses casos são

considerados partições do universo (conjuntos disjuntos do universo). Contrariamente, os grânulos *fuzzy* possuem uma região de fronteira *fuzzy*.

Por exemplo, dado o problema de contratar uma pessoa para o cargo de uma empresa. Considerando que um dos critérios de avaliação é o *diploma* que o candidato possui e que um *diploma* pode ser de três tipos, a saber, "MBA", "MCE" ou "Msc"; pode ser visto que as fronteiras que distinguem os diferentes tipos deste atributo são *crisp*, pois um *diploma* pode ser apenas "MBA" ou "MCE" ou "Msc"; mas não de dois ou três tipos ao mesmo tempo. Por outra parte, dado o problema de calcular a *temperatura* adequada para uma máquina lavadora e assumindo que o atributo *temperatura* pode ser de três tipos, "baixa", "média" e "alta"; o valor de uma *temperatura* considerada como "alta", também pode ser entendido como "baixa" e "média" num menor grau; i.e., um valor de temperatura pode ser de vários tipos simultaneamente. Este fato leva a considerar os grânulos da variável *temperatura* como grânulos *fuzzy*. A Figura A.1, à esquerda, mostra os grânulos *crisp* do atributo *diploma* e à direita, os grânulos fuzzy do atributo *temperatura*.

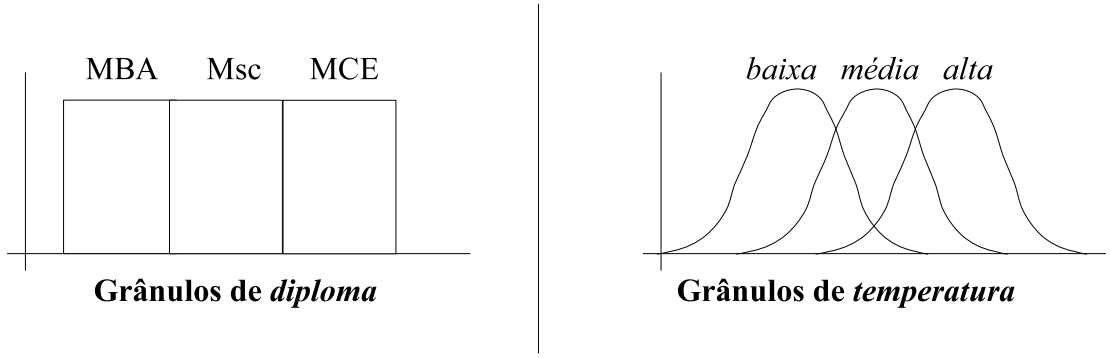


Figura A.1: Grânulos *crisp* do atributo *diploma* e grânulos *fuzzy* do atributo *temperatura*.

Os grânulos *fuzzy* são utilizados, e.g., na teoria *fuzzy*. Já no casso dos grânulos *crisp*, são usados na análise de intervalos, na discretização, na teoria de Dempster-Shafer, na teoria de *rough sets*, entre outros [BP03] [Zad97].

A.5 Importância da Computação Granular

No sentido mais filosófico, a computação granular pode descrever uma forma de pensar que tem fundamento na habilidade humana de perceber a realidade em vários níveis de granularidade, para assim, abstrair e considerar apenas aqueles aspectos que servem para num problema específico. Dando ênfase a diferentes níveis de granularidade, pode-se obter diferentes níveis de conhecimento e um maior entendimento do mesmo. Portanto, o raciocínio usando grânulos de informação é essencial para a maneira na qual os seres humanos resolvem problemas e tem um impacto significativo no projeto e implementação de sistemas inteligentes [BP03].

Apêndice B

Lógica *Fuzzy*

B.1 Introdução

Este apêndice contém uma breve descrição ao respeito dos conceitos da teoria *fuzzy* que se relacionam com o sistema neuro-*fuzzy* descrito na Seção 2.5.

B.2 Definições Básicas

A teoria de conjuntos *fuzzy* pode ser vista como uma extensão da teoria de conjuntos clássica. Como é conhecido, a pertinência de um elemento a um conjunto na teoria de conjuntos clássica é avaliada como não ou sim, falso ou verdadeiro, ou 0 ou 1. No caso da teoria *fuzzy*, a pertinência não é estritamente absoluta, pois se dá a ela a possibilidade de tomar uma série de valores num determinado intervalo.

Um *conjunto fuzzy* A, e um conjunto de pares ordenados, tal que,

$$A = \{(x, \mu_A(x)) : x \in X\}, \tag{B.1}$$

onde, X é o *universo do discurso*, i.e., o conjunto de todas as designações numéricas possíveis para A; x é um elemento pertencente ao universo de discurso X; e $\mu_A(x)$ é uma função chamada *função de pertinência*, $\mu_A(x) : A \rightarrow [0, 1]$. Esta função associa a cada $x \in X$ um número real no intervalo $[0, 1]$[1] que indica quanto o elemento x pertence ao conjunto *fuzzy* A. $\mu_A(x) = 1$ significa pertinência absoluta ao conjunto A; enquanto que, $\mu_A(x) = 0$ significa não pertinência ao conjunto *fuzzy* A. Valores intermédios indicam graus pertinência parciais.

[1] Na literatura, geralmente, tem-se usado o intervalo $[0, 1]$ para definir o grau de pertinência a um conjunto *fuzzy*. No entanto, outros intervalos são usados; por exemplo $[-1, 1]$, como é caso do SNFDI.

B.3 Operações de Conjuntos *Fuzzy*

Inicialmente, Zadeh [Zad65] definiu três operações *fuzzy*. Cada uma delas correspondia às operações básicas da teoria de conjuntos clássica, a saber, união, intersecção e complemento. Assim, Zadeh definiu estas operações *fuzzy* como a seguir:

A *união* de dois conjuntos *fuzzy* A e B é o conjunto *fuzzy* $C = A \cup B$ ou $C = A$ **ou** B; cuja função de pertinência é definida como,

$$\mu_C(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x). \tag{B.2}$$

A *intersecção* de dois conjuntos *fuzzy* A e B é o conjunto *fuzzy* $C = A \cap B$ ou $C = A$ **e** B cuja função é definida como,

$$\mu_C(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x). \tag{B.3}$$

Por último, o *complemento* de conjunto fuzzy A, denotado por $\overline{A}$ ou **não** A, é definido como,

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \tag{B.4}$$

As definições, dadas por Zadeh [Zad65], para os operadores *fuzzy*, intersecção e união são um caso especial de uma forma mais geral de definir ambas as operações. Esta forma mais geral está determinada por um conjunto de funções que cumprem certos requisitos. No caso da intersecção *fuzzy*, as funções que os cumprem são denominadas *t-normas*; enquanto que, no caso da união *fuzzy*, tais funções são denominadas *t-conormas* ou *s-normas*.

Uma função $T : [0,1] \times [0,1] \to [0,1]$ é uma t-norma se cumpre as seguintes propriedades,

- Condições limite[2]: $T(0,0) = 0$, $T(a,1) = T(1,a) = a$.
- Monotonicidade: $T(a,b) \leq T(c,d)$ quando $a \leq c$ e $b \leq d$.
- Comutatividade: $T(a,b) = T(b,a)$.
- Associatividade: $T(T(a,b),c) = T(a,T(b,c))$.

Uma função $S : [0,1] \times [0,1] \to [0,1]$ é uma t-conorma se,

- Condições limite[3]: $S(1,1) = 1$, $S(a,-1) = S(-1,a) = a$.
- Monotonicidade: $S(a,b) \leq S(c,d)$ quando $a \leq c$ e $b \leq d$.

[2]Para funções definidas no intervalo $[-1,1]$, como é o caso das t-normas de Zanusso [Zan97], a condição limite é dada por $T(-1,-1) = -1, T(a,1) = T(1,a) = a$.

[3]Para funções definidas no intervalo $[-1,1]$, como é o caso das t-conormas de Zanusso [Zan97], a condição limite é dada por $S(1,1) = 1, S(a,-1) = S(-1,a) = a$.

- Comutatividade: $S(a,b) = S(b,a)$.
- Associatividade: $S(S(a,b),c) = S(a,S(b,c))$.

Em alguns casos exige-se a continuidade e a subidempotência da t-normas.

É importante ressaltar que as t-normas e as t-conormas se reduzem aos operadores clássicos de união e intersecção quando os conjuntos em questão são conjuntos *crisp*; i.e., conjuntos clássicos.

A Tabela B.1 mostra as t-normas e t-conormas mais utilizadas. Uma tabela mais completa de t-normas é apresentada por Klir em [KY95].

Nome	t-norma	t-conorma
Zadeh	$\min(a,b)$	$\max(a,b)$
Probabilística	$a.b$	$a+b-ab$
Lukasiewicz	$\max(a+b-1,0)$	$\min(a+b,1)$

Tabela B.1: Principais t-normas e t-conormas.

B.4 Significados das Propriedades de T-norma e T-conorma

Zanusso [Zan97] fez uma série de considerações sobre o significado de cada axioma, para que a t-norma seja considerada intuitivamente aceitável como intersecções *fuzzy* significativas para quaisquer pares de conjuntos *fuzzy*.

- A primeira condição para t-norma e t-conorma estabelece que as operações de intersecção e união para conjuntos *fuzzy* sejam extensões das mesmas operações clássicas sobre os conjuntos ordinários, pois estas operações se reduzem a operações não-*fuzzy* quando os graus de pertinência estão restritos ao conjunto $\{0,1\}$. Também por isto são denominados operadores lógicos. Quando o argumento de T é 1, expressando pertinência total, as condições-limite e a comutatividade também asseguram, como a concepção intuitiva requer, que o grau de pertinência na intersecção seja igual ao do outro argumento.
- A monotonicidade e a comutatividade expressam a exigência natural de que um decrescimento no grau de pertinência no conjunto A ou B não possa produzir um aumento no grau de pertinência da intersecção.
- A comutatividade assegura que a intersecção *fuzzy* é simétrica, i.e., indiferente quanto à ordem em que os conjuntos a serem combinados são considerados.

- A associatividade assegura que se pode tomar a intersecção para qualquer número de conjuntos em qualquer ordem de agrupamento pareada desejada. Esta operação permite estender a intersecção *fuzzy* para além de dois conjuntos. O grau de pertinência independe do agrupamento dos conceitos.

- A continuidade impede que ocorra uma situação em que uma mudança muito pequena no grau de pertinência do conjunto A ou do conjunto B produza uma grande (descontínua) mudança no grau de pertinência de $A \cap B$.

- A subidempotência expressa que o grau de pertinência ao conjunto *fuzzy* $A \cap B$ neste caso especial não pode exceder a. Se se exigisse a idempotência, isto significaria que a conjunção de conjuntos iguais resultaria no mesmo conjunto.

Apêndice C

Conjuntos de Dados

Conjunto de Dados	No. Exemplos	No. Classes	No. Exemplos por Classe
Vogal	871	6	Classe 1 (Vogal ∂): 72 Classe 2 (Vogal a): 89 Classe 3 (Vogal i): 172 Classe 4 (Vogal u): 151 Classe 5 (Vogal e): 207 Classe 6 (Vogal o): 180
Diabetes	768	2	Classe 1 (Diagnóstico -): 500 Classe 2 (Diagnóstico +): 268
Coração	270	2	Classe 1 (Diagnóstico -): 150 Classe 2 (Diagnóstico +): 120

Tabela C.1: Informações dos conjuntos de dados.

Referências Bibliográficas

[ABO00] F. M. Azevedo, L. M. Brasil, e R. C. L. Oliveira. *Redes Neurais com Aplicações em Controle e em Sistemas Especialistas*. Visual Books Editora, 2000.

[BBM93] D. Beasley, D. R. Bull, e R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[BdCL00] A. P. Braga, A. P. L. F. de Carvalho, e T. B. Ludermir. *Redes Neurais Artificiais: Teoria e Aplicações*. LTC Press, 2000.

[BL06] F. L. Biajoli e L. A. N. Lorena. Mirrored traveling tournament problem: an evolutionary approach. In *International Joint Conference 2006, 10th Ibero-American Conference on AI, 18th Brazilian Symposium on AI, Ribeirão Preto - SP, Brazil, October 23-27, 2006, Proceedings*. 2006.

[BMP98] M. Banerjee, S. Mitra, e S. K. Pal. Rough fuzzy MLP: Knowledge encoding and classification. *IEEE Transactions on Neural Networks*, 9(6):1203–1216, 1998.

[Bos06] I. Bose. Deciding the financial health of dot-coms using rough sets. *Information & Management*, 43(7):835–846, 2006.

[BP03] A. Bargiela e W. Pedrycz. *Granular Computing: An Introduction*. Springer, 2003.

[CPC06] J. E. Cabral, J. O. P. Pinto, e K. Collazos. Methodology for fraud detection using rough sets. In *IEEE International Conference on Granular Computing 2006*, páginas 244–249. 2006.

[CPK05] E. Cantu-Paz e C. Kamath. An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 35(5):915–927, 2005.

[Dav91] L. Davis. *Handbook of genetic algorithms*. Van Nostrand Reinhold, 1991.

[DB95] J. M. Donato e E. Barbieri. Mathematical representation of fuzzy membership functions. In *SSST '95: Proceedings of the 27th Southeastern Symposium on System Theory (SSST'95)*, página 290. IEEE Computer Society, 1995.

[DCRS98] J. S. Deogun, S. K. Choubey, V. V. Raghavan, e H. Sever. Feature selection and effective classifiers. *Journal of the American Society for Information Science*, 49(5):423 –434, 1998.

[Dia98] J. S. Dias. *Sensibilidade Paramêtrica como Guia para o Treinamento Híbrido de Redes Neurais*. Tese de doutorado, Universidade Federal de Santa Catarina, Dezembro 1998.

[EA05a] A. Evsukoff e P. E. M. Almeida. Sistemas fuzzy. In *Sistemas Inteligentes: Fundamentos e Aplicações*, páginas 169–201. Manole Ltda, 2005.

[EA05b] A. Evsukoff e P. E. M. Almeida. Sistemas neuro-fuzzy. In *Sistemas Inteligentes: Fundamentos e Aplicações*, páginas 203–224. Manole Ltda, 2005.

[eJES03] A. E. Eiben e J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[FMNU95] T. Furuhashi, Y. Miyata, K. N., e Y. Uchikawa. A new approach to genetic based machine learning and an efficient finding of fuzzy rules. In *Selected papers from the IEEE/Nagoya-University World Wisepersons Workshop on Advances in Fuzzy Logic, Neural Networks and Genetic Algorithms*, páginas 173–189. 1995.

[FS94] H. C. Fu e J. J. Shann. A fuzzy neural network for knowledge learning. *International Journal of Neural Systems*, 5(1):13–22, 1994.

[Fu95a] L. Fu. Introduction to knowledge-based neural networks. *Knowledge-Based Systems*, 8(6):299–300, 1995.

[Fu95b] L. Fu. Special issue: Knowledge-based neural networks. *Knowledge-Based Systems*, 8(6):298–298, 1995.

[Gal88] S.I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2):152–169, 1988.

[GB02] J. W. Grzymala-Busse. Data reduction: discretization of numerical attributes. In *Handbook of data mining and knowledge discovery*, páginas 218–225. Oxford University Press, 2002.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[GR94] M. Gupta e D. H. Rao. On the principles of fuzzy neural networks. *Fuzzy Sets and Systems*, 61:1–18, 1994.

[Has07] A. Hassaniena. Fuzzy rough sets hybrid scheme for breast cancer detection. *Image and Vision Computing*, 25(2):172–183, 2007.

[Hay98] S. Haykin. *Neural networks: A comprehensive foundation*. Prentice Hall, 1998.

[HLS03] K. Hu, Y. Lu, e C. Shi. Feature ranking in rough sets. *AI Communications*, 16(1):41–50, 2003.

[HO06] M. R. Heinen e F. S. Osório. Gait control generation for physically based simulated robots using genetic algorithms. In *International Joint Conference 2006, 10th Ibero-American Conference on AI, 18th Brazilian Symposium on AI, Ribeirão Preto - SP, Brazil, October 23-27, 2006, Proceedings*. 2006.

[Hof01] F. Hoffmann. Evolutionary algorithms for fuzzy control system design. In *Proceedings of the IEEE*, páginas 1318–1333. 2001.

[Hol75] J. H. Holland. *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, 1975.

[HSG89] S. A. Harp, T. Samad, e A. Guha. Towards the genetic synthesis of neural networks. In *Proceedings of the Third International Conference on Genetic Algorithms*, páginas 643–649. 1989.

[Hu95] X. Hu. *Knowledge Discovery in Databases: Attribute-Oriented Rough Set Approach*. Tese de doutorado, University of Regina, Canada, Junho 1995.

[HWF06] A. Hedar, J. Wang, e M. Fukushima. Tabu search for attribute reduction in rough set theory. Relatório Técnico 2006-008, Department of Applied Mathematics and Physics, Kyoto University, 2006.

[HWW07] T.-P. Hong, T.-T. Wang, e S.-L. Wang. Mining fuzzy ß-certain and ß-possible rules from quantitative data based on the variable precision rough-set model. *Expert Systems with Applications*, 32(1):223–232, 2007.

[INK00] H. Ishibuchi, T. Nakashima, e T. Kuroda. A hybrid fuzzy gbml algorithm for designing compact fuzzy rule-based classification systems. In *The Ninth IEEE International Conference on Fuzzy Systems*, volume 2, páginas 706–711. 2000.

[INYT95] H. Ishibuchi, K. Nozaki, N. Yamamoto, e H. Tanaka. Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(3):260–270, 1995.

[IY04] H. Ishibuchi e T. Yamamoto. Multi-objective evolutionary design of fuzzy rule-based systems. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 3, páginas 2362 – 2367. 2004.

[Jan91] J. S. R. Jang. Fuzzy modeling using generalized neural networks and kalman filter algorithm. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, páginas 762–767. 1991.

[Jan93] J. S. R. Jang. Anfis: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man and Cybernetics*, 23(3):665–685, 1993.

[Kas96] N. Kasabov. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*. MIT Press, 1996.

[Kit90a] H. Kitano. Designing a neural network using genetic algorithm with graph generation system. *Complex Systems*, 4(4):461–476, 1990.

[Kit90b] H. Kitano. Empirical studies on the speed of convergence of neural network training using genetic algorithms. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, volume 2, páginas 789–795. 1990.

[KPPS99] J. Komorowski, Z. Pawlak, L. Polkowski, e A. Skowron. Rough sets: A tutorial. In *Rough-Fuzzy Hybridization: A New Method for Decision Making*, páginas 3–98. Springer-Verlag Telos, 1999.

[KY95] G. J. Klir e B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall, 1995.

[LCB05] T. B. Ludermir, A. C. P. L. F. Carvalho, e A. P. Braga. Computação evolutiva. In *Sistemas Inteligentes: Fundamentos e Aplicações*. Manole Ltda, 2005.

[LCBS05] T. B. Ludermir, A. C. P. L. F. Carvalho, A. P. Braga, e M. C. P. Souto. Sistemas inteligentes híbridos. In *Sistemas Inteligentes: Fundamentos e Aplicações*, páginas 225–248. Manole Ltda, 2005.

[LF99] E. G. M. Lacerda e A. C. P. L. F.Carvalho. Introdução aos algoritmos genéticos. *Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação*, II:51–126, 1999.

[Lin96] P. Lingras. Rough sets and artificial neural networks. In *Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'96)*, páginas 1445–1450. 1996.

[Lin97] T. Y. Lin. Granular computing. *Announcement of the BISC Special Interest Group on Granular Computing*, 1997.

[LRG96] M. H. Lim, S. Rahardja, e B. H. Gwee. A GA paradigm for learning fuzzy rules. *Fuzzy Sets and Systems*, 82(2):177–186, 1996.

[MA03] S. Mitra e T. Acharya. *Data Mining: Multimedia, Soft Computing, and Bioinformatics*. John Wiley and Sons, 2003.

[Mam74] E. H. Mamdani. Application of fuzzy algorithms for control of simple dynamic plants. In *Proceedings of IEEE Control and Science*, páginas 1585–1588. 1974.

[Man93] M. Mandischer. Representation and evolution of neural networks. In *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, páginas 643–649. 1993.

[Man94] V. Maniezzo. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5(1):39–53, 1994.

[MC05] P. Melin e O. Castillo. *Hybrid Intelligent Systems for Pattern Recognition Using Soft Computing: An Evolutionary Approach for Neural Networks and Fuzzy Systems*. Springer, 2005.

[MJ94] G. Michal e S. Jacek. RSL- the rough set library version 2.0, ics research report, 1994.

[MLKN07] G. Meza-Lovón, M. L. Kanashiro, e P. M. B. Neto. Manual de referência: SNFDI. Relatório técnico, Department Computação e Estatística, Universidade Federal de Mato Grosso do Sul, Brasil, 2007.

[MP92] S. Mitra e S.K. Pal. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks*, 3(5):683–697, 1992.

[MP94] S. Mitra e S. K. Pal. Logical operation based fuzzy MLP for classification and rule generation. *Neural Networks*, 27(2):353–373, 1994.

[MST94] D. Michie, D. J. Spiegelhalter, e C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Prentice Hall, 1994.

[NK95] D. Nauck e R. Kruse. NEFCLASS a neuro-fuzzy approach for the classification of data. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, páginas 461–465. 1995.

[NK97a] D. Nauck e R. Kruse. Function approximation by NEFPROX. In *Proc. Second European Workshop on Fuzzy Decision Analysis and Neural Networks for Management, Planning and Optimization*, páginas 160–169. 1997.

[NK97b] D. Nauck e R. Kruse. Neuro-fuzzy systems for function approximation. In *Proc. 4th Int. Workshop Fuzzy-Neuro-System*, páginas 316–323. 1997.

[NKK97] D. Nauck, R. Kruse, e F. Klawonn. *Foundations of Neuro-Fuzzy Systems*. John Wiley and Sons, Inc., 1997.

[Oli06] F. R. Oliveira. *Rede Neural Difusa com T-normas Diferenciáveis e Interativas*. Dissertação de mestrado, Universidade Federal de Mato Grosso do Sul, Novembro 2006.

[Paw82] Z. Pawlak. Rough sets. *International Journal of Information and Computer Sciences*, 11(5):341–356, 1982.

[Paw91] Z. Pawlak. *Rough Sets - Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, 1991.

[Ped91] W. Pedrycz. A referencial scheme of fuzzy decision making and its neural network structure. *IEEE Transactions on Systems, Man and Cybernetics*, 21:1593–1604, 1991.

[Ped05] W. Pedrycz. *Knowledge-Based Clustering: From Data to Information Granules*. Wiley-Interscience, 2005.

[PJBSZ95] Z. Pawlak, J.G-B., R. Slowinski, e W. Ziarko. Rough sets. *Communications of the ACM*, 38:88–95, 1995.

[PM04] S. K. Pal e P. Mitra. *Pattern Recognition Algorithms for Data Mining*. Chapman Hall/CRC, 2004.

[PP97] J. C. F. Pujol e R. Poli. Evolution of the topology and the weights of neural networks using genetic programming with a dual representation. Relatório Técnico CSRP-97-07, The University of Birmingham, School of Computer Science, February 1997.

[PR93] W. Pedrycz e A. F. Rocha. Fuzzy-set based models of neurons knowledge-based networks. *IEEE Transactions on Fuzzy Systems*, 1(4):254–266, 1993.

[PS94] Z. Pawlak e R. Slowinski. Decision analysis using rough sets. *International Transactions in Operational Research*, 1(1):107–114, 1994.

[PS99] S. K. Pal e A. Skowron. *Rough Fuzzy Hybridization: A New Trend in Decision-Making*. Springer-Verlag Telos, 1999.

[RHW86] D. E. Rumelhart, G. E. Hinton, e R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[Rot06] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer, 2006.

[SK88] M. Sugeno e G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28(1):15–33, 1988.

[SP94] M. Srinivas e L.M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656–667, 1994.

[SR92] A. Skowron e C. Rauszer. The discernibility matrices and functions in information systems. In *Intelligent Decision Support: Handbook of Applications and Advances to Rough Sets Theory*, páginas 331–362. Kluwer Academic Publishers, 1992.

[SS96] R. Stonier e D. Sturgess. Genetic learning of the irrigation cycle for water flow in cropped soils. In *SEAL'96: Selected papers from the First Asia-Pacific Conference on Simulated Evolution and Learning*, páginas 89–96. 1996.

[Sus98] R. Susmaga. Experiments in incremental computation of reducts. In *Rough sets in knowledge discovery 1*, páginas 530–553. Physica-Verlag, 1998.

[TCC06] Y. C. Tsai, C. H. Cheng, e J.-R. Chang. Entropy-based fuzzy rough classification approach for extracting classification rules. *Expert Systems with Applications*, 31(2):436–443, 2006.

[TG96] Y. Tsujimura e M. Gen. Genetic algorithms for solving multiprocessor scheduling problems. In *SEAL'96: Selected papers from the First Asia-Pacific Conference on Simulated Evolution and Learning*, páginas 106–115. 1996.

[TS85] T. Takagi e M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15:116–132, 1985.

[TS94] G. G. Towell e J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119–165, 1994.

[UCP00] N. Urquhart, K. Chisholm, e B. Paechter. Optimising an evolutionary algorithm for scheduling. In *Lecture Notes In Computer Science*, volume 1803, páginas 307–318. 2000.

[WL03] C. J. Wu e G. Y. Liu. A genetic approach for simultaneous design of membership functions and fuzzy control rules. *Journal of Intelligent and Robotic Systems*, 28:195–211, 2000(3).

[WYT+07] X. Wang, J. Yang, X. Teng, W. Xia, e R. Jensen. Feature selection based on rough sets and particle swarm optimization. *Pattern Recognition Letters*, 28(4):459–471, 2007.

[Yao05] Y. Yao. Perspectives of granular computing. *IEEE International Conference on Granular Computing*, 1:85–90, 2005.

[Yas95] R. Yasdi. Combining rough sets learning- and neural learning-method to deal with uncertain and imprecise information. *Neurocomputing*, 7(1):61–84, 1995.

[YLL07] H.-H. Yanga, T.-C. Liub, e Y.-T. Lina. Applying rough sets to prevent customer complaints for ic packaging foundry. *Expert Systems with Applications*, 32(1):151–156, 2007.

[Zad65] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zad97] L. A. Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy Sets and Systems*, 90:111–127, 1997.

[Zan97] M. B. Zanusso. *Famílias de T-Normas Diferenciáveis, Funções de Pertinência Relacionadas e Aplicações*. Tese de doutorado, Universidade Federal de Santa Catarina, Dezembro 1997.

[ZKF02] L.-Y. Zhai, L.-P. Khoo, e S.-C. Fok. Feature extraction using rough set theory and genetic algorithms: an application for the simplification of product quality evaluation. *Computers and Industrial Engineering*, 43(4):661–676, 2002.