Pós-Graduação em Ciência da Computação

DIOGO DA SILVA SEVERO

"Proposta de uma Rede Neural Modular que Seleciona um Conjunto Diferente de Características por Módulo"

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2013

DIOGO DA SILVA SEVERO

“Proposta de uma Rede Neural Modular que Seleciona um Conjunto Diferente de Características por Módulo"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: George Darmiton da Cunha Cavalcanti
CO-ORIENTADOR: Tsang Ing Ren

RECIFE
2013

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S498p Severo, Diogo da Silva
Proposta de uma rede neural modular que seleciona um conjunto diferente de características por módulo / Diogo da Silva Severo. – 2013.
83 f.: il., fig., tab.

Orientador: George Darmiton da Cunha Cavalcanti.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2013.
Inclui referências.

1. Inteligência artificial. 2. Redes neurais artificiais. 3. Aprendizagem de máquina. 4. Reconhecimento de padrão. I. Cavalcanti, George Darmiton da Cunha (orientador). II. Título.

006.31 CDD (23. ed.) UFPE- MEI 2016-105

Dissertação de Mestrado apresentada por **Diogo da Silva Severo** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Proposta de uma Rede Neural Modular que Seleciona um Conjunto Diferente de Características por Módulo**", orientada pelo **Prof. George Darmiton da Cunha Cavalcanti** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Leandro Maciel Almeida
Centro de Informática / UFPE

_____

Profa. Renata Lúcia Mendonça Ernesto do Rego
Departamento de Sistemas de Informação / IFPE

_____

Prof. George Darmiton da Cunha Cavalcanti
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 15 de agosto de 2013

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Dedico este trabalho a minha família, em especial a minha mãe, a meus amigos e a todos os professores que contribuíram para a minha formação desde o ensino básico até o superior.

Agradecimentos

Primeiramente, agradeço a Deus por ter chegado a esse ponto da minha vida acadêmica e por tudo mais.

Agradeço à minha família, a meus avôs *in memorian*, a meus tios e tias. Faço um agradecimento especial à minha mãe Maria por sempre me pôr no caminho dos estudos desde a mais tenra idade. Agradeço a ela pelo amor e apoio incondicionais. Agradeço à minha irmã Juliana também pelo apoio e pelo encorajamento sempre.

Agradeço a George Darmiton, meu professor de muitas disciplinas na graduação e mestrado. Agradeço a ele por ter aceitado me orientar, pelas conversas sempre francas, pelos conselhos, pela confiança depositada em mim e pela compreensão generosa. Agradeço também ao meu co-orientador, professor Tsang.

Agradeço aos avaliadores Leandro Almeida e Renata Mendonça do Rêgo pelas suas respectivas avaliações que foram bem criteriosas e muito contribuíram para a melhora do documento final.

Agradeço aos meus amigos de todas as horas: Yane, Everson, Nivan e Ruan. A Everson, reforço os agradecimentos por revisar minha dissertação e artigos e pelas dicas sempre pertinentes. Agradeço aos meus amigos do começo do curso: Leyla, Thiago Lessa, Thiago Carvalho e Gleicy. Agradeço a Paulo Ricardo, Felipe Kühner, Eduardo Gade, Nelson Gutemberg e João Rufino, amizades que começaram no fim da graduação e se intensificaram no mestrado.

Por fim, agradeço a todos os professores do CIn com quem tive a oportunidade de ter aulas. Em especial aos professores George, Ruy, Teresa, Kátia, Renata, André Santos e Edna. Faço um agradecimento a todos os funcionários do Centro de Informática por garantirem uma estrutura e ambientes tão propícios aos estudos. Não é à toa que o centro sempre vem se destacando nacionalmente em avaliações de qualidade.

O futuro pertence àqueles que acreditam na beleza de seus sonhos.

—ELLEANOR ROOSEVELT

Resumo

Redes Neurais Artificiais foram inspiradas nas redes neurais biológicas e as principais semelhanças compartilhadas por ambas são: capacidade de processamento de informação de forma paralela e distribuída, presença de unidades de processamento simples e capacidade de aprendizado através de exemplos. Entretanto, as redes neurais artificiais não apresentam uma característica inerente às redes neurais biológicas: modularização. Em contraste com as redes neurais artificiais, nosso cérebro apresenta áreas especializadas distintas responsáveis por tarefas específicas como visão, audição e fala, por exemplo. Com o intuito de aproximar ainda mais as redes neurais artificiais das redes neurais biológicas, foram propostas as redes neurais modulares. Tais redes tiram proveito da modularização para superar as redes neurais simples quando lidam com problemas complexos. Um conceito crucial relacionado ao uso de redes neurais modulares é a decomposição. A decomposição trata da divisão do problema original em vários subproblemas, menores e mais simples de serem resolvidos. Cada subproblema é tratado por um especialista (rede neural simples) específico. Ao solucionar seus respectivos subproblemas, cada módulo faz uso de todo o conjunto original de características para treinar seus especialistas. Entretanto, é esperado que diferentes módulos requeiram diferentes características para realizar suas tarefas. Dessa forma, é importante escolher quais características melhor preservam a informação discriminatória entre classes necessária à tarefa de classificação de cada módulo. Este trabalho propõe uma arquitetura de rede neural modular que seleciona um conjunto específico de características por módulo, sendo este um tópico pouco explorado na literatura uma vez que, em sua maioria, os trabalhos envolvendo redes neurais modulares não realizam seleção de características para cada módulo específico. O procedimento de seleção de características é um método de otimização global baseado no PSO binário. Outra contribuição do presente trabalho é um método híbrido de seleção e ponderação de características baseado no PSO binário. Foram realizados experimentos com bases de dados públicas e os resultados mostraram que a arquitetura proposta obteve melhores taxas de classificação ou taxas iguais, porém, fazendo uso de menos características quando comparadas a redes neurais modulares que não realizam a seleção de características por módulo. Os experimentos realizados com o método híbrido de seleção e ponderação de características baseado em otimização por enxame de partículas mostraram taxas de classificação superiores às taxas obtidas pelos métodos que serviram de comparação.

Palavras-chave: Decomposição de Tarefas. Otimização Global. *Particle Swarm Optimization*. Redes Neurais Modulares. Seleção de Características

Abstract

Artificial Neural Networks were inspired by biological neural networks and the major similarities shared by both are: the ability to process information in a parallel and distributed way, the presence of simple processing units and the ability for learning through examples. However, artificial neural networks do not present an inherent characteristic of biological neural networks: modularization. In contrast to artifical neural networks, our brain has distinct specialized areas for specific tasks such as vision, hearing and speech, for example. With the aim of bringing even more artificial neural networks to biological neural networks, modular neural networks were proposed. Such networks take advantage of modularization to outperform the simple neural networks when dealing with complex problems. A crucial concept related to the use of neural networks is the task decomposition. The task decomposition divides the original problem into several subproblems, smaller and simpler to resolve. Each subproblem is handled by a specific expert (simple neural network). To solve their subproblems, each module makes use of the whole set of features to train its expert. Nevertheless, it is expected that different modules require different features to perform their tasks. Thus, it is important to choose which features better preserve the discriminant information among classes for each module. This work proposes a modular neural network architecture that selects a specific set of features per module. This approach is a topic little explored in literature since in most cases research involving modular neural networks do not perform feature selection for each particular module. The feature selection procedure is an optimization method based on the binary particle swarm optimization. Another contribution of this work is a hybrid feature selection and weighting method based on binary PSO. Experiments were carried out on public datasets and the results show that the proposed architecture achieved better accuracy rates or equal rates, however, using less features when compared to modular neural networks that do not select features per module. Experiments with the hybrid feature selection and weighting method based on optimization particle swarm show better accuracy rates when compared to other hybrids methods used in this work as comparison methods.

Keywords: Task Decomposition. Global Optimization. Particle Swarm Optimization. Modular Neural Networks. Feature Selection

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

AG	Algoritmo Genético
TS	*Tabu Search*
FIFO	*first-in-first-out*
SA	*Simulated Annealing*
PSO	*Particle Swarm Optimization*
BPSO	*Binary Particle Swarm Optimization*
IBPSO	*Improved Binary Particle Swarm Optimization*
RNA	Rede Neural Artificial
RNM	Rede Neural Modular
CT	*Crosstalk Table*
FLD	*Fisher's Linear Discriminant*
UPGMA	*Unweighted Pair-Group Method Average*
SOM	*Self-organizing Maps* (Mapas Auto-organizáveis)
LVQ	*Learning Vector Quantization*
ART	*Adaptive Resonance Theory*
OP	*Output Paralellism*
PD	*Pattern Distributor*
DP	Distribuidor de Padrões
k-NN	*k-Nearest Neighbor*
MLP	*Multilayer Perceptron*
RIF	*Relative Importance Factor*

Sumário

1 Introdução

Embora o trabalho precursor das redes neurais artificiais tenha sido originalmente proposto na década de quarenta (1), foi só na década de oitenta que essa técnica de aprendizagem de máquina recebeu uma expressiva atenção da comunidade científica. Desde então, muitos foram os trabalhos utilizando redes neurais artificiais para os mais diversos fins: reconhecimento de padrões, processamento de imagens, previsão de séries temporais, entre outros (2).

Com forte inspiração biológica, as redes neurais artificiais tiveram como modelo as redes neurais naturais presentes em nosso cérebro. Embora consolidada no meio acadêmico, as redes neurais artificiais são alvo de críticas por não serem um modelo tão fiel às redes neurais biológicas (3) (4). Uma das críticas direcionada às redes neurais artificiais foi a falta de modularização, característica inerente às redes neurais biológicas, visto que em nosso cérebro há regiões específicas do córtex cerebral responsáveis por determinadas tarefas como: visão, audição e memória, por exemplo (5). Com o propósito de modularizar o funcionamento das redes neurais artificiais, Micheli-Tzanakou conduziu um trabalho pioneiro sobre Redes Neurais Modulares (RNMs) para representar o funcionamento da retina de vertebrados (6). Esse foi o primeiro trabalho a introduzir o termo redes neurais modulares (7).

Desde então, a pesquisa em redes neurais modulares cresceu vertiginosamente (7). O objetivo das redes neurais modulares é tratar um problema grande e complexo como subproblemas menores e de mais fácil resolução. Cada um desses subproblemas é então tratado por um dos módulos que compõem a rede neural modular. Para quebrar o problema original em subproblemas, são utilizados métodos de decomposição de tarefas baseados no agrupamento de padrões semelhantes ou no agrupamento de classes semelhantes (classes próximas no espaço de características) (7).

Após o fim da decomposição, cada um dos subproblemas é então tratado por um módulo específico ou por diferentes módulos (quando se quer ter uma maior tolerância a falhas). A resposta final do sistema modular é então a combinação das diferentes respostas produzidas pelos diferentes módulos ou então a maior resposta produzida pelos módulos.

Um problema que surge ao utilizar as redes neurais modulares é o uso integral de todas as características presentes no problema original pelos diferentes módulos. Porém, como os módulos tratam problemas menores (no contexto de problemas de classificação, cada módulo, geralmente, é formado por menos classes e menos padrões que o problema inicial), é esperado que utilizem menos características para tratarem suas respectivas tarefas. Logo, é importante selecionar de alguma maneira as características mais relevantes para cada um dos módulos que formam a

topologia da rede modular, uma vez que é sabido que nem todas as características são úteis à tarefa de discriminar um determinado padrão entre as classes disponíveis. Algumas características são até prejudiciais à tarefa de classificação visto que podem apresentar redundância ou inconsistência (8).

Como um tópico pouco explorado na literatura, o presente trabalho propõe investigar o uso de seleção de características em cada um dos módulos de uma rede modular. Através da seleção de diferentes conjuntos de características por módulo, espera-se gerar redes especialistas ainda mais compactas, com menos neurônios na camada de entrada e menos interferência entre os pesos durante a atualização (9), visto que menos conexões sinápticas serão necessárias para repassar a informação da camada de entrada para a camada de saída. Com isso, é esperado conseguir uma melhora no desempenho final do sistema quando comparado a redes neurais modulares em que não há a seleção de características por módulo, acarretando assim uma diminuição do custo computacional e de armazenamento das redes neurais especialistas (redes neurais artificiais de cada módulo). Uma abordagem de rede neural modular em que cada módulo tem sua dimensionalidade reduzida utilizando técnica de extração de características em vez de técnicas de seleção de características foi desenvolvida (10). A técnica de extração de características, a saber *high dimensional data clustering* (11), realiza tanto a decomposição de tarefas quanto a redução de dimensionalidade. Entretanto, a arquitetura proposta ficou restrita a apenas uma técnica de decomposição de tarefas.

1.1 Objetivos

Ao decompor o problema original em subproblemas e atribuir cada subproblema a um módulo específico de uma rede neural modular, toda a informação necessária para separar os padrões entre as k classes existentes no problema é repassada integralmente aos diferentes módulos da RNM. Entretanto, os subproblemas, geralmente, possuirão menos classes e menos padrões, sendo provável que informação desnecessária esteja sendo repassada para cada módulo, sobrecarregando-o.

Tendo como objetivo contornar esse problema, foi proposta uma arquitetura de rede neural modular que seleciona automaticamente um conjunto de características específico para cada módulo. O método de seleção de características utilizado em cada módulo é um algoritmo de otimização global baseado em enxames de partículas (12). Com o uso de seleção de características embutida, espera-se obter módulos ainda mais compactos e taxas de classificação melhores quando comparado a arquiteturas que não fazem uso da seleção de características por módulo.

1.2 Contribuições da dissertação

Como visto na seção anterior (ver seção 1.1), ao realizar a decomposição de tarefas com consequente criação dos módulos que compõem uma rede neural modular, toda a informação

(conjunto original de características) que é utilizada para discriminar os padrões dentre as classes existentes é repassada de forma integral aos diferentes módulos. Porém, como tais módulos tratarão, geralmente, de problemas com menos classes, características irrelevantes podem estar sendo utilizadas pelos diferentes módulos com provável prejuízo ao processo de classificação. Poucos são os trabalhos na literatura que tratam da seleção de características para cada módulo específico de uma RNM, o que mostra o caráter incipiente e inovador de tal área de pesquisa. Como exemplos de trabalhos que realizam a seleção de características aplicada aos módulos de uma RNM, cabe destacar os trabalhos de Guan e Liu (8) e de Guan e Zhu (13) em que os autores reportam bons resultados (para mais detalhes sobre esses dois trabalhos, ver seção 5.2, pág. 53). Buscando investigar o uso de seleção de características aplicado a RNMs, este trabalho resultou nas seguintes contribuições:

(i) Um método híbrido de seleção e ponderação de características baseado no PSO binário.

(ii) Uma arquitetura de rede neural modular que seleciona diferentes conjuntos de características para cada um dos módulos que compõem a topologia da rede.

(iii) Dois artigos aceitos e publicados em duas grandes conferências internacionais. A saber:

- Hybrid feature selection and weighting method based on binary particle swarm optimization. In: IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI), Herndon (Virginia), USA, 2013, p. 433-438.

- A modular neural architecture that selects a different set of features per module. In: IEEE International Joint Conference on Neural Networks (IJCNN), Beijing, China, 2014, p. 1370-1374.

Dessa forma, os principais avanços destas contibuições em relação aos trabalhos anteriores é contribuir com o fomento dessa área de pesquisa já que poucos são os trabalhos de RNM que realizam essa abordagem e investigar o uso de novos métodos de seleção de características aplicados aos módulos de uma RNM. Em Guan e Liu (8), o método de seleção é baseado em algoritmos genéticos e em Guan e Zhu (13), o método de seleção é baseado em rede neural simples combinado com discriminante linear de Fisher. No presente trabalho, o método de seleção utilizado é baseado no uso do PSO binário. Como contribuição lateral, foi proposto um método híbrido de seleção e ponderação de características também baseado no PSO binário que, quando comparado aos métodos híbridos utilizados por Barros e Cavalcanti (14), mostrou resultados superiores. Os resultados obtidos nessa abordagem híbrida motivaram o uso do algoritmo PSO binário como método de seleção de características na proposta de uma nova arquitetura de RNM em que há seleção de características por módulo.

1.3 Estrutura geral do trabalho

Esta dissertação está divida em sete capítulos. Abaixo, segue uma breve descrição acerca do conteúdo presente em cada um deles.

- **Capítulo 2 - Seleção e ponderação de características**: Esse capítulo trata da importância de selecionar as características mais relevantes em uma base de dados, cobrindo as principais métricas utilizadas para mensurar a importância de um atributo para a tarefa de classificação. São destacadas também as três principais abordagens de seleção utilizadas por algoritmos de busca;
- **Capítulo 3 - Algoritmos de Otimização Global**: Define-se otimização, diferenciando o tipo global da local e são descritos em detalhes os algoritmos de otimização utilizados no presente trabalho;
- **Capítulo 4 - Redes Neurais Modulares**: Apresenta a motivação para uso de tais redes e detalha cada uma das etapas envolvidas no projeto de redes neurais modulares;
- **Capítulo 5 - Métodos Propostos**: Descreve os dois métodos propostos por essa dissertação, já levantados na seção 1.2. São apresentados os seus funcionamentos e os detalhes de implementação;
- **Capítulo 6 - Experimentos e Resultados**: São apresentados os experimentos realizados sobre os métodos propostos a fim de verificar seus desempenhos e informações acerca das bases de dados utilizadas;
- **Capítulo 7 - Conclusão**: Resume o que foi discutido ao longo do documento e propõe sugestões para a continuação do trabalho.

2
Seleção e Ponderação de Características

Este capítulo apresentará a motivação para a realização de seleção de características explicando as três principais abordagens utilizadas por algoritmos de busca. O capítulo também trata das principais métricas usadas para mensurar a importância de um determinado atributo. Por fim, é explicado o método de ponderação de características, que atribui pesos proporcionais à importância de cada característica, mas que não realizam redução de dimensionalidade.

2.1 Seleção de Características

Em um típico problema de reconhecimento de padrões, um padrão de entrada é descrito por um vetor de atributos, ora também chamado vetor de características. A representação do vetor de características pode afetar o desempenho de um sistema de reconhecimento de padrões de três maneiras. São elas:

- **Performance**: As características utilizadas para descrever os padrões fornecem uma informação a ser aprendida pelo classificador. Se essa informação não for expressiva o suficiente, o classificador terá problemas para capturá-la e utilizá-la durante a classificação. Portanto, a quantidade de informação fornecida pelo vetor de características pode limitar a performance do classificador visto que nem todas as características são necessárias à etapa de aprendizagem do mesmo, algumas até influenciam negativamente o desempenho da aprendizagem;

- **Tempo de aprendizado**: A quantidade de características presente em um padrão está relacionada ao tamanho do espaço de busca a ser explorado pelo algoritmo de aprendizagem. Portanto, a presença de muitas características irrelevantes acarreta um aumento desnecessário no espaço de busca a ser explorado e no tempo necessário para aprender a função de classificação;

- **Número de exemplos necessário**: Quanto maior for a quantidade de características presentes em uma base de dados, maior será a quantidade necessária de exemplos para treinar um classificador. A isso, chama-se maldição da dimensionalidade (15).

A seleção de características é uma etapa de pré-processamento que surge como uma alternativa para mitigar esses três tópicos (performance, tempo de aprendizado e número de exemplos

necessário) levantados acima. Seleção de características busca selecionar o subconjunto de características mais relevantes que mantêm ou aumentam a separabilidade de padrões pertencentes a classes distintas. Dessa forma, técnicas de seleção de características reduzem a dimensionalidade de um dado problema descartando aquelas que forem irrelevantes ou redundantes (16).

Há inúmeras formas para selecionar atributos. Entretanto, antes de iniciar o processo de seleção propriamente dito, é necessário encontrar uma maneira de mensurar quais atributos descrevem melhor as diferentes classes que compõem a base de dados. Há na literatura muitas maneiras de quantificar a importância de uma característica para o problema em questão. Em geral, pode-se considerar um atributo relevante para o problema de classificação aquele que quando removido tem a medida de importância considerada em relação aos demais deteriorada (16). Dentre as métricas utilizadas para medir a importância de um atributo, destacam-se (16):

- Métricas baseadas em informação: Determinam o ganho de informação de um atributo, isto é, o ganho de informação diante da ausência ou presença deste na base de dados. Como exemplo de medida baseada em informação, tem-se a entropia;
- Métricas baseadas em distância: Definem a separação entre as classes baseando-se em alguma medida de distância (distância Euclidiana, por exemplo);
- Métricas baseadas em dependência: Considera-se a correlação entre os atributos, ou seja, o quão fortemente duas características estão associadas ou correlacionadas uma com a outra. Como exemplo, pode-se citar a correlação de Pearson;
- Métricas baseadas em consistência: Buscam manter a consistência entre os atributos. A inconsistência entre atributos pode ser definida como a existência de atributos com valores muito semelhantes mas pertencentes a classes distintas;
- Métricas de precisão: Algoritmos de aprendizagem de máquina são utilizados para selecionar diferentes subconjuntos de dados. O subconjunto que apresentar maior precisão em relação ao problema será definido como o subconjunto ideal.

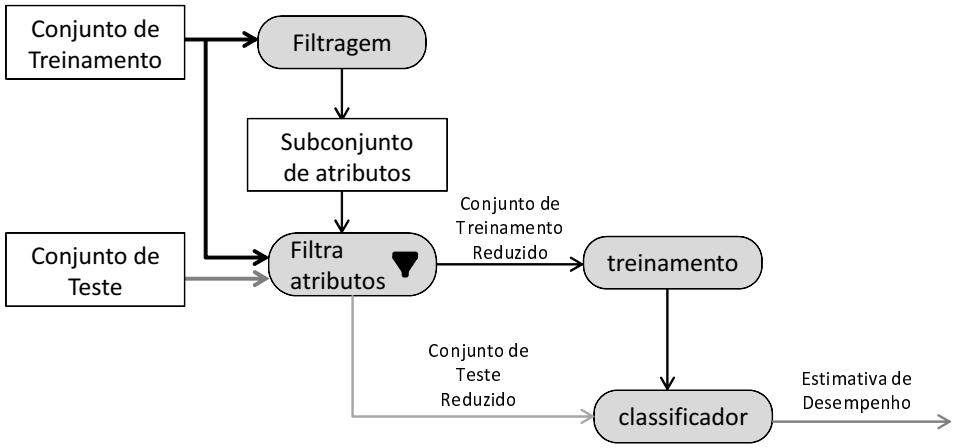
Na área de aprendizagem de máquina, a seleção de atributos se mostra como etapa muito importante antes de se proceder à tarefa de classificação. Um ponto positivo ao se usar seleção de características é a melhora do desempenho do classificador, uma vez que características redundantes ou até mesmo ruidosas, geralmente, são descartadas. Tais características atrapalham o desempenho do classificador. Outro ponto positivo é a redução do tempo de treinamento e de execução do classificador, visto que serão usadas poucas características em relação à dimensionalidade original. Também pode ser levantada a questão da compreensão acerca dos dados. Dados de grande dimensionalidade são pouco inteligíveis. Logo, com a redução de dimensão, a complexidade da base é diminuída implicando uma melhor compreensão e visualização dos dados (17).

Do exposto acima, vê-se que a solução desejada em um problema de seleção de características é o menor subconjunto de características possível que preserve, em grande parte, a informação discriminatória presente nos dados originais. Dessa forma, a tarefa de seleção pode ser vista como um problema de busca ou otimização cuja função objetivo seria melhorar a eficácia dos algoritmos de aprendizagem de máquina durante a classificação, bem como reduzir o conjunto de características utilizado. Ao avaliar os diferentes subconjuntos de características encontrados por algoritmos de busca, pode-se utilizar três abordagens distintas: abordagem filtro, abordagem *wrapper* e abordagem *embedded*. Nas seções seguintes, serão descritas em detalhes tais abordagens.

2.1.1 Abordagem Filtro

Na abordagem filtro, a seleção de características é feita em uma etapa de pré-processamento anterior à classificação. O método responsável por selecionar as características é independente do classificador utilizado. Tal método filtra as características irrelevantes com base em algum critério. O processo de seleção é rápido, porém o desempenho obtido pelo classificador após a seleção pode não ser o esperado, uma vez que o classificador não participa do processo de seleção. De acordo com a Figura 2.1, pode-se ver que após a filtragem dos atributos irrelevantes pelo algoritmo de seleção sobre o conjunto de treinamento, os conjuntos de treinamento e teste têm suas dimensões reduzidas com base no subconjunto de atributos retornado. Em seguida, o classificador é treinado com o conjunto de treinamento já reduzido e seu desempenho é avaliado sobre o conjunto de teste (18).

Figura 2.1: Abordagem Filtro.



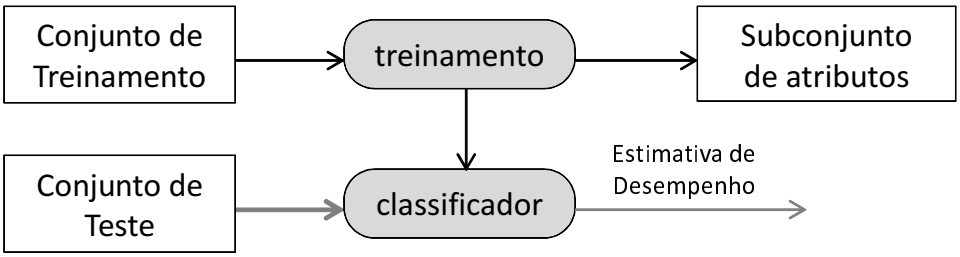
Fonte: o autor

2.1.2 Abordagem *Embedded*

Na abordagem *embedded*, a seleção do subconjunto de características é realizada de forma embutida pelo próprio classificador (a técnica que realiza a seleção de características

também é o classificador final). Essa forma de seleção converge rapidamente por não ser necessário realizar múltiplos treinamentos. Árvores de decisão (19) são um bom exemplo para essa abordagem. Nas árvores de decisão, a cada iteração, um atributo é selecionado baseando-se na sua capacidade de discriminar as classes. De acordo com a Figura 2.2, o conjunto de treinamento é utilizado pelo classificador e o resultado após o treinamento é o subconjunto de características retornado, bem como o classificador já treinado. Em seguida, o desempenho do sistema é medido com base no conjunto de teste (18).

Figura 2.2: Abordagem *Embedded*.



Fonte: o autor

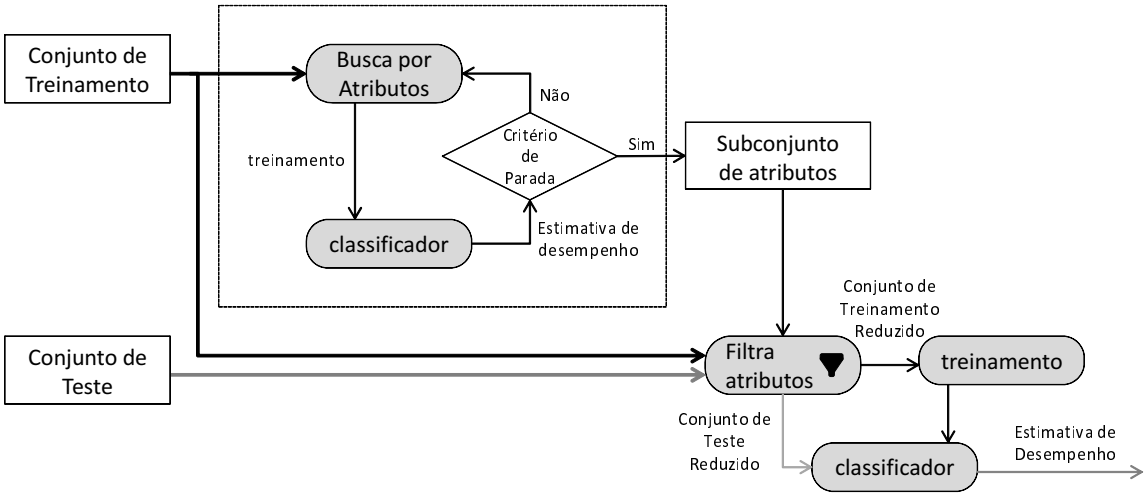
2.1.3 Abordagem *Wrapper*

Na abordagem *wrapper*, a seleção de características é feita anteriormente à classificação como na abordagem filtro. Nessa abordagem, o classificador é "empacotado" na estrutura de *loop* e funciona como uma caixa-preta, uma vez que o projetista não precisa ter conhecimento sobre o mesmo. Cada solução candidata encontrada é então avaliada pelo classificador do laço (ver Figura 2.3, pág. 22). Como critério de parada da busca, pode-se adotar um determinado limiar (taxa de acerto conseguida com o subconjunto de atributos), ou seja, caso a taxa de acerto de um determinado subconjunto for maior ou igual a um valor tido como aceitável, a busca tem fim. Caso contrário, o processo continua. Após terminada a busca, o conjunto de treinamento e o conjunto de teste têm suas características filtradas de acordo com o subconjunto retornado na etapa anterior. Por fim, o desempenho do sistema é avaliado sobre os conjuntos reduzidos por um classificador independente do classificador presente no laço. A grande desvantagem desse método é o alto custo computacional exigido, porém, geralmente, essa abordagem apresenta melhores resultados quando comparada às outras duas (18).

2.2 Ponderação de Características

Outra forma de mensurar a capacidade discriminatória das características de uma base de dados é através da ponderação. Diferentemente dos métodos de seleção de características, os métodos de ponderação não realizam redução de dimensionalidade. Em vez disso, tais métodos atribuem um peso (valor real) a cada uma das características presentes na base de dados. Esse

Figura 2.3: Abordagem *Wrapper*.



Fonte: o autor

peso é proporcional ao poder de separabilidade de cada característica em relação às classes existentes. Os algoritmos de ponderação de características podem ser divididos em dois grupos (20): (i) o primeiro grupo compreende aqueles que realizam uma busca por um conjunto de pesos através de um processo iterativo no qual cada conjunto de pesos, gerado aleatoriamente, é avaliado por um classificador e com base nessa resposta decide-se pelo uso desse conjunto de pesos ou pela geração de novos pesos e (ii) o segundo grupo compreende os métodos que calculam os pesos utilizando algum modelo pré-existente como, por exemplo, probabilidades condicionais, projeção de classe ou informação mútua (21).

Como exemplos de algoritmos pertencentes ao primeiro grupo, pode-se citar os trabalhos de Raymer et al. (22) e Punch et al. (23) em que ambos fazem uso de algoritmos genéticos combinados com a técnica k-NN na busca de um vetor de pesos ótimo para os problemas tratados. O classificador k-NN é responsável, nesses trabalhos, por fornecer um *feedback* para que os algoritmos genéticos procedam à seleção de um novo conjunto de pesos.

Como exemplo de algoritmo pertencente ao segundo grupo, tem-se o trabalho de Guvenir e Akkus (24) em que os autores propõem uma extensão do algoritmo *k*NNFP (*k Nearest Neighbor Algorithm on Feature Projections*) que incorpora a atribuição de pesos para características com o intuito de mensurar as características mais relevantes. O processo de atribuição de pesos é feito por um algoritmo proposto pelos autores (24). Outro exemplo de trabalho pertencente ao segundo grupo foi proposto por Paredes e Vidal (25). Nesse trabalho, uma medida de dissimilaridade é proposta para melhorar o desempenho do k-NN de forma que padrões pertencentes a uma mesma classe tenham pequenas distâncias enquanto que padrões de classes distintas tenham distâncias maiores. Os autores também propõem um técnica chamada programação fracional para calcular o vetor de pesos.

2.3 Considerações Finais

Neste capítulo foram apresentadas as definições e diferenças de seleção e ponderação de características, duas abordagens que tratam de determinar as características mais importantes no processo discriminatório entre classes. Também foram vistas as principais métricas utilizadas para medir a importância de um atributo. Na busca pelo menor subconjunto de características, foram vistas as três maneiras de realizar a busca (abordagens filtro, *wrapper* e *embedded*). Na parte que trata de ponderação de características, foram vistos os dois grupos de algoritmos que realizam ponderação de características com alguns exemplos de trabalhos na literatura.

3
Algoritmos de Otimização Global

Otimização é definida como a tarefa de encontrar a melhor solução dentre um conjunto de possíveis soluções (26). O processo de busca parte de uma solução inicial ou de um conjunto delas (algoritmos que trabalham com populações, como é o caso dos algoritmos genéticos (AGs), por exemplo) e, iterativamente, são realizadas operações que guiam o processo de busca para direções que contenham a melhor solução possível ou até mesmo um conjunto delas.

Em termos matemáticos, seja uma função $f : \Re^n \rightarrow \Re$. Seja S o conjunto de possíveis soluções para um determinado problema e considere o problema de minimizar $f(s)$ onde $s \in S$. Se $s^{'} \in S$ e se existe uma vizinhança de $s^{'}$ denotada por $V(s^{'}) \subset S$ de forma que $\forall\ s \in V(s^{'})$, $f(s) \geq f(s^{'})$, então $s^{'}$ é chamado de mínimo local da função f. Já se $s^{'} \in S$ e $f(s) > f(s^{'})$, $\forall\ s \in V(s^{'})$, com $s \neq s^{'}$, então $s^{'}$ é dito uma solução ótima local da função f. Das conclusões acima, vê-se que uma determinada solução $s^{'} \in S$ é dita uma solução ótima global para o problema em questão se $\forall\ s \in S$, com $s \neq s^{'}$, $f(s) > f(s^{'})$.

A busca local tem como objetivo encontrar a melhor solução em uma restrita região do espaço de busca. Já a busca global tem como meta encontrar a melhor solução possível de todo o espaço de busca. O resultado da busca local está atrelado ao ponto de busca inicial, já na busca global essa dependência é menor. Porém, o custo computacional de algoritmos de busca global é superior aos custos de algoritmos de busca local uma vez que o espaço a ser explorado é muito maior. Diante do alto custo computacional para se encontrar a solução ótima global, uma solução ótima local se mostra bastante atraente, até porque, dependendo do problema, encontrar o ótimo global torna-se inviável (26).

A seguir, serão descritos os algoritmos de otimização global utilizados neste trabalho.

3.1 *Tabu Search*

O algoritmo de busca *Tabu Search* (TS) foi desenvolvido independentemente por Glover (27) e Hansen (28) para solucionar problemas de otimização combinatória. *Tabu Search* é um método de busca iterativa caracterizado pelo uso de uma memória flexível. A cada iteração do algoritmo, um número limitado de novas soluções é gerado e avaliado. Dessa quantidade de soluções avaliadas, a melhor solução, em termos de função de custo, é aceita como a nova solução atual mesmo que o seu custo seja maior do que o custo da solução corrente. Assim, explorando e escolhendo a melhor solução de uma vizinhança como nova solução atual, o algoritmo *Tabu Search* consegue escapar de mínimos locais (29).

Durante as iterações, o algoritmo faz uso de uma lista (memória) onde são armazenadas as **T** soluções visitadas recentemente. A finalidade da lista é evitar que haja ciclos durante a busca da melhor solução na vizinhança, sinalizando quais movimentos devem ser evitados (*tabu*) nas próximas iterações (30). O tamanho da lista *tabu* é um fator crucial, que influencia diretamente o desempenho do algoritmo. Ao definir o tamanho da lista, deve-se tomar cuidado para que não seja escolhido um valor muito pequeno, limitando o espaço de busca a pequenas regiões, ocasionando assim, ciclos durante a busca. Da mesma forma, deve-se tomar cuidado com valores muito grandes, o que acarretaria um aumento no custo computacional já que a vizinhança a ser explorada seria formada por muitas soluções (30).

Definido o tamanho da lista como **T**, serão armazenadas na memória as **T** soluções recentemente visitadas. Quando o tamanho da lista é atingido, a solução mais antiga armazenada na lista é removida para que seja armazenada a nova solução que não é *tabu*, obedecendo à política FIFO (***first-in-first-out***).

O Algoritmo 3.1 contém o pseudo-código da técnica *Tabu Search*. Nele, S representa o espaço de possíveis soluções, s_{bsf} representa a melhor solução encontrada até o momento e f representa uma função de custo real. O objetivo do algoritmo é encontrar o mínimo ou máximo global da função f.

Algoritmo 3.1 Pseudo-código *Tabu Search*

1: $s_0 \leftarrow$ solução inicial $\in S$
2: lista tabu $\leftarrow s_0$
3: $I \leftarrow$ número de iterações
4: $V \leftarrow$ vizinhança de uma solução
5: $s_{bsf} \leftarrow s_0$ (melhor solução até o momento)
6: **para** $i = 0 \rightarrow I - 1$ **faça**
7: gerar a vizinhança V de s_i
8: encontrar a melhor solução $s^{'}$ de V que não seja *tabu*
9: $s_{i+1} \leftarrow s^{'}$
10: Atualizar lista tabu inserindo s_{i+1}
11: Atualizar s_{bsf} **se** $f(s_{i+1}) < f(s_{bsf})$
12: **fim para**
retorne s_{bsf}

O algoritmo tem fim quando é atingida a quantidade máxima de iterações (I), retornando a melhor solução s_{bsf}. A quantidade de vizinhos a ser gerada é um parâmetro variável do algoritmo e a maneira como eles são gerados está atrelada à codificação das soluções.

3.2 *Simulated Annealing*

A técnica *Simulated Annealing* (SA) é inspirada no processo de esfriamento de sólidos, que têm reduzidas as suas temperaturas lentamente até que sejam formadas estruturas cristalinas perfeitas (31). Na analogia entre um problema de otimização combinatória e o processo de

aquecimento (*annealing*), os estados dos sólidos representam as possíveis soluções para o problema de otimização, as energias dos estados correspondem aos valores da função objetivo computados para estas soluções e o estado de energia mínima corresponde à solução ótima do problema (30).

O algoritmo é formado por uma sequência de iterações em que cada iteração consiste em mudar aleatoriamente a solução corrente, a fim de criar uma nova solução na vizinhança que seja melhor do que a solução atual (31). Uma vez criada a nova solução, o seu custo é computado e a variação no custo (ΔE) é utilizada para decidir pela aceitação ou não da nova solução como solução atual. Se o custo da nova solução for menor do que o custo da solução atual, ela é imediatamente aceita como solução atual. Caso contrário, ela pode ser aceita ou não de acordo com o critério *Metropolis*, baseado na probabilidade de *Boltzman* (32).

De acordo com o critério *Metropolis*, um número aleatório δ no intervalo [0,1] é gerado a partir de uma distribuição uniforme. Se $\delta \leq e^{(-\Delta E/T)}$ onde ΔE representa a variação dos custos entre a solução atual e a nova solução e T representa o parâmetro temperatura presente no algoritmo, então a solução gerada é aceita como solução atual. Do contrário, a solução atual não sofre alteração e o processo continua. Existem vários esquemas de resfriamento responsáveis tanto por definir a temperatura inicial quanto pela regra que iterativamente reduz o valor da temperatura. Por ser um dos esquemas mais utilizados, foi adotado o esquema de resfriamento geométrico (30). O Algoritmo 3.2 contém o pseudo-código da técnica *Simulated Annealing*:

Algoritmo 3.2 Pseudo-código *Simulated Annealing*

```
1:  S_0 ← solução inicial ∈ S
2:  T_0 ← Temperatura inicial
3:  I ← número de iterações
4:  para i = 0 → I − 1 faça
5:      gera nova solução S'
6:      se f(S') < f(S_i) então
7:          S_{i+1} ← S'
8:      senão
9:          S_{i+1} ← S' com probabilidade e^{−[f(S')−f(S_i)]/T_{i+1}}
10:     fim se
11:     i ← i + 1
12:     Atualiza temperatura T_i
13: fim para
        retorne S_i
```

No Algoritmo 3.2, S representa o espaço de possíveis soluções e f representa uma função de custo real. O objetivo do algoritmo é encontrar o mínimo global s, tal que $f(s) < f(s^{'})$, $\forall$ $s^{'} \in S$. O algoritmo tem fim quando é atingida a quantidade máxima de iterações I, retornando a melhor solução encontrada até então (solução atual).

3.3 Algoritmos Genéticos

Os algoritmos genéticos (AGs) foram inspirados na teoria da evolução das espécies de Darwin (33). Diferentemente de outras técnicas de otimização, os AGs realizam uma busca paralela no espaço de soluções, visto que trabalham com um conjunto de soluções candidatas simultaneamente. A esse conjunto de soluções é dado o nome de população. Cada solução candidata que compõe a população recebe o nome de indivíduo ou ainda cromossomo. A cada indivíduo da população está associado um valor denominado aptidão, que mede a qualidade da solução para resolver o problema em questão.

Os AGs trabalham em cima de uma população inicial. A formação dessa população inicial se dá através de duas maneiras: uma forma consiste em usar soluções produzidas aleatoriamente. Esse método é aconselhado para problemas em que não há um conhecimento *a priori* sobre a natureza do problema. A outra maneira emprega um conhecimento *a priori* sobre o problema de otimização em questão. Neste caso, esse conhecimento é usado para obter um conjunto de critérios de forma que as soluções que satisfaçam esses critérios são reunidas para formar a população inicial. A diferença principal entre os dois métodos, em termos de eficiência, está no tempo necessário para convergir a uma solução ótima ou próxima da ótima. A segunda maneira, geralmente, requer menos tempo do que a primeira para convergir (30).

Os operadores genéticos são responsáveis pela seleção dos indivíduos mais aptos da população para reproduzirem e pela garantia da diversidade gênica. O operador genético de seleção desempenha um importante papel no processo evolucionário, que é o de direcionar a busca para regiões onde se encontram as melhores soluções para o problema abordado. Alguns dos principais operadores de seleção são o método de seleção proporcional (método da roleta) e a seleção baseada em torneio. No método da roleta, os indivíduos da população ocupam espaços (*slots*) na roleta proporcionais aos seus valores de aptidão, de forma que indivíduos com maiores valores de aptidão terão maiores chances de serem selecionados para a fase de reprodução, porém, há o risco do indivíduo mais apto não ser selecionado.

Já no método de seleção por torneio, os indivíduos são selecionados aos pares e o indivíduo que possuir maior valor de aptidão de cada par é selecionado para a fase de reprodução. A desvantagem desse forma de seleção é que indivíduos com os menores valores de aptidão da população não terão chances de serem selecionados. No método da roleta, ainda há a chance mesmo que pequena.

Depois de selecionados os indivíduos, tem início o processo de reprodução. Os operadores genéticos responsáveis pela etapa de reprodução são os operadores de mutação e recombinação gênica (*crossover*). A mutação é um operador unário, isto é, trabalha em cima de um único indivíduo invertendo os seus *bits* de acordo com uma certa probabilidade. Por sua vez, a recombinação gênica é um operador que trabalha com dois ou mais indivíduos, no qual novos indivíduos são gerados a partir das informações presentes nos indivíduos pais. Tanto o *crossover* quanto o operador de mutação adicionam diversidade à população, permitindo que novas áreas

do espaço de busca sejam exploradas (34). O Algoritmo 3.3 contém o pseudo-código de um algoritmo genético típico.

Algoritmo 3.3 Pseudo-código Algoritmo Genético

```
1: f ← função de aptidão
2: k ← número de indivíduos na população
3: p ← população inicial com k indivíduos gerados aleatoriamente
4: g ← número de indivíduos que permanece na geração seguinte
5: r ← fração da população substituída após crossover
6: m ← taxa de mutação
7: h ← indivíduo
8: para i = 0 → k faça
9:     Para cada h em p, calcular f(h)
10: fim para
11: while critério de parada não é atingido faça
12:     Criar uma nova geração p_n
13:     Selecionar com base na aptidão (1 - r)*g membros de p para adicionar em p_n
14:     Selecionar r*g/2 pares de indivíduos de p com base na aptidão para reproduzirem
15:     Adicionar a descendência produzida no passo 15 em p_n
16:     Selecionar m por cento dos membros de p_n para terem um bit invertido aleatoriamente
17:     p ← p_n
18:     para i = 0 → k faça
19:         Para cada h em p, calcular f(h)
20:     fim para
21: fim while
        retorne Retornar o indivíduo de p com maior valor de aptidão
```

3.4 *Particle Swarm Optimization*

PSO (*Particle Swarm Optimization*) é uma técnica de otimização baseada em populações, proposta inicialmente por Kennedy e Eberhart (35) em 1995. O algoritmo PSO, assim como outros algoritmos evolucionários, é um algoritmo estocástico, que não necessita de informação sobre o gradiente descendente derivado da função erro. A ideia por trás do PSO é que a partir de simples interações locais entre os indivíduos, emergem comportamentos coletivos mais complexos como andar em bandos ou explorar ambientes (36).

Diferentemente de outras técnicas de otimização, o PSO trabalha simultaneamente com uma população de soluções candidatas em que cada solução recebe o nome de partícula, daí o nome *particle swarm*. O PSO é um algoritmo iterativo cujas soluções candidatas são continuamente melhoradas. As partículas compartilham informações entre si de tal maneira que elas lembram a melhor solução visitada até o momento (região onde ela foi encontrada). Informações sobre mínimos globais também são compartilhadas por toda a população. Baseando-se nessa interação entre os indivíduos da população, geralmente, chega-se a uma melhor solução em comparação a métodos em que não há essa simbiose.

Em comparação com os algoritmos genéticos, que também são algoritmos evolutivos que trabalham com população, há diferenças claras entre os dois. Enquanto os AGs apenas rastreiam a posição das melhores soluções, o PSO mantém informação sobre posição (local no espaço de busca em que se encontra a solução candidata) e velocidade (vetor adicionado à posição da partícula e que é responsável por direcionar o processo de busca) da melhor solução encontrada por um determinado indivíduo, bem como mantém informações sobre a melhor solução encontrada por todo o enxame (37). Além disso, o PSO não faz uso de operadores de mutação e *crossover* como os AGs. A diversidade da população é obtida através de operações matemáticas simples, responsáveis por atualizar a posição e velocidade das partículas. Dessa forma, os principais atrativos do algoritmo PSO são a facilidade de implementação, baixo custo de memória e velocidade, além de ser possível utilizar o PSO em problemas em que é computacionalmente caro obter o gradiente de uma função.

Cada partícula que compõe o enxame pode ser vista como uma solução potencial para o problema em questão. A cada iteração do algoritmo, as soluções são avaliadas através de uma função objetivo que atribui às mesmas um valor de aptidão que representa o quão boa é aquela solução para o problema que está sendo tratado. Assim, cada solução candidata pode ser vista como uma partícula "voando" através do espaço de busca à procura do mínimo ou máximo da função objetivo. Basicamente, o PSO é formado por três vetores com n componentes em que n representa a dimensão do espaço de busca. Para cada dimensão *j* que compõe uma partícula *i*, temos:

(i) $\mathbf{x}_{ij}$, que representa a posição atual

(ii) $\mathbf{v}_{ij}$, que representa a velocidade atual

(iii) $\mathbf{y}_{ij}$, que representa a melhor posição visitada pela partícula

Durante o processo iterativo, a posição de cada partícula é alterada pela adição da velocidade à posição atual que é definida por:

$$\mathbf{x}_{ij}(t+1) = \mathbf{x}_{ij}(t) + \mathbf{v}_{ij}(t+1) \tag{3.1}$$

Na Equação 3.1, o vetor velocidade é responsável por direcionar o processo de busca e reflete tanto o conhecimento adquirido pela partícula quanto o conhecimento proveniente da interação com as outras partículas que compõem o enxame. A atualização do vetor velocidade é dada da seguinte forma:

$$\mathbf{v}_{ij}(t+1) = \mathbf{v}_{ij}(t) + c_1 r_{1j}[\mathbf{y}_{ij}(t) - \mathbf{x}_{ij}(t)] + c_2 r_{2j}[\hat{\mathbf{y}}_j(t) - \mathbf{x}_{ij}(t)] \tag{3.2}$$

Na Equação 3.2, os termos r_1 e r_2 são valores aleatórios definidos no intervalo [0,1] responsáveis por introduzir um comportamento estocástico ao algoritmo. Os segundo e terceiro termos da soma na Equação 3.2 representam, respectivamente, os componentes cognitivo e

social. O componente cognitivo indica a melhor posição visitada por uma determinada partícula enquanto o componente social indica a melhor posição já alcançada por todo o enxame (melhor posição global). A melhor posição global é representada por $\hat{\mathbf{y}}$ enquanto a melhor posição local de cada partícula i é denotada por $\mathbf{y}_i$. Outros trabalhos que fazem uso do algoritmo PSO muitas vezes tratam $\hat{\mathbf{y}}$ e $\mathbf{y}_i$ como *gbest* e *pbest*, respectivamente.

Por sua vez, os termos c_1 e c_2, que geralmente possuem valores no intervalo [0,2], são constantes positivas utilizadas para dimensionar a contribuição dos componentes cognitivo e social na atualização da velocidade. Também chamados de coeficientes de aceleração, os termos c_1 e c_2, em outras palavras, indicam o tamanho do passo a ser dado pela partícula, a cada iteração, na direção da melhor posição individual e global encontradas até o momento (38).

Um aspecto importante que tem um impacto na eficácia de uma técnica de otimização está relacionado ao equilíbrio entre a exploração global e a exploração local (refinamento da busca). A exploração global define a capacidade do algoritmo em explorar diferentes regiões do espaço à procura de um bom ótimo. Em contrapartida, a exploração local é a habilidade do algoritmo em concentrar a busca a pequenas regiões promissoras com o intuito de refinar (aprimorar) a solução candidata (36).

No contexto do PSO, esse balanceamento entre exploração e refinamento é conseguido através da limitação da velocidade $\mathbf{v}_{ij}$. Assim, o valor da velocidade em cada dimensão fica restrito ao intervalo $[-v_{max}, v_{max}]$, buscando dessa forma reduzir as chances que uma partícula tem de extrapolar os limites do espaço de busca. Se o espaço de busca é definido pelas fronteiras $[-x_{max}, x_{max}]$, então o valor de v_{max} é definido como $v_{max} = k \times x_{max}$ onde $0.1 \leq k \leq 1.0$ (37).

Com a boa aceitação do PSO na comunidade científica, muitos foram os trabalhos que utilizaram o PSO como técnica de otimização e muitas foram as modificações propostas ao PSO original. Uma das modificações largamente utilizadas foi a introdução do peso inercial (w) na atualização da velocidade, proposta por Shi e Eberhart (38). O peso inercial controla o grau de exploração do espaço de busca e geralmente seu valor decresce linearmente de 0.9 a 0.4. Isso permite ao PSO explorar grandes áreas do espaço de busca no início da execução e gradualmente ir refinando a busca pelas melhores soluções a regiões restritas do espaço. Assim, com a introdução do peso inercial, a atualização da velocidade das partículas fica definida pela Equação 3.3.

$$\mathbf{v}_{ij}(t+1) = w\mathbf{v}_{ij}(t) + c_1 r_{1j}[\mathbf{y}_{ij}(t) - \mathbf{x}_{ij}(t)] + c_2 r_{2j}[\hat{\mathbf{y}}_j(t) - \mathbf{x}_{ij}(t)] \tag{3.3}$$

Uma outra modificação feita no tradicional algoritmo PSO está relacionada à natureza do problema. O PSO foi concebido originalmente como uma técnica de otimização para espaços contínuos. Entretanto, muitos problemas de otimização ocorrem em espaços discretos. Visando à adequação do algoritmo a esses casos, Kennedy e Eberhart propuseram o PSO para universos discretos e batizaram-no como BPSO (*Binary Particle Swarm Optimization*) (39). Na seção seguinte, será descrito o BPSO, que pouco difere do PSO original, e o IBPSO (*Improved Binary*

Particle Swarm Optimization) (12), que resulta de uma ligeira modificação no BPSO.

3.4.1 *Binary Particle Swarm Optimization*

No PSO discreto, as partículas podem ser vistas como "pontos" se aproximando ou se afastando das arestas de um hipercubo através da inversão de vários *bits* que compõem a partícula (39). Em sua versão discreta, os valores de $\boldsymbol{x}_i$ (posição de uma partícula) e $\boldsymbol{y}_i$ (melhor posição visitada por uma partícula) são restritos ao conjunto (0,1). Porém, não há restrição quanto ao valor da velocidade de cada partícula, ou seja, o valor da velocidade pode assumir valores reais.

Como a velocidade é responsável por alterar a posição de cada partícula que compõe o enxame e como no PSO discreto as posições assumem os valores 0 ou 1, a velocidade aqui passa a ser vista como uma probabilidade que representa a chance de uma determinada posição da partícula ter seu valor trocado. Vista agora como uma probabilidade, a velocidade no PSO binário passa por um processo de normalização através da função sigmoide logística, para assegurar que seus valores estejam no intervalo [0,1]. Essa normalização é definida pela Equação 3.4.

$$\mathbf{v}'_{ij}(t) = sig(\mathbf{v}_{ij}(t)) = \frac{1}{1+e^{-\mathbf{v}_{ij}(t)}} \tag{3.4}$$

Após a normalização da velocidade, o seu valor passa a indicar a chance do *bit* de uma determinada posição da partícula ter seu valor trocado. Por exemplo, supondo que a velocidade, após a aplicação da função sigmoide, seja 0.4, isso indicaria uma chance de 40% do *bit* assumir valor 1 e 60% de chance do *bit* assumir valor 0. Como a velocidade passa a ter uma significado probabilístico no PSO binário, a maneira de atualizar a posição de uma partícula difere do PSO original, entretanto, a forma como é atualizada a velocidade da partícula continua a mesma descrita pela equação 3.3. A forma de atualizar as posições das partículas passa a ser:

$$\mathbf{x}_{ij}(t+1) = \begin{cases} 1 & \text{se } r_{3j} < sig(\mathbf{v}_{ij}(t+1)), \\ 0 & \text{se } r_{3j} \geq sig(\mathbf{v}_{ij}(t+1)). \end{cases} \tag{3.5}$$

Na Equação 3.5, $r_{3j}(t) \sim U(0,1)$. Em outras palavras, é gerado um número aleatório no intervalo [0,1] para cada dimensão da partícula e seu valor é comparado com a probabilidade de cada dimensão para decidir se o *bit* será trocado ou não de acordo com a equação acima.

Chuang et al. (12) fizeram uso do PSO binário com uma ligeira modificação para selecionar as características mais relevantes de dados de expressão gênica. Os autores observaram que caso a melhor solução global encontrada pelo enxame de partículas permanecesse a mesma por um determinado número de iterações, ficaria configurado que a melhor solução global ($\hat{\mathbf{y}}$) estaria presa em uma região de ótimo local. Para escapar dessa região de ótimo local e alcançar melhores resultados de classificação, selecionando um menor número de genes (características), os autores propuseram "resetar" a partícula $\hat{\mathbf{y}}$ para a posição zero. Essa nova variação do algoritmo BPSO, batizada pelos autores de IBPSO (*Improved Binary Particle Swarm Optimization*), evita as regiões de mínimo, procurando por regiões mais promissoras (altas taxas de classificação) e

que tenham um baixo número de características (genes no caso do trabalho original) uma vez que os demais genes buscam se aproximar do $\hat{\mathbf{y}}$ que está na posição zero (sem características) (12). Empiricamente, os autores definiram que três iterações, com o valor de $\hat{\mathbf{y}}$ permanecendo imutável, seriam suficientes para afirmar que o $\hat{\mathbf{y}}$ estaria preso em uma região de ótimo local.

Ao *resetar* a partícula $\hat{\mathbf{y}}$ para zero, as melhores posições encontradas por cada partícula ($\mathbf{y}_i$) são mantidas. O ato de resetar a posição da melhor solução global sinaliza que nenhuma das características é selecionada para formar o melhor subconjunto de característcias. O algoritmo tem continuidade com o $\hat{\mathbf{y}}$ sendo zero. Ao decorrer do processo iterativo, as partículas que compõem o enxame modificariam suas posições em direção à posição $\hat{\mathbf{y}}$ e o próprio $\hat{\mathbf{y}}$ se deslocaria para novas regiões do espaço de busca. O Algoritmo 3.4 contém o pseudo-código da técnica IBPSO que é idêntico ao pseudo-código do algoritmo BPSO a não ser pelas instruções presentes nas linhas 11, 12 e 13:

Algoritmo 3.4 Pseudo-código *Improved Binary Particle Swarm Optimization*

1: Inicializar aleatoriamente o enxame de partículas
2: **while** número de iterações ou critério de parada não é atingido **faça**
3: Avaliar o valor de aptidão de cada partícula através do 1-NN (vizinho mais próximo. Algoritmo k-NN com k igual a 1)
4: **para** $i = 1 \rightarrow$ número de partículas **faça**
5: **se** Aptidão de X_i é maior que a aptidão de $\mathbf{y}_i$ **então**
6: $\mathbf{y}_i \leftarrow X_i$
7: **fim se**
8: **se** Há no enxame uma partícula com aptidão maior que $\hat{\mathbf{y}}$ **então**
9: $\hat{\mathbf{y}} \leftarrow$ posição dessa partícula
10: **fim se**
11: **se** Aptidão de $\hat{\mathbf{y}}$ é a mesma **Max** vezes **então**
12: resetar $\hat{\mathbf{y}}$
13: **fim se**
14: **para** $j = 1 \rightarrow$ número de dimensões da partícula **faça**
15: $\mathbf{v}_{ij}(t+1) = w\mathbf{v}_{ij}(t) + c_1 r_{1j}[\mathbf{y}_{ij}(t) - \mathbf{x}_{ij}(t)] + c_2 r_{2j}[\hat{\mathbf{y}}_j(t) - \mathbf{x}_{ij}(t)]$
16: **se** $\mathbf{v}_{ij}(t+1) \notin (V_{min}, V_{max})$ **então**
17: $\mathbf{v}_{ij}(t+1) = max(min(V_{max}, \mathbf{v}_{ij}(t+1)), -V_{max})$
18: **fim se**
19: $sig(\mathbf{v}_{ij}(t+1)) = \frac{1}{1+e^{-\mathbf{v}_{ij}(t+1)}}$
20: **se** $r_{3j} < sig(\mathbf{v}_{ij}(t+1))$ **então**
21: $\mathbf{x}_{ij}(t+1) \leftarrow 1$
22: **senão**
23: $\mathbf{x}_{ij}(t+1) \leftarrow 0$
24: **fim se**
25: **fim para**
26: **fim para**
27: Continuar o algoritmo com a próxima geração até o critério de parada ser atingido
28: **fim while**

Como dito anteriormente, para que um algoritmo de otimização tenha sucesso, é necessá-

rio garantir o equilíbrio entre a exploração global e local. No BPSO, esse equilíbrio é realizado através da limitação das velocidades máxima e mínima de uma partícula. O trecho de código da linha 16 até a linha 18 do algoritmo IBPSO garante que uma partícula não extrapole os limites do espaço de busca através, justamente, da restrição das velocidades.

Após o cálculo das novas velocidades, é gerado um número aleatório no intervalo [0,1] para cada dimensão da partícula (r_{3j}) que por sua vez é comparado com as respectivas probabilidades de cada dimensão da partícula. Se o valor de (r_{3j}) for menor do que a sua respectiva probabilidade, então aquela dimensão (característica no contexto de problemas de seleção) é selecionada como característica necessária para a próxima iteração, senão a referida dimensão não é selecionada para compor a próxima iteração.

3.5 Considerações Finais

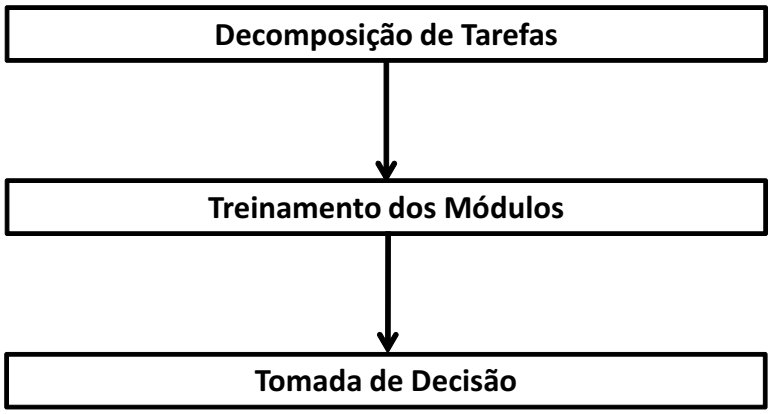
Neste capítulo foi visto o significado de otimização. Sublinhou-se a diferença entre otimização local e global. Foram detalhados os algoritmos de otimização utilizados no presente trabalho como *Tabu Search*, *Simulated Annealing* e *Particle Swarm Optimization*. Além do PSO tradicional, foram vistas em detalhes a versão binária do PSO e uma versão melhorada do PSO binário que foram utilizadas pelas técnicas propostas nesta dissertação.

4
Redes Neurais Modulares

As redes neurais artificias (RNAs) foram inspiradas nas redes neurais biológicas presentes nos cérebros de seres humanos. As principais características em comum com as redes neurais biológicas são a capacidade de processamento de informação de forma paralela e distribuída, a presença de unidades de processamento simples, a presença de detectores de características e a capacidade de aprendizado através de exemplos (1). Entretanto, tais redes não apresentam uma característica inerente às redes neurais biológicas: modularização, visto que nosso cérebro possui áreas especializadas distintas (módulos) responsáveis por tarefas específicas como: visão, audição e fala (5). Logo, para aproximar ainda mais as redes neurais artificias das redes neurais biológicas, foram propostas as redes neurais modulares (RNMs). Tais redes consistem de vários módulos (redes neurais simples) que funcionam de forma integrada, sendo cada módulo responsável por solucionar uma subtarefa, menor e mais simples que o problema original.

Com o aumento no número de pesquisas sobre redes neurais modulares, muitas foram as arquiteturas propostas para resolver uma variedade de problemas (7). Contudo, tais arquiteturas não apresentam um padrão bem definido como as RNAs, por exemplo. Analisando diferentes arquiteturas, Auda e Kamel (7) sugeriram três etapas principais como sendo comuns à maioria das arquiteturas de RNMs: decomposição de tarefas, treinamento dos módulos e tomada de decisão (ver Figura 4.1).

Figura 4.1: Três etapas principais de uma arquitetura de rede neural modular.



Fonte: o autor

A decomposição de tarefas nada mais é que a decomposição do problema original em vários subproblemas (subtarefas). Dessa forma, as subtarefas representam pequenos pedaços da

tarefa inicial que podem ser solucionados separadamente por cada um dos módulos. Geralmente, cada módulo fica responsável por solucionar uma subtarefa diferente. Em casos em que se preza por uma maior tolerância a falhas, diferentes módulos ficam responsáveis por solucionar uma mesma subtarefa. A decomposição de tarefas apresenta a mesma ideia da abordagem dividir-para-conquistar.

Após a formação dos módulos, a rede é treinada, podendo ser em paralelo, uma vez que as subtarefas são problemas independentes ou em uma certa ordem específica, caso a arquitetura apresente módulos dispostos em camadas (resposta de um módulo serve como entrada para outro imediatamente seguinte). Uma vez treinada a rede, os módulos passam a representar especialistas responsáveis por tratar um determinado subproblema. Quando tem fim a etapa de treinamento, inicia-se a integração dos diferentes resultados produzidos pelos diferentes especialistas em uma única decisão (resposta), fase da tomada de decisão.

Uma vez que o desempenho da rede neural modular é função dessas três etapas (decomposição de tarefas, treinamento dos módulos e tomada de decisão), a RNM precisa fornecer um equilíbrio razoável entre a simplificação conseguida através da decomposição e a eficácia da tomada de decisão. Enquanto a etapa de decomposição de tarefas tenta produzir subtarefas as mais simples possíveis, os módulos buscam fornecer a informação estratégica necessária para que a etapa da tomada de decisão tome uma decisão global precisa (7). Essas três etapas se bem conduzidas levam a uma arquiterura de RNM que, geralmente, implica algum ganho para o tratamento do problema, seja em taxas de acerto durante a classificação, simplicidade nas topologias ou em tempo de processamento. Nas próximas seções, cada uma dessas três etapas envolvidas na concepção de uma rede neural modular será descrita e casos de cada uma delas serão exemplificados.

4.1 Decomposição de Tarefas

Sendo uma das etapas cruciais ao se projetar uma arquitetura de rede neural modular, a decomposição de tarefas divide um problema complexo em subproblemas mais simples e fáceis de serem resolvidos. Ao fazer a decomposição, automaticamente tem-se a quantidade de módulos resultante que formará a arquitetura final. As maneiras mais utilizadas para gerar os módulos são baseadas no agrupamento de classes e no agrupamento de padrões semelhantes.

As técnicas de decomposição de tarefas ainda podem ser classificadas quanto à automatização, isto é, se a decomposição se dá de forma manual ou automática. Na decomposição manual, especialistas em um determinado tipo de problema são responsáveis pela etapa de decomposição. Dois exemplos desse tipo de decomposição são a decomposição explícita e a decomposição manual pelas classes.

Na decomposição explícita, o especialista pode separar o volume de dados em dois módulos de acordo com a presença de ruídos ou no caso de desbalanceamento entre classes, por exemplo. Dessa forma, um módulo conteria os dados ruidosos e o outro conteria os dados sem

ruídos bem como um módulo abrangeria os padrões das classes minoritárias (difícil classificação devido à insuficiência de dados) e o outro módulo, os padrões das classes majoritárias. Já na decomposição manual pelas classes, a decomposição se baseia na similaridade entre classes. Como exemplo, pode-se citar o reconhecimento de dígitos e caracteres manuscritos. Dada a similaridade entre os dígitos 1 e 7 e os caracteres "e" e "i", por exemplo, padrões desses grupos de classes semelhantes são agrupados em módulos separados, responsáveis por suas classificações, uma vez que são mais difíceis de serem diferenciados.

Diante da impraticabilidade de se usar métodos de decomposição manuais para uma variedade de problemas devido à dificuldade da presença de especialistas, além da elevada possibilidade de haver estimativas erradas acerca da quantidade de módulos, faz-se necessário o uso de métodos de decomposição de tarefas automáticos. Algumas técnicas de decomposição de tarefas automáticas descritas a seguir compreendem: algoritmos genéticos, *crosstalk tables* e decomposição por agrupamentos, que por sua vez abrange os mapas auto-organizáveis e o algoritmo k-médias.

4.1.1 Algoritmos Genéticos

Uma forma automática de decomposição de tarefas por agrupamento de classes são os algoritmos genéticos (40). Nessa abordagem, cada indivíduo **p** é um vetor de tamanho n, onde n representa o número de classes do problema tratado. Cada gene p_i do cromossomo **p** varia de 1 a k em que k é o número de grupos. Dessa forma, a classe c_i pertence ao grupo (módulo) p_i. Tomando como exemplo o cromossomo **p** = {31231}, teríamos os seguintes grupos formados: {{2,5},{3},{1,4}}. Com base nos grupos formados e no genótipo do cromossomo **p**, depreende-se que a classe 1 pertence ao grupo 3, a classe 2 pertence ao grupo 1, a classe 3 pertence ao grupo 2, a classe 4 pertence ao grupo 3 e a classe 5 pertence ao grupo 1.

Uma vez formados os grupos, cada cromossomo precisa ter seu valor de aptidão calculado. Para isso, monta-se uma topologia de rede neural modular baseada no genótipo do cromossomo. Cada módulo é tratado por uma rede neural simples (*multilayer perceptron* , por exemplo). Depois de treinada, a rede neural modular representada pelo genótipo do cromossomo é avaliada sobre o conjunto de teste. O valor de aptidão do cromossomo será, portanto, a taxa de acerto obtida durante a classificação. Dessa forma, quanto maior o valor de aptidão (taxa de acerto) do cromossomo, maiores serão as suas chances de ser selecionado para a próxima geração do algoritmo genético.

Como nos AGs a geração dos cromossomos se dá de forma automática, pode haver casos de dois ou mais cromossomos com genótipos diferentes mas mesmos fenótipos. Por exemplo, os cromossomos $\mathbf{p1} = \{1,1,2,2\}$ e $\mathbf{p2} = \{2,2,1,1\}$ possuem genótipos diferentes, mas seus fenótipos (grupos formados) são idênticos, i.e., {{1,2},{3,4}}. Para evitar que haja tal anomalia, os indivíduos são normalizados de acordo com o Algoritmo 4.1 (40).

Algoritmo 4.1 Normalização de Cromossomos

```
1: p ← cromossomo de tamanho n
2: p_norm ← cromossomo normalizado
3: k ← quantidade de grupos
4: k ← valorMaximo(p)
5: t ← 1
6: p_norm ← −p;
7: para i = 1 → k faça
8:     se p_norm[i] ≤ 0 então
9:         para j = (i+1) → n faça
10:            se p_norm[i] == p_norm[j] então
11:                p_norm[j] ← t
12:            fim se
13:        fim para
14:        p_norm[i] ← t
15:        t ← t + 1
16:    fim se
17: fim para
        retorne p_norm
```

4.1.2 *Crosstalk Table*

Uma forma automática de decomposição de tarefas baseada no agrupamento de padrões por classes é o *Crosstalk Tables* (CT) (40). O *Crosstalk Tables* nada mais é que uma matriz de distâncias entres as classes. Esse método visa agrupar dados de classes semelhantes, isto é, classes cujos padrões estejam próximos no espaço de características. No contexto de RNMs, as classes mais próximas tendem a ficar em um mesmo módulo já que são mais difíceis de serem discriminadas.

Para facilitar o cálculo da distância entre as classes, os dados são projetados para uma única dimensão através do discriminante linear de Fisher (*Fisher's Linear Discriminant - FLD*) (41) (42). Para problemas com duas classes, o FLD projeta todos os padrões de uma classe para uma única dimensão através de uma matriz de transformação (**w**), como pode ser visto na Equação 4.1.

$$x_{proj} = \mathbf{w}^t \mathbf{x} \tag{4.1}$$

Ao projetar os dados para uma única dimensão, torna-se fácil calcular a distância entre as classes bastando para isso calcular a distância entre os centroides (centros de massa das classes).

A matriz de transformação **w** da Equação 4.1 é calculada de forma a maximizar a dispersão entre-classes e minimizar a dispersão intra-classes, ou seja, dados de classes distintas têm de estar mais distantes uns dos outros e dados de uma mesma classe devem estar mais coesos. Seu cálculo é mostrado na Equação 4.2.

$$\mathbf{w} = \frac{(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j)}{\mathbf{S}_i + \mathbf{S}_j} \tag{4.2}$$

Na Equação 4.2, $\bar{\mathbf{x}}_i$ e $\bar{\mathbf{x}}_j$ são as médias dos padrões das classes i e j, enquanto $\mathbf{S}_i$ e $\mathbf{S}_j$ são as matrizes de dispersão, que são calculadas de acordo com a Equação 4.3 ($\mathbf{c}_i$ representa todos os padrões da classe i).

$$\mathbf{S}_i = \sum_{\mathbf{x} \in \mathbf{c}_i} (\mathbf{x} - \bar{\mathbf{x}}_i)(\mathbf{x} - \bar{\mathbf{x}}_i)^t \tag{4.3}$$

O processo é repetido para todas as classes, tomadas duas a duas. Ao fim do processo, tem-se a tabela de distâncias (*crosstalk table*) entre todas as classes. Guan et al. (40) não deixam claro como são formados os módulos após a obtenção da matriz de distâncias. Alves e Cavalcanti (43) propuseram o uso do algoritmo *Unweighted Pair-Group Method Average* (UPGMA) para formar os módulos com base na *crosstalk table*. Neste trabalho, a técnica *crosstalk table* combinada com o algoritmo UPGMA foi utilizada como uma das técnicas de decomposição de tarefas. A seguir, o algoritmo UPGMA é descrito brevemente.

4.1.2.1 *Unweighted Pair-Group Method Average*

O algoritmo *Unweighted Pair-Group Method Average* (UPGMA) é um algoritmo de agrupamento hierárquico. No agrupamento hierárquico, busca-se formar grupos de dados homogêneos baseando-se em medidas de similaridades entre os mesmos. O resultado final desse tipo de agrupamento é uma estrutura em árvore. Os algoritmos de agrupamentos hierárquicos se dividem basicamente em duas categorias: aglomerativa ou divisiva (44). Essas duas abordagens diferem entre si pela maneira como são formadas as árvores resultantes.

Na abordagem aglomerativa, a construção da árvore se dá de baixo para cima (*bottom-up*), isto é, das folhas até a raiz. As folhas representam grupos unitários, e a cada passo, o algoritmo vai aglomerando os grupos mais próximos, formando novos grupos, até que no último passo reste apenas um grupo contendo todos os elementos (nó raiz). Já na abordagem divisiva, a construção da árvore se dá de cima para baixo (*top-down*), ou seja, da raiz até as folhas. O algoritmo começa com todos os elementos em um único grupo (nó raiz) e a cada passo o algoritmo vai dividindo os grupos maiores em grupos menores até que restem apenas grupos com um único elemento (nós folhas). Nas duas abordagens, vê-se que todos os casos são cobertos pelas técnicas de agrupamentos hierárquicos, ou seja, ao caminhar da raiz (1 único grupo com todos os elementos) até as folhas (n grupos unários em que n representa o número de elementos), o algoritmo passa pelos nós internos que representam casos intermediários: 2 grupos, 3 grupos, 4 grupos, etc.

O algoritmo adotado, UPGMA, é um caso de algoritmo aglomerativo. Logo, ele inicia com n grupos unários e vai aglomerando dois a dois com base numa medida de similaridade. Dessa forma, os dois grupos mais similares entre si são aglomerados em um único grupo. Supondo dois grupos R e Q, a similaridade entre os mesmos é definida como a média aritmética

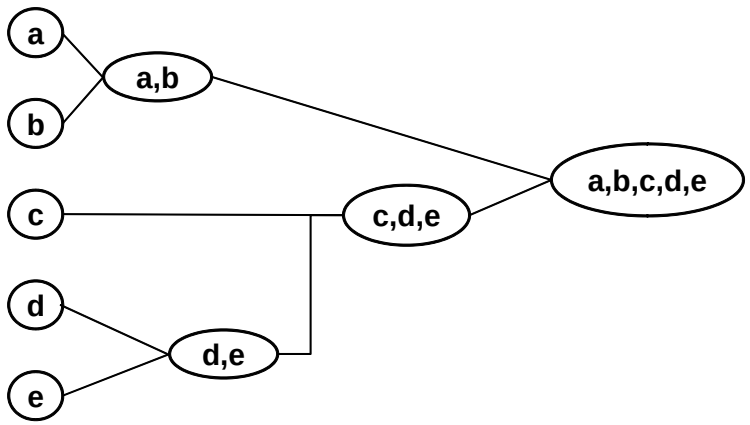
de todas as similaridades d(i, j) em que i representa um elemento do grupo R e j representa um elemento do grupo Q (a métrica mais usada é a distância Euclidiana). Em outras palavras, a medida de similaridade entre R e Q é a média aritmética de todas as distâncias entre todos os elementos do grupo R em relação ao grupo Q (44).

A entrada do algoritmo UPGMA será a matriz de distâncias entre classes encontrada pelo método *crosstalk table*. Considerando a Tabela 4.1 a matriz de distâncias retornada pelo método *crosstalk table*, após aplicar o algoritmo UPGMA sobre essa tabela, teríamos como grupos finais os nós da árvore representada pela Figura 4.2.

Tabela 4.1: Exemplo de matriz de distâncias entre classes formada através da técnica CT.

	a	**b**	**c**	**d**	**e**
a	0	2.0	6.0	10.0	9.0
b	2.0	0	5.0	9.0	8.0
c	6.0	5.0	0	4.0	5.0
d	10.0	9.0	4.0	0	3.0
e	9.0	8.0	5.0	3.0	0

Figura 4.2: Exemplo de formação dos grupos através do UPGMA.



Fonte: o autor

4.1.3 Decomposição por Agrupamentos de Padrões Semelhantes

Nesse tipo de decomposição, geralmente são utilizados métodos não-supervisionados. Estes agrupam padrões semelhantes entre si com base em alguma medida de similaridade, desconsiderando informações acerca das classes. Os grupos de padrões formados através desses métodos representam, no contexto de redes neurais modulares, os módulos a serem tratados por um determinado especialista. Como exemplos clássicos de algoritmos de agrupamento, serão

descritos a seguir os mapas auto-organizáveis, também chamados de mapas de Kohonen, e o algoritmo k-médias.

4.1.3.1 Mapas Auto-Organizáveis

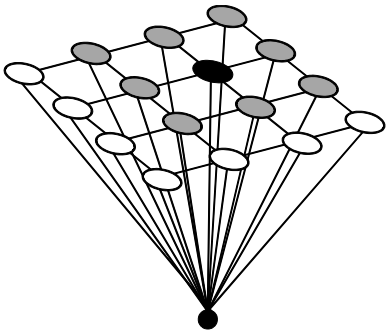
Os mapas auto-organizáveis, também chamados de redes SOM (*Self-organizing Maps*), foram propostos originalmente por Teuvo Kohonen em 1982 (5). Com forte inspiração biológica, as redes neurais do tipo SOM buscam emular o comportamento presente no cérebro humano. Verificou-se que o cérebro está organizado em várias regiões funcionais, de forma que diferentes estímulos sensoriais são representados por mapas computacionais ordenados topologicamente (9). Em outras palavras, há grupos de neurônios com funções similares que respondem mais fortemente a determinados estímulos do que outras regiões. Por exemplo, estímulos visuais e acústicos são mapeados para áreas diferentes do córtex cerebral onde há grupos de neurônios especializados em tais sinais.

Baseando-se nessa ideia, as redes SOM são formadas por uma grade unidimensional ou bidimensional (mais comum) em que estão dispostos os neurônios da camada de saída, que por sua vez estão ligados aos neurônios da camada entrada através de conexões sinápticas. Através de um aprendizado competitivo, os neurônios da camada de saída competem entre si para responderem a determinados estímulos da camada de entrada, sendo que apenas um neurônio sai vencedor numa estratégia chamada *Winner Takes All* ou o vencedor leva tudo. A competição entre os neurônios da camada de saída acontece através de conexões inibitórias laterais. Dessa forma, o neurônio vencedor juntamente com seus vizinhos diretos (ligados através de conexões laterais) são excitados de forma a aumentar a semelhança desses nodos em relação ao padrão de entrada, formando assim uma mapa topológico no qual nodos mais próximos respondem de forma semelhante a padrões de entrada semelhantes. O neurônio vencedor será aquele que estiver mais próximo do padrão de entrada, ou seja, aquele que lhe for mais similar. Para calcular o grau de semelhança entre os neurônios da camada de saída e um determinado padrão de entrada, usam-se medidas de similaridade em que a mais comumente utilizada é a distância Euclidiana. Após a definição do neurônio vencedor, ele e os seus vizinhos terão seus pesos (valores reais atribuídos às conexões sinápticas) atualizados em direção ao padrão de entrada com diferentes graus de intensidade. A Figura 4.3 ilustra uma típica topologia das redes SOM.

Na Figura 4.3 (pág. 41), o círculo preto na camada de saída representa o neurônio vencedor e a gradação de cores representa a intensidade da atualização dos pesos do neurônio vencedor e de seus vizinhos. Os neurônios mais distantes do neurônio vencedor são atualizados em menor intensidade. O passo-a-passo do funcionamento dos mapas de Kohonen pode ser sumarizado em três etapas:

1. **Competição**: Para cada padrão de entrada, os neurônios da camada de saída calculam suas distâncias (distância Euclidiana ou qualquer outra medida de similaridade) em relação ao estímulo de entrada. O neurônio da camada de saída que possuir o menor valor, ou seja, que for mais similar ao padrão de entrada, será o neurônio vencedor.

Figura 4.3: Topologia de uma rede SOM.



Fonte: o autor

2. **Cooperação**: O neurônio vencedor determina sua vizinhança topológica, ou seja, os neurônios que terão seus pesos atualizados mais fortemente em relação a esse estímulo de entrada, formando assim uma região de cooperação estreita.

3. **Adaptação Sináptica**: O neurônio vencedor e sua vizinhança atualizam seus pesos para responderem mais fortemente ao tipo de entrada que os estimulou.

Esses passos são repetidos até que seja atingido o critério de parada. Exemplos de critérios de parada podem ser o número de iterações máximas permitidas ou a pouca modificação do mapa topológico (neurônios vencedores e suas regiões de influência). O pseudo-código das redes do tipo SOM pode ser visto no algoritmo 4.2.

Algoritmo 4.2 Pseudo-código Algoritmo SOM

1: Inicializar pesos e parâmetros.
2: Apresentar padrão de entrada à rede.
3: Definir nodo vencedor.
4: Atualizar os pesos deste nodo e de seus vizinhos.
5: Reduzir fator de aprendizagem linearmente.
6: Reduzir raio da vizinhança linearmente.
7: Repetir os passos 2 a 6 até o critério de parada ser atingido.

4.1.3.2 k-Médias

Um dos mais conhecidos algoritmos de agrupamento, o k-médias é um algoritmo não-supervisionado que agrupa os padrões com base em suas semelhanças em relação aos centroides (protótipos). Inicialmente, são definidos k centroides de forma aleatória. Em seguida, é calculada a distância de cada padrão do conjunto de treinamento em relação aos centroides. O padrão de treinamento pertencerá ao centroide mais próximo (de menor distância). Após atribuir cada padrão de treinamento a um determinado grupo, as posições dos centroides são atualizadas e definidas como a média dos padrões (centro de massa) que compõem seus respectivos grupos (42). O algoritmo tem fim quando os centroides não apresentam mais modificações significativas. Seu passo-a-passo é definido no Algoritmo 4.3.

Algoritmo 4.3 Pseudo-código Algoritmo k-Médias

1: Definir aleatoriamente K centroides no espaço de padrões a ser agrupado.
2: Calcular a distância de cada padrão de treinamento em relação aos centroides e associá-lo ao centroide mais próximo.
3: Após associar todo padrão de treinamento a um determinado grupo, atualizar as posições dos centroides
4: Repetir os passos 2 e 3 até que os centroides não se movam mais.

4.2 Treinamento dos Módulos

Após terminada a fase de decomposição de tarefas, os módulos formados, seja através de agrupamento de classes ou agrupamento de padrões semelhantes, são treinados. Geralmente, o treinamento é feito em paralelo, já que os módulos representam, na maioria das vezes, tarefas independentes. Realizar o treinamento em paralelo permite uma rápida convergência, diminuindo o tempo de resposta do sistema (7). O treinamento também pode acontecer de forma sequencial dependendo da arquitetura da RNM em questão. Nesses casos, a resposta de um módulo é necessária ao módulo seguinte, visto que os módulos, na maioria dos casos, estão dispostos numa arquitetura em camadas (disposição sequencial) (45) (46) (47). A seguir, serão descritas brevemente diferentes formas de treinamento dos módulos de RNMs encontradas na literatura.

No trabalho realizado por Iwata et al. (48), a rede neural modular cresce automaticamente de acordo com a apresentação dos padrões de treinamento. A RNM faz uso do algoritmo LVQ (*Learning Vector Quantization*), um algoritmo de agrupamento supervisionado (9), para determinar a necessidade ou não da criação de novos módulos, baseando-se na similaridade entre as classes presentes em algum módulo da arquitetura. Caso essa medida de similaridade seja menor que um determinado limiar, novos módulos são adicionados à rede. Uma alternativa levantada no trabalho também seria a adição de novos neurônios escondidos a um módulo já existente em vez de criar novas redes (novos módulos). Como a criação dos módulos acontece durante a etapa de treinamento, percebe-se que a decomposição de tarefas acontece simultaneamente à etapa de treinamento.

No trabalho realizado por Nishikawa (49), a decomposição de tarefas e o treinamento dos módulos são realizados simultaneamente, como no modelo anterior. Para isso, uma rede neural SOM atribui cada novo exemplo de treinamento a um dos módulos resultantes do agrupamento. Depois de atribuir os padrões de treinamento aos módulos, estes são treinados com os padrões que os compõem. Essa técnica sofre quando trata problemas cujas fronteiras de decisão são complexas por causa da superposição de classes, uma vez que a rede SOM pode atribuir vários padrões pertencentes a uma mesma classe a módulos distintos.

4.3 Tomada de Decisão

Após a etapa de treinamento dos módulos, tem início a etapa de tomada de decisão. É nessa etapa que é definida a maneira como será fornecida a resposta final do sistema. A tomada de decisão está baseada na relação existente entre os módulos que compõem a RNM. Há basicamente três casos de relacionamento: módulos desassociados, módulos competitivos e módulos cooperativos (7).

No caso de módulos desassociados, a decisão é feita baseando-se no maior valor de ativação absoluto de todos os módulos (50) (51) (52). Nesse esquema de tomada de decisão, cada módulo não tem informação sobre os outros, não havendo, portanto, nenhum tipo de inibição entre eles (como acontece no caso de módulos competitivos) caso algum módulo produza respostas não satisfatórias para um determinado tipo de entrada.

Já no caso de módulos competitivos, haverá uma competição entre os módulos para definir quem fornecerá a resposta final para um determinado estímulo de entrada. A competição, em alguns casos, é conseguida através do uso de redes neurais não-supervisionadas (52) (48). A inibição nos módulos competitivos tipicamente se dá pelo uso de redes neurais que intrinsecamente apresentam em suas estruturas conexões inibitórias para penalizar nodos (módulos na arquitetura modular) que forneçam respostas não satisfatórias para uma determinada entrada, como são os casos das redes SOM, ART e LVQ (7). As redes neurais não-supervisionadas utilizadas pelos módulos competitivos tipicamente contêm duas camadas de neurônios, como por exemplo, as redes SOM e ART (7). Uma é a camada de entrada, que recebe os vetores de características, e a outra é a camada de saída, que contém uma certa quantidade de nodos (equivalentes aos módulos) que irão competir entre si (7). Quando o padrão de entrada é apresentado, o neurônio da camada de saída que possuir o conjunto de pesos mais próximo ao padrão de entrada (baseando-se em uma métrica de distância) será o neurônio vencedor. Dessa forma, o módulo correspondente a esse neurônio vencedor será o módulo que tomará a decisão final.

A desvantagem dessa abordagem é a alta sensibilidade de redes não-supervisionadas ao tratar espaços de características complexos (49) (48). Tais redes categorizam dados que estão próximos o bastante, de acordo com uma métrica de distância, em uma mesma categoria. Essa estratégia não é adequada para classificar padrões complexos cujas distâncias entre padrões de classes diferentes ou até mesmo padrões pertencentes a uma mesma classe são muito variáveis.

Por fim, nos módulos cooperativos, todas as saídas produzidas pelos diferentes módulos são combinadas em vez de se usar apenas a resposta fornecida por um único módulo (abordagens competitiva e desassociada). O grau de dependência dos módulos difere de uma técnica para outra, mas no trabalho proposto por Rogova (53), sugere-se que a combinação dos resultados de vários classificadores independentes é melhor que a combinação de classificadores dependentes, mesmo que estes apresentem performances individuais melhores.

Uma maneira de cooperação encontrada na literatura é a adição de um *bit* extra à saída de cada módulo (51). Este *bit* aciona um alarme caso um módulo fique responsável por tratar um

padrão de entrada cuja classe não pertence ao grupo de classes tratado por ele. Dessa forma, a resposta para esse padrão seria fornecida por um outro módulo. Nesse caso, as classes tratadas pelos diferentes módulos são disjuntas, ou seja, não pode haver uma mesma classe presente em módulos distintos.

Na proposta de Lincoln e Skrzypek (54), múltiplos módulos idênticos (redes neurais artificiais *feedforward* treinadas com *backpropagation*) são treinados em paralelo e a resposta final do sistema é definida como a média das respostas produzidas por cada módulo. Isso é feito na tentativa de aumentar a tolerância a falhas, visto que determinado módulo pode fornecer uma resposta não satisfatória. Essa estratégia, chamada de comitês de redes neurais, primeiramente proposta por Hansen e Salamon (55) em 1990, tem como desvantagem seu alto custo computacional e sua baixa velocidade de treinamento. Esse tipo de treinamento em que há o agrupamento de módulos, pesos ou respostas é chamado de treinamento cooperativo. Desde sua proposição, muitos estudos têm sido realizados na tentativa de otimizar a combinação das respostas produzidas pelos módulos do comitê (56) (57).

Outra forma de cooperação entre os módulos é o voto majoritário. Nesse tipo de abordagem, todas as saídas produzidas pelos diferentes módulos são consideradas e a resposta final produzida pelo sistema será a resposta (classe) que apareceu mais vezes dentre todos os votos (consenso). O objetivo nesse tipo de esquema é atingir uma decisão final representativa de acordo com a diversidade de respostas produzidas pelos diferentes módulos. O esquema de voto majoritário é usado para cooperação entre múltiplos módulos treinados sobre o mesmo conjunto de dados (58) (59).

4.4 Exemplos de Arquiteturas de Redes Neurais Modulares

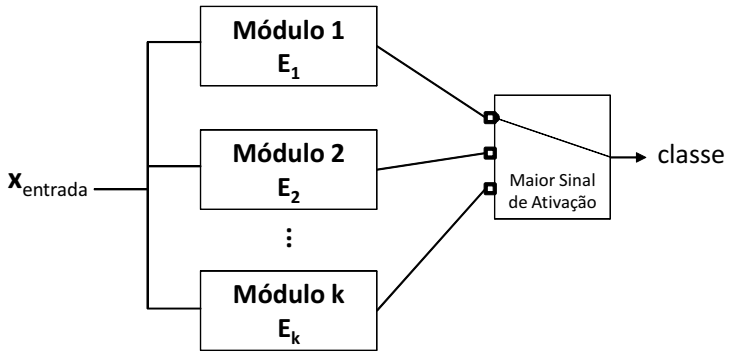
Como visto anteriormente, uma arquitetura de RNM compreende praticamente três etapas: decomposição de tarefas, uma maneira de treinar os módulos e alguma estratégia para tomada de decisão. Esta seção exemplificará algumas arquiteturas encontradas na literatura.

4.4.1 *Decoupled Modules*

Um exemplo de arquitetura de RNM é o método *Decoupled Modules* (módulos desassociados) cujos módulos são formados por agrupamento de padrões semelhantes (7). Tal arquitetura é composta de duas etapas, sendo uma não-supervisionada e outra supervisionada. Na etapa não-supervisionada, os padrões de treinamento são agrupados através de uma rede neural não-supervisionada chamada ART (*Adaptive Resonance Theory*), poderia ser utilizada também uma rede neural do tipo SOM. Após serem formados os grupos, cada um deles é atribuído a um módulo diferente. Na segunda etapa, redes neurais supervisionadas são treinadas com os padrões presentes em cada módulo. Após terminado o treinamento, a resposta final do sistema para um dado padrão de teste será fornecida pelo módulo que apresentar o maior valor de ativação

(ver Figura 4.4). Uma desvantagem dessa arquitetura é que cada módulo não tem nenhuma informação sobre os outros e, portanto, não há nenhum tipo de inibição para reações equivocadas dos módulos, isto é, um módulo classificando um padrão de uma classe com a qual ele não foi treinado (7).

Figura 4.4: Arquitetura *Decoupled Modules*.



Fonte: o autor

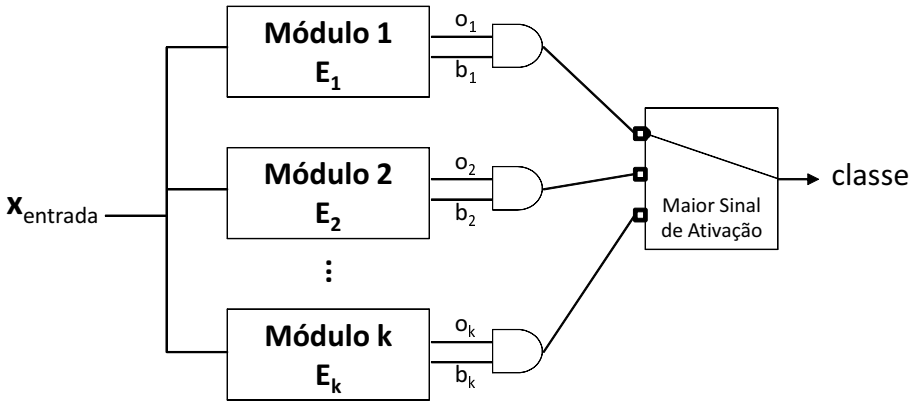
4.4.2 *Other-output*

Também formada por duas etapas, uma não-supervisionada utilizada para a formação dos grupos (módulos) e outra supervisionada para treinamento dos mesmos, a arquitetura *Other-output* (ver Figura 4.5, pág. 46) apresenta uma diferença em relação ao modelo *Decoupled Modules*: a adição de um *bit* extra à saída de cada módulo. Tais *bits*, representados na Figura 4.5 por (b_1, b_2, ..., b_k), fornecem um alarme caso um módulo fique responsável pela classificação de um padrão cuja classe não pertença ao conjunto de classes tratadas pelo mesmo. Caso isso ocorra, a resposta do módulo, representada por (o_1, o_2, ..., o_k), será anulada pelo *bit* extra que será zero, sendo essa a principal vantagem, por exemplo, em relação às redes *Decoupled Modules* em que não há a prevenção de um padrão ser tratado por um módulo cujas classes não englobam a classe do padrão de entrada. Para que cada módulo reconheça os padrões de suas respectivas classes, é realizado um treinamento negativo no qual cada módulo é treinado com todos os exemplos que não os seus (51), ou seja, cujas classes não estejam presentes nos módulos. Esta forma de treinamento pode ser complexa se houver grande diversidade nessas classes (50).

4.4.3 Modelo de Múltiplos Especialistas

No modelo de múltiplos especialistas, o problema original é decomposto em subproblemas em que cada subproblema é tratado por um módulo (especialista). Cada especialista é formado por redes neurais simples, que se diferenciam entre si apenas pela inicialização dos pesos. Nesse modelo de arquitetura há uma unidade extra chamada *gating*, que é responsável por atribuir um peso a cada resposta fornecida pelos especialistas (ver Figura 4.6, pág. 47). Tal

Figura 4.5: Arquitetura *Other-output*.



Fonte: o autor

unidade pode ser uma rede neural (60) ou uma heurística qualquer responsável por calcular os pesos. A unidade *gating* atribui pesos a cada saída dos especialistas, alterando assim o impacto de cada resposta individual na resposta final do sistema. Em Jacobs et al. (60), o peso de cada especialista é calculado com base no seu erro cometido durante a classificação, ou seja, diferença entre a resposta esperada e a resposta fornecida. Com base nos erros e no padrão de entrada, a unidade *gating* é treinada para calibrar os pesos de cada módulo a fim de obter uma resposta mais satisfatória dos mesmos quando submetidos a estímulos de entrada (60).

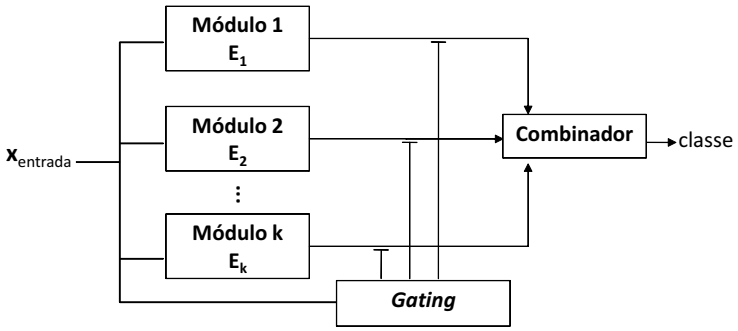
A vantagem dessa arquitetura é a presença dessa unidade especial que atribui diferentes pesos aos módulos especialistas baseando-se na capacidade de determinado módulo classificar corretamente um determinado padrão de entrada. Ao mesmo tempo que tal unidade mostra-se como vantagem, a escolha da maneira como a unidade *gating* calculará tais pesos será uma fator crucial para o desempenho da arquitetura como um todo, uma vez que os pesos sinalizam qual módulo é o mais apropriado para tratar determinado padrão de entrada.

O combinador não representa outro módulo (rede neural), trata apenas da maneira como a resposta de todo o sistema será determinada, isto é, se será uma combinação linear das respostas produzidas pelos diferentes especialistas, a maior resposta produzida pelos módulos ou a resposta produzida pelo módulo que tem maior peso associado (calculado pela unidade *gating*) como é observado no trabalho de Jacobs et al. (60).

4.4.4 *Output Paralellism*

Guan et al. (61) propuseram uma nova arquitetura de RNM chamada *Output Paralellism* (OP). Nessa arquitetura, o conjunto de dados original é dividido em vários subconjuntos menores com classes distintas. Os autores não especificaram qual técnica de decomposição de tarefas foi utilizada, ficando a cargo do desenvolvedor determinar um método de sua preferência. Após feita a divisão dos dados originais em subconjuntos menores, os módulos são treinados com todos os seus padrões representantes, ou seja, todos os padrões pertencentes às classes verificadas nos

Figura 4.6: Arquitetura *Modelo de Múltiplos Especialistas*.

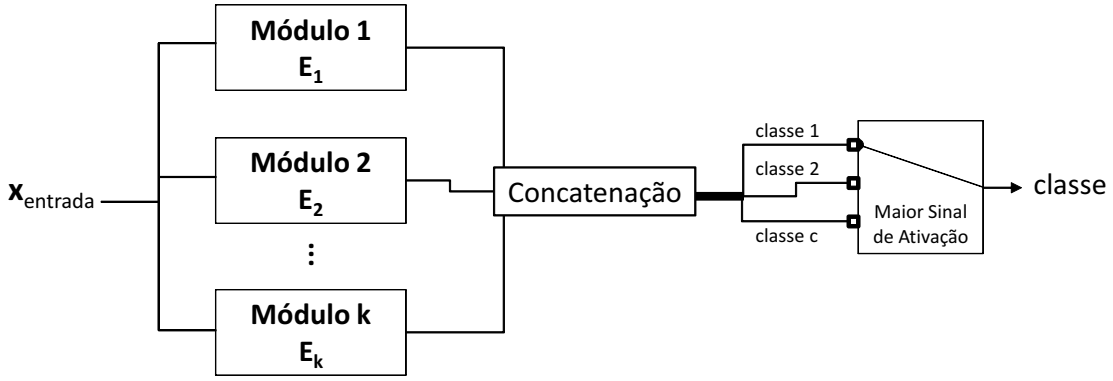


Fonte: o autor

módulos. Essa arquitetura faz uso de algoritmos de treinamento construtivos, logo, as redes neurais simples de cada módulo crescem através da adição de neurônios escondidos e pesos para melhor se adequarem ao subconjunto de classes que elas representam.

Após treinados os módulos, inicia-se a tomada de decisão. Ao ser apresentado um padrão de teste, todos os módulos o recebem integralmente como entrada e produzem como resposta uma porção final do vetor de saída (o número de classes em cada módulo é equivalente ao número de neurônios na camada de saída de suas redes simples). As porções de respostas produzidas pelos diferentes módulos da arquitetura são então concatenadas em uma resposta única. A resposta final do sistema será a classe do neurônio que produzir o maior sinal de ativação (regra do vencedor leva tudo). Como exemplo, considere um problema com 11 classes cuja decomposição de tarefas produziu os seguintes módulos: {{6,3,2,11,9},{10,1,8},{4,7},{5}}. Ao receber um padrão de teste, ele será processado pelos quatro módulos e as suas respectivas saídas serão agrupadas em uma resposta única, no caso um vetor com 11 posições. A classe final do padrão de teste será então a classe representada por um dos 11 neurônios que produzir o maior valor de resposta. A arquitetura pode ser vista na Figura 4.7.

Figura 4.7: Arquitetura *Output Paralellism*.



Fonte: o autor

As vantagens dessa abordagem são: (i) a criação de sub-redes menores que acarretam uma redução da interferência interna entre as camadas de neurônios escondidos durante a atualização

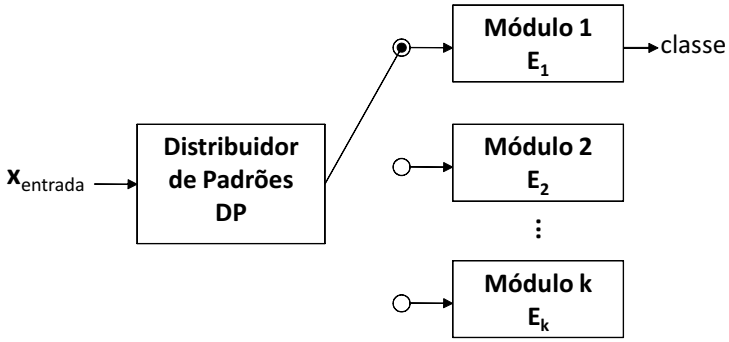
de pesos com relação à abordagem não modular, (ii) a redução do custo computacional no treinamento das sub-redes, já que as redes são mais compactas, (iii) a melhora do desempenho do sistema como um todo e (iv) o uso de algoritmos de treinamento construtivos, não sendo necessário o desenvolvedor estabelecer uma topologia de rede fixa. Assim, a rede se adequa ao conjunto de dados de sua responsabilidade (61). Uma desvantagem está relacionada ao tempo de treinamento. Como esta arquitetura trabalha com algoritmos construtivos, se o tempo de treinamento for longo, haverá um custo computacional demasiado com possível *overfitting* e má generalização (61).

4.4.5 *Pattern Distributor*

Tendo como base a arquitetura *Output Paralellism* (OP), Guan et al. (40) propuseram uma nova arquiteruta de RNM chamada *Pattern Distributor* (PD). Tal qual o modelo OP, a arquitetura PD agrupa os dados por classes, ou seja, os módulos são formados por subconjuntos de classes disjuntas, porém, a arquitetura PD não é restrita a métodos de agrupamento por classes, havendo também a possibilidade de utilizar métodos de decomposição nos quais o agrupamento é feito com base em padrões semelhantes. Os autores usam como técnicas de decomposição de tarefas os métodos *crosstalk table* e algoritmos genéticos. A diferença em relação ao modelo OP é a adição de forma hierárquica de um módulo chamado *pattern distributor* (distribuidor de padrões) (ver Figura 4.8, pág. 49) . A função desse módulo é entregar cada padrão de entrada a apenas um módulo específico. Assim, em vez de levar em consideração as respostas produzidas por todos os módulos (arquitetura OP), a resposta final na arquitetura PD é dada por apenas um módulo. A introdução desse módulo originou um novo método de decomposição de tarefas chamado de decomposição de tarefas com *pattern distributor* (40).

Para saber qual módulo selecionar, o módulo PD é treinado com toda a base de dados com o intuito de classificar os padrões que recebe nas classes possíveis (as classes do módulo PD passam a ser o rótulo dos módulos). Os outros módulos da arquitetura são treinados apenas com os padrões de suas respectivas classes. Com a introdução desse módulo hierárquico, os autores reportaram melhora na performance de todo o sistema quando comparado à arquitetura OP e uma melhora na capacidade de generalização da rede como um todo quando comparado com métodos comuns de decomposição de tarefas (40). Outra vantagem dessa arquitetura é a possibilidade de explorar as mais variadas técnicas para formação dos módulos seja através de agrupamento por padrões semelhantes ou agrupamento por classes semelhantes. A ideia por trás do módulo distribuidor de padrões, a independência da arquitetura quanto à maneira de formação dos módulos bem como os resultados e melhoras relatados pelos autores motivaram a utilização desta arquitetura no presente trabalho.

Figura 4.8: Arquitetura *Pattern Distributor*.



Fonte: o autor

4.5 Considerações Finais

Neste capítulo foram cobertos os principais aspectos de uma rede neural modular. Passou-se pela sua inspiração biológica e o objetivo de se aproximar ainda mais das redes neurais naturais presentes em nosso cérebro via modularização. Foram abordadas as três etapas principais à maioria das arquiteturas de redes neurais modulares que são: decomposição de tarefas, treinamento dos módulos e tomada de decisão. Foram descritas as duas técnicas de decomposição de tarefas utilizadas neste trabalho que são os algoritmos genéticos e a técnica *crosstalk table* combinada com o algoritmo UPGMA. Por fim, alguns exemplos de arquiteturas de RNMs, dentre as quais a arquitetura utilizada neste trabalho, tiveram seus funcionamentos e características descritos sublinhando algumas vantagens e desvantagens de tais arquiteturas.

5
Métodos Propostos

Como objetivo inicial do presente trabalho, a investigação do uso de métodos de seleção de características aplicados aos módulos de redes neurais modulares resultou nas seguintes contribuições, que foram publicadas em duas grandes conferências da área de inteligência artificial (ver seção 1.2, pág. 15): (i) um método híbrido que realiza seleção e ponderação de características simultaneamente baseado no PSO binário (BPSO) e (ii) uma nova arquitetura de rede neural modular que seleciona automaticamente diferentes subconjuntos de características por módulo. Nas seções seguintes, tais contribuições terão os seus funcionamentos detalhados.

5.1 Método Híbrido de Seleção e Ponderação de Características baseado em BPSO

Tahir et al. (20) propuseram um método híbrido de seleção e ponderação de características baseado na técnica de otimização global *Tabu Search* (ver seção 3.1, pág. 24). Nesse método, o algoritmo *Tabu Search* é hibridizado com o conhecido algoritmo baseado em instâncias *k-Nearest Neighbor* (k-NN) (62). A função do algoritmo de otimização TS é realizar a busca pelo subconjunto de características ponderadas (solução candidata) enquanto a função do classificador k-NN é avaliar cada uma dessas soluções candidatas, fornecendo um *feedback* para a condução da busca. Quando finalizada a busca, o melhor subconjunto de características ponderadas é retornado e utilizado pelo classificador k-NN sobre os conjuntos de treinamento e teste para aferição do desempenho do sistema. O k-NN é hibridizado com o TS para a realização da busca e também utilizado como classificador final. Esse método híbrido de seleção e ponderação de características segue a abordagem *wrapper* (ver seção 2.1.3, pág. 21).

Baseando-se nesse trabalho elaborado por Tahir et al. (20), Barros e Cavalcanti (14) propuseram um algoritmo híbrido para seleção e ponderação de características que usa o *Tabu Search* ou o *Simulated Annealing* (ver seção 3.2, pág. 25) como algoritmos de busca. Ambos são também hibridizados com o classificador k-NN e em vez de utilizar a distância Euclidiana tradicional para o cálculo dos vizinhos mais próximos, os autores fizeram uso da distância Euclidiana adaptativa (63). As motivações por trás desse trabalho foram (i) a redução do tempo computacional durante a busca pelas melhores soluções, visto que o *Simulated Annealing* gera e avalia apenas um vizinho para cada melhor solução encontrada, enquanto o *Tabu Search* gera uma vizinhança formada por um número n de potenciais soluções e cada uma dessas n soluções

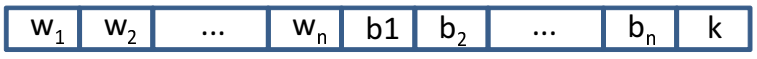
são avaliadas e comparadas com a melhor solução encontrada até o momento e (ii) melhorar os resultados obtidos pelo k-NN, através do uso da distância Euclidiana adaptativa, em casos em que haja superposição de classes.

O método híbrido de seleção e ponderação de características proposto nesse trabalho também segue a metodologia proposta por Tahir et al. (20) e baseia-se no uso do PSO binário como algoritmo de busca. O método proposto segue a mesma metodologia definida por Tahir et al. (20) no que se refere à maneira como a solução candidata é codificada (ver seção 5.1.1), à maneira como a população inicial é gerada (ver seção 5.1.2, pág. 52) e à maneira como se dá o processo de busca pela melhor solução (ver seção 5.1.3, pág. 52). A única diferença entre o método proposto no presente trabalho e o método proposto por Tahir et al. (20) está no algoritmo de otimização hibridizado com o classificador k-NN. Enquanto os autores do trabalho original fizeram uso do algoritmo *Tabu Search*, o presente trabalho optou por uma versão melhorada do PSO binário, sendo essa a primeira vez que o PSO binário foi combinado com o k-NN com o propósito de seleção e ponderação de características. A motivação para o uso do PSO binário é o compartilhamento de informações sobre ótimos locais e mínimo ou máximo global (dependendo da função objetivo), visitados até o momento, por todo o enxame de partículas. Essa cooperação mútua entre as partículas geralmente resulta em melhores soluções quando comparada a métodos de busca em que não há essa simbiose, caso dos algoritmos TS e SA. Outra motivação é o fato do PSO binário ser feito para trabalhar com problemas discretos, como é o caso do problema de seleção de características.

5.1.1 Codificação de uma solução candidata

A abordagem proposta utiliza o algoritmo BPSO combinado com o k-NN na busca pelo melhor subconjunto de características para um dado problema. A técnica realiza seleção e ponderação de características simultaneamente. Dessa forma, cada partícula do enxame é codificada de acordo com vetor mostrado na Figura 5.1.

Figura 5.1: Codificação de uma partícula utilizada pelo algoritmo BPSO ponderado.



Fonte: o autor

A Figura 5.1 mostra que cada partícula é composta por três partes. A primeira parte (w_1, w_2, ..., w_n) é formada por valores reais (pesos) que são associados a cada característica.

A segunda parte (b_1, b_2, ..., b_n) consiste de valores {0,1} para cada característica. Esses *bits* representam a ausência ou presença da característica no subconjunto a ser retornado. Por fim, a terceira parte representa o valor k de vizinhos mais próximos a ser usado pelo algoritmo k-NN. O valor n representa a quantidade de características presentes na dase de dados.

5.1.2 População inicial

Foi adotado o mesmo tamanho de população definido por Tahir et al. (20), que tem um total de $M\ x\ N + P$ partículas onde M e N têm seus valores definidos em 10 e 2, respectivamente. O valor do parâmetro P tem seu valor definido pela Equação 5.1 em que F representa o número de características de uma dada base de dados.

$$P = teto(\sqrt{F}) \tag{5.1}$$

Nessa abordagem, $M\ x\ N$ soluções candidatas diferentes são geradas a partir da parte real que compõe a codificação da solução (ver Figura 5.1), ao atribuir M valores reais randômicos a N características diferentes. Por sua vez, P diferentes soluções candidatas são geradas, randomicamente, ao excluir ou incluir uma característica na parte binária que faz parte do sistema de codificação de uma solução candidata. Na seção 6.2.1 (pág. 60), é explicado em detalhes, através de um exemplo, o mecanismo de gerar, através desses três parâmetros, a população inicial utilizada pelos algoritmos de busca.

A população inicial tem seus pesos, valores reais, gerados aleatoriamente no intervalo [0,1] e a parte binária tem seus valores definidos aleatoriamente em 0 ou 1. O valor k, presente na codificação da solução, tem seu valor definido aleatoriamente entre 1 e 7. Não houve algum critério específico para definir os valores de k no algoritmo k-NN no intervalo [1,7]. Tahir et al. (20) adotaram arbitrariamente esses valores, mas nada impede que o intervalo seja expandido ou diminuído. No presente trabalho, foi adotado esse mesmo intervalo como explicitado anteriormente. O BPSO original trabalha apenas com a parte binária da solução candidata e com o valor k utilizado pelo classificador k-NN. A fim de tratar a parte da solução candidata que representa os pesos, foi necessário adicionar uma instrução extra, responsável pela atualização dos mesmos, ao algoritmo original do BPSO. A atualização de pesos é feita após a atualização das posições e velocidades de todo o enxame (linha 27 do Algoritmo 3.4).

5.1.3 Etapa de busca

O objetivo da etapa de busca é explorar o espaço que contém as possíveis soluções para o problema abordado à procura da melhor solução para o mesmo. Para isso, cada partícula da população (solução candidata) é avaliada utilizando o classificador k-NN e seu custo é representado pelo número de padrões classificados erroneamente. Tal custo, atribuído a cada solução candidata pelo classificador k-NN, representa a qualidade da solução para o problema a ser tratado, de forma que quanto menor o custo da solução, melhor é a solução candidata. Dessa forma, a função objetivo é minimizar a função custo (Equação 5.2).

$$Custo = \sum_{i=1}^{k} C_k \tag{5.2}$$

A Equação 5.2 é utilizada para avaliar a aptidão de cada solução candidata no Algoritmo 3.4. Nela, k representa o número de classes do problema e C_k representa o número de padrões classificados equivocadamente por classe. Os vizinhos utilizados pelo k-NN são calculados utilizando a distância Euclidiana definida pela Equação 5.3 em que $\mathbf{x}$ e $\mathbf{y}$ são vetores de entrada e d é o número de características.

$$D(x,y) = \sum_{p=1}^{d} (\mathbf{x}_p - \mathbf{y}_p)^2 \tag{5.3}$$

Para calcular os vizinhos, a parte binária dos padrões de treinamento bem como do padrão que se está querendo classificar são multiplicados pelos seus respectivos pesos. O enxame de partículas vai iterativamente atualizando as velocidades e posições das partículas de acordo com o $\hat{\mathbf{y}}$ (melhor posição encontrada por todo o enxame) e com os $\mathbf{y}_i$ (melhores posições atingidas por cada partícula i). Quando atingido o número máximo de iterações, o valor $\hat{\mathbf{y}}$ é retornado como a solução do problema.

Após concluída a busca, o valor $\hat{\mathbf{y}}$ é utilizado como uma máscara para gerar o novo subconjunto de características que é avaliado pelo classificador k-NN.

5.2 Proposta de Nova Arquitetura de Rede Neural Modular

Redes neurais modulares tiram proveito da modularização para superar as redes neurais simples ao lidar com problemas complexos. Um conceito crucial em RNMs é a decomposição. A decomposição é conseguida através do uso de algoritmos de decomposição de tarefas que dividem um problema complexo em vários subproblemas mais fáceis de serem resolvidos. Depois da decomposição, cada módulo usa o conjunto inteiro de características para treinar seu respectivo especialista. Entretanto, é esperado que diferentes módulos requeiram diferentes características para realizarem suas tarefas. Portanto, faz-se necessário escolher quais características melhor preservam o poder de discriminação entre as classes presentes em cada módulo.

Seleção de características aplicada aos módulos individuais de uma rede neural modular é um tópico pouco explorado na literatura. Os poucos trabalhos encontrados na literatura que tratam deste asssunto são descritos brevemente em seguida.

Um trabalho, proposto por Guan e Zhu (13), envolve um esquema modular para classificação baseado em algoritmos genéticos. Este esquema modular é responsável pela decomposição de tarefas em vários módulos. Como método de seleção de características, os autores propuseram uma nova técnica chamada *Relative Importance Factor* (RIF) baseada na função do discriminante linear de Fisher. A técnica RIF foi integrada ao esquema de decomposição baseado em AGs para encontrar o subconjunto de características mais adequado para cada módulo. Bons resultados foram atingidos e reportados.

Outro trabalho, proposto por Guan e Liu (8), utiliza um esquema incremental para identificar e descartar as características irrelevantes através de uma rede neural simples. A rede

neural simples neste caso aprende um atributo por vez e descarta as características que não trazem contribuição à discriminação entre classes. O discriminante linear de Fisher é utilizado para fazer um *ranking* das características. Este método criado foi utilizado em redes neurais modulares para selecionar as características mais relevantes para cada módulo. Boa performance, durante a classificação, foi reportada.

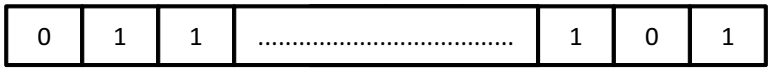
Na proposta deste trabalho de uma arquitetura de rede neural modular que seleciona automaticamente o subconjunto de características mais adequado para cada módulo, o procedimento de seleção de características é realizado pelo algoritmo *Improved Binary Particle Swarm Optimization* (IBPSO) (12) (ver seção 3.4.1 e Algoritmo 3.4).

A arquitetura de RNM proposta, de nome *Improved Binary Particle Swarm Optimization - Pattern Distributor* (IBPSO-PD), é baseada na arquitetura *Pattern Distributor* (ver Figura 4.8, pág. 49) proposta por Guan et al. (40) (ver seção 4.4.5, pág. 48). Como outras arquiteturas de redes neurais modulares, a arquitetura PD divide o problema em subproblemas em que cada subproblema é tratado por um módulo diferente. A principal diferença entre a arquitetura PD e outras arquiteturas modulares é a presença de um módulo especial chamado distribuidor. O módulo distribuidor é responsável por escolher o módulo que dará a resposta final para cada padrão de entrada. Assim, a seleção do módulo vencedor é feita antes da classificação, diferentemente de outras arquiteturas de RNMs onde a seleção do módulo vencedor é geralmente feita após a classificação de um dado padrão de entrada.

Na arquitetura proposta, os métodos de decomposição de tarefas utilizados são os mesmos *Crosstalk table* (ver seção 4.1.2, pág. 37) e algoritmos genéticos (ver seção 4.1.1, pág. 36) utilizados por Guan et al. (40). No processo evolucionário dos algoritmos genéticos, cada solução candidata (cromossomo) é avaliada por uma rede neural simples.

No problema de seleção de características, cada subconjunto de características é representado por um vetor binário onde cada posição no vetor binário indica a ausência (*bit* 0) ou presença (*bit* 1) da característica correspondente. Cada solução candidata utilizada pelo algoritmo IBPSO (ver Algoritmo 3.4) é codificada como mostra a Figura 5.2 (a quantidade de *bits* presente na solução candidata é igual à quantidade de características da base de dados).

Figura 5.2: Codificação de uma solução candidata usada pelo algoritmo IBPSO.



Fonte: o autor

Como já explicitado no capítulo sobre RNMs, as principais etapas de uma RNM são a decomposição de tarefas, treinamento dos módulos e tomada de decisão. Cada uma dessas três etapas será explicada em detalhes no contexto da arquitetura IBPSO-PD. A Figura 5.3 (ver pág. 56) apresenta o fluxo de dados durante o treinamento na arquitetura proposta. O treinamento da mesma pode ser descrito em quatro passos:

- **Decomposição de tarefas**: Inicialmente, o conjunto original de dados é dividido em módulos através de uma técnica de decomposição de tarefas. A arquitetura é flexível o suficiente para admitir diferentes métodos de decomposição. A escolha da técnica de decomposição é um fator importante, visto que uma má decomposição pode levar a um deterioramento na performance do sistema. Dado um conjunto de treinamento $\{(x_1,c_1),(x_2,c_2),...(x_n,c_n)\}$ em que x_i é um padrão e c_i é sua respectiva classe, todas as instâncias desse conjunto são usadas pela técnica de decomposição para achar o número de grupos (módulos) representado por k. Terminada a decomposição, um conjunto de grupos de dados é retornado: $G = \{G_1, G_2, ..., G_k\}$;

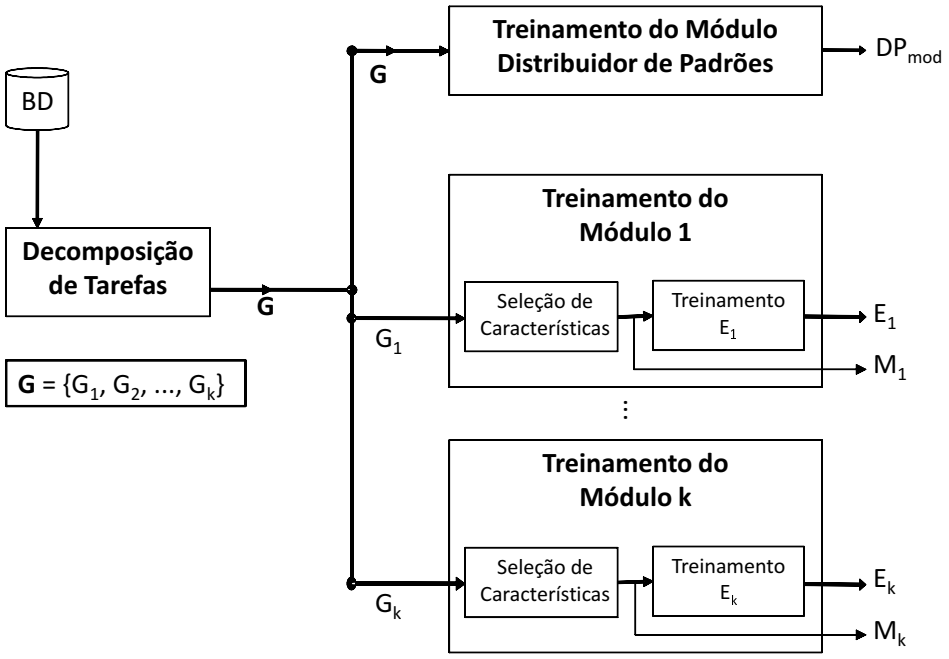
- **Treinamento do módulo DP**: Os padrões de todos os grupos são usados para treinar o módulo distribuidor (DP). Os padrões de cada grupo têm suas classes alteradas para uma nova classe que indica de que grupo cada padrão é proveniente. Assim, os padrões do G_1 têm suas classes alteradas para 1; os padrões do grupo G_2 têm suas classes alteradas para 2 e assim por diante até o grupo G_k. Após essas alterações, o módulo DP é treinado para decidir a que grupo um padrão pertence. O módulo distribuidor já treinado é representado na Figura 5.3 (pág. 56) por DP_{mod};

- **Seleção de características**: Os padrões em cada grupo são usados pelo algoritmo IBPSO com o objetivo de selecionar as características que preservam a informação discriminante exigida para a tarefa de classificação. Durante o processo iterativo, cada solução candidata é avaliada através do vizinho mais próximo (ver Algoritmo 3.4, pág. 32) que lhe atribui o valor de aptidão definido pela equação 5.2 (pág. 52). Para isso, a máscara (representada por **M** na Figura 5.3, pág. 56) de cada solução candidata é previamente aplicada a todos os padrões que compõem o módulo. Após o processo iterativo, o algoritmo IBPSO retornará um vetor binário (ver Figura 5.2, pág. 54), que representa a melhor solução encontrada pelo algoritmo, em que cada posição do vetor indicará a ausência ou presença da respectiva característica. Então, essa máscara é aplicada a todos os padrões que formam o módulo, resultando em uma redução de dimensionalidade. A seleção de características de cada módulo pode ser feita em paralelo, uma vez que os módulos são independentes;

- **Treinamento dos módulos**: Após realizar a seleção de características, os novos padrões de cada módulo são usados para treinar seus respectivos especialistas (redes neurais simples). Logo, o i-ésimo módulo utiliza apenas os padrões pertencentes ao grupo G_i com suas classes originais para treinar o especialista E_i. Se um módulo possuir apenas uma classe, não será necessário treinar um classificador, uma vez que apenas uma saída é possível. Caso um módulo tenha mais de uma classe, uma rede neural simples será treinada para classificar um padrão dentre suas classes existentes. O treinamento dos módulos é feito em paralelo, visto que as subtarefas dos módulos são independentes. Terminado o treinamento, o sistema retornará os especialistas

treinados (E_1, E_2, ..., E_k) bem como suas respectivas máscaras (M_1, M_2, ..., M_k), como mostra a Figura 5.3.

Figura 5.3: Treinamento da Arquitetura IBPSO-PD.



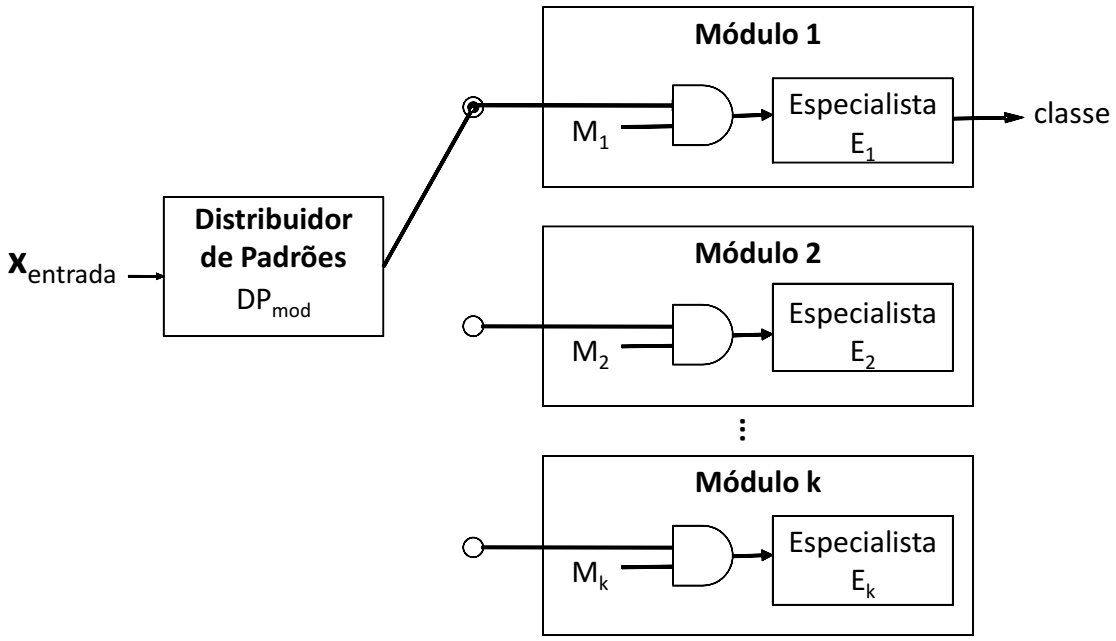
Fonte: o autor

Após realizado o treinamento da arquitetura IBPSO-PD, tem início a etapa da tomada de decisão. Cada padrão do conjunto de teste, representado por ($x_{entrada}$), será classificado pelo sistema (ver Figura 5.4, pág. 57). Primeiro, o módulo DP seleciona o módulo apropriado, que por sua vez aplica sua respectiva máscara ao padrão $x_{entrada}$, filtrando suas características. Então, o especialista do módulo fornece a classe prevista para o padrão de entrada. Como as técnicas de decomposição de tarefas utilizadas (CT e AG) retornam grupos não redundantes, isto é, padrões de uma classe particular estão presentes em apenas um módulo, se o módulo distribuidor (DP) cometer um erro, o especialista selecionado não será capaz de corrigir o erro, sendo então propagado pelo módulo. A tomada de decisão é mostrada na Figura 5.4 (ver pág. 57).

5.3 Considerações Finais

Neste capítulo foram vistas em detalhes as duas contribuições desse trabalho que foram o método híbrido de seleção e ponderação de caracterísitcas baseado no PSO binário e a arquitetura de rede neural modular que seleciona automaticamente um subconjunto de características para cada módulo da rede. No que se refere ao método híbrido, foi abordada primeiramente a técnica proposta por Tahir et al. (20) em que foi baseado o algoritmo de otimização desenvolvido

Figura 5.4: Tomada de Decisão da Arquitetura IBPSO-PD.



Fonte: o autor

no presente trabalho bem como outro método híbrido com o qual a técnica desenvolvida foi comparada, que foi o trabalho de Barros e Cavalcanti (14). Além disso, foram vistos os três principais pontos do algoritmo híbrido de otimização que são a maneira como as soluções candidatas são codificadas, a maneira como é gerada a população inicial e a fase de busca do algoritmo. Na arquitetura de rede neural modular proposta, também sublinha-se o trabalho em que foi baseada essa nova arquitetura bem como as etapas que formam o seu funcionamento que são o treinamento da rede e o processo de tomada de decisão.

6
Experimentos e Resultados

Este capítulo descreve os experimentos realizados tanto com os métodos propostos quanto com as técnicas utilizadas para fins de comparação. As bases de dados utilizadas, a metodologia dos experimentos e uma análise dos resultados obtidos são descritos.

6.1 Bases de dados

Para demonstrar a eficácia dos métodos propostos, foram realizados experimentos com várias bases tidas como *benchmark*. À exceção da base *vowel* (64), todas as outras bases de dados foram provenientes do repositório de aprendizagem de máquina UCI (65). Todas as bases utilizadas nos experimentos representam problemas de classificação.

A base de dados *diabetes* (65) trata do diagnóstico de diabetes em pacientes com herança da tribo indígena Pima no estado do Arizona. Todas as pacientes são mulheres com idade mínima de 21 anos. São averiguados oito atributos para diagnosticar se uma determinada paciente apresenta ou não diabetes (nível de glicose no sangue igual ou superior a 200mg/dl). O problema nessa base é binário, ou seja, há apenas duas classes, a classe 0 indicando ausência de diabetes e a classe 1 indicando a presença.

A base de dados *glass* (65) apresenta originalmente 10 características, mas uma delas, o atributo de identificação *Id*, foi removida, já que não é importante do ponto de vista da tarefa de classificação. Assim, cada padrão da base é descrito por um vetor de 9 características que indicam a concentração de elementos químicos no vidro e o seu índice de refração. O objetivo é classificar cada amostra entre os 6 tipos de vidros presentes na base. A motivação para esse estudo de tipo de vidros foi a investigação de crimes.

Já a base *heart* (65) utiliza 13 características como tipos de dores no peito, idade, sexo e colesterol, por exemplo, para diagnosticar a presença ou ausência de doença no coração. Cada padrão de entrada da base *ionosphere* (65) é composto por 34 atributos contínuos que são utilizados para indicar a presença ou ausência de algum tipo de estrutura na ionosfera. Os dados foram coletados no município de Goose Bay na província de Labrador no Canadá por antenas de alta-frequência com um poder de transmissão da ordem de 6.4 kilowats.

Cada padrão na base *sonar* (65) é formado por um conjunto de 60 números no intervalo [0,1], cada número representa a energia dentro de uma determinada banda de frequência, por um determinado período de tempo. O objetivo é classificar cada padrão como sendo rocha ou minério. Por sua vez, a base *vehicle* (65) trata de identificar um veículo pelos seus contornos

dentre os quatro tipos de veículos presentes na base. Os atributos dessa base fazem referência a aspectos físicos dos veículos como circularidade, retangularidade, compacidade, dentre outras características geométricas (65).

A Tabela 6.1 sumariza as principais informações das bases adotadas nos experimentos em que são comparados os três métodos híbridos de seleção e ponderação de características.

Tabela 6.1: Bases de dados utilizadas pelo método BPSO/k-NN.

Base de dados	#Padrões	#Características	#Classes
Diabetes	768	8	2
Glass	214	9	6
Heart	270	13	2
Ionosphere	351	34	2
Sonar	208	60	2
Vehicle	846	18	4

A base *balance* (65) foi gerada para modelar resultados psicológicos experimentais. Há três situações possíveis: estar desequilibrado para a direita, desequilibrado para a esquerda ou estar equilibrado. Por sua vez, a base *iris* (65) faz uso de quatro atributos (comprimento e largura da pétala e comprimento e largura da sépala) para classificar uma planta iris em três tipos: *setosa*, *versicolour* e *virginica*.

A base *LIBRAS* (65) contém 15 classes em que cada classe representa um tipo de movimento de mão na Linguagem Brasileira de Sinais. A base *liver* (65) trata amostras formadas por testes sanguíneos sensíveis a problemas no fígado. Com base nesses testes, um padrão é classificado como tendo ou não alguma desordem no fígado. A base *semeion* (65) trata do reconhecimento de dígitos manuscritos. Possui 256 características e 10 classes, que são os dígitos de 0 a 9. Por fim, a base *vowel* (64) trata do reconhecimento fonético das vogais. Cada um dos onze diferentes sons foi pronunciado 6 vezes por 15 falantes independentes totalizando um conjunto de 990 padrões.

A Tabela 6.2 resume as principais informações das bases adotadas nos experimentos com a arquitetura de rede neural modular proposta, que seleciona as características mais relevantes de cada módulo.

Tabela 6.2: Bases de dados utilizadas com a arquitetura IBPSO-PD.

Bases de dados	#Padrões	#Características	#Classes
Balance	625	4	3
Ionosphere	351	34	2
Iris	150	4	3
LIBRAS	360	91	15
Liver	345	6	2
Semeion	1593	256	10
Vehicle	846	18	4
Vowel	990	10	11

6.2 Metodologia dos experimentos

Para demonstrar a eficácia dos métodos propostos, foram realizados diversos experimentos com as bases de dados descritas na seção 6.1. Ao comparar as diferentes técnicas, teve-se o cuidado de submetê-las às mesmas condições experimentais com o intuito de obter resultados mais confiáveis. A seguir, são descritas as metodologias empregadas para cada um dos métodos propostos.

6.2.1 Metodologia empregada pelo método BPSO/k-NN

O método híbrido de seleção e ponderação de características proposto, de nome BPSO/k-NN, foi comparado com o método SA/k-NN proposto por Barros e Cavalcanti (14) e com o método original TS/k-NN proposto por Tahir et al. (20). Além disso, o método proposto foi comparado com o k-NN tradicional, ou seja, sem seleção de características. O número de k vizinhos utilizados pelo método k-NN tradicional foi definido empiricamente no intervalo [1,7], isto é, o valor k, variando de 1 a 7, que propiciou a maior taxa de acerto, foi utilizado para definir a quantidade de vizinhos a ser utilizada na predição da classe de um determinado padrão de entrada. Para definir o melhor valor de k, a base de dados foi dividida, sem interseção, em conjunto de treinamento (formado por 70% do total de instâncias) e conjunto de teste (formado por 30% do total de instâncias).

A métrica de desempenho utilizada pelos quatro métodos foi a taxa de acerto, que consiste na razão entre o número de padrões classificados de forma correta sobre o número total de padrões. Para se ter uma melhor e mais confiável estimativa da taxa de acerto obtida por cada método, foi utilizado o método de validação cruzada *k-fold* (66) com k sendo 10. Dessa forma, o conjunto original de dados foi dividido em 10 subconjuntos disjuntos e a cada iteração do 10-*fold*, um *fold* foi utilizado como conjunto de teste e os nove restantes foram utilizados para treinar os classificadores. Ao fim do 10-*fold*, têm-se dez medidas de acerto para cada classificador e o desempenho final de cada um é dado como a média dessas taxas de acertos.

O número máximo de iterações utilizado pelos métodos de busca TS/k-NN, SA/k-NN e BPSO/k-NN foi definido como 100. O tamanho **T** da lista *tabu* é definido através da Equação 6.1 em que F representa a quantidade de características presente na base de dados.

$$T = teto(\sqrt{F}) \tag{6.1}$$

O tamanho da população dos três métodos de busca é definido como $M\ x\ N + P$ em que M e N assumem os valores 10 e 2, respectivamente, e o valor P é definido pela equação 5.1 (20). A população no método BPSO/k-NN é inicializada de forma aleatória como visto na seção 5.1.2. No método SA/k-NN, Barros e Cavalcanti (14) seguiram o modelo definido por Tahir et al. (20).

A inicialização do conjunto de soluções iniciais nos métodos TS/k-NN e SA/k-NN é baseada em uma solução geradora cujos pesos, *bits* e valor k da solução candidata são todos

iguais a um (ver esquema de codificação na Figura 5.1, pág. 51). A partir dessa solução inicial geradora, M valores aleatórios (pesos) no intervalo [0,1] são adicionados a uma característica específica. Por exemplo, se M for 2, dois pesos são adicionados a uma característica específica f_x. Dessa forma, 2 novas soluções são geradas. As características que recebem os valores aleatórios gerados anteriormente são selecionadas de forma aleatória. Assumindo um problema formado por 5 características (f_1, f_2, f_3, f_4, f_5) e N como sendo 2, isso significa que duas (por exemplo, f_2 e f_3) das cinco características receberiam os pesos, gerando assim 4 novas soluções. Até o momento, quatro soluções são criadas de forma que a primeira solução adiciona um valor aleatório à característica f_2, a segunda solução adiciona um outro valor aleatório à característica f_2 e a terceira e quarta soluções fazem o mesmo com a característica f_3.

Por fim, soluções extras são geradas invertendo o valor atual de P características (ver parte binária na Figura 5.1). Para completar nosso exemplo, assuma que P seja 3. Dessa forma, 3 soluções são criadas, cada uma com o *bit* de uma característica aleatória invertido. Como pode ser visto, ao final desse processo, um total de $M\ x\ N + P$ soluções vizinhas são geradas. Essa forma de gerar o conjunto inicial de soluções candidatas também é utilizada para gerar a vizinhança nos métodos TS/k-NN e SA/k-NN.

6.2.2 Metodologia empregada pela arquitetura IBPSO-PD

Uma rede neural modular baseada na arquitetura *Pattern Distributor* (ver seção 4.4.5, pág 48) foi desenvolvida usando *crosstalk table* (ver seção 4.1.2, pág. 37) (DP + CT) e algoritmos genéticos (ver seção 4.1.1, pág. 36) (DP + AG) como técnicas de decomposição de tarefas. Essas redes neurais modulares foram avaliadas com e sem o uso do algoritmo IBPSO como método de seleção de características para cada módulo individual. A arquitetura que aplica a seleção de características, através do IBPSO, para cada módulo é a arquitetura proposta (ver seção 5.2, pág. 53). Uma rede neural simples com e sem o uso do algoritmo IBPSO foi comparada com essas implementações anteriores. Esses experimentos foram realizados com as bases da Tabela 6.2.

Todos os classificadores utilizados pelas arquiteturas modulares IBPSO-PD e DP, bem como as redes neurais simples, são redes do tipo *multilayer perceptron* (MLP) com valores fixos de taxa de aprendizagem e *momentum*, respectivamente, iguais a 0.1 e 0.5. A função de ativação utilizada foi a sigmoide logística. O número máximo de épocas para todas as redes neurais foi definido como 1000. O número de neurônios escondidos foi definido empiricamente no intervalo [10,180]. Para este propósito, uma rede neural MLP com uma única camada intermediária teve sua quantidade de neurônios escondidos variando de 10 a 180, de dez em dez. Cada configuração de rede foi então avaliada com as bases de dados. Para este propósito, a base de dados foi dividida em conjunto de treinamento (50%), validação (25%) e teste (25%). Os três conjuntos sendo disjuntos. A rede neural que apresentou a maior taxa de acerto forneceu seu número de neurônios escondidos para ser usado em experimentos posteriores. Os valores encontrados podem ser vistos na Tabela 6.3.

Tabela 6.3: Quantidade de neurônios escondidos encontrada empiricamente.

Bases de dados	#Neurônios escondidos
Balance	70
Ionosphere	170
Iris	20
LIBRAS	90
Liver	10
Semeion	90
Vehicle	50
Vowel	170

A métrica de desempenho utilizada para avaliar as técnicas envolvidas nos experimentos desta seção foi a taxa de acerto. Também foi utilizado o método de validação cruzada 10-*fold*. Com o 10-*fold*, um *fold* foi utilizado para avaliar o desempenho das arquiteturas modulares e redes simples enquanto os nove *folds* restantes foram utilizados para: (i) treinar as redes neurais simples e (ii) conceber os módulos. Nas arquiteturas modulares, a cada iteração do 10-*fold*, os nove *folds* que formam o conjunto de "treinamento" são utilizados pelas técnicas de decomposição CT e AG para formar os módulos. Após formados os módulos, seus padrões são utilizados para treinar seus respectivos especialistas (redes MLP). Para essa finalidade, 80% da quantidade de padrões de cada módulo foram utilizados efetivamente para formar o conjunto de treinamento, enquanto os 20% restantes formaram o conjunto de validação com o intuito de reduzir o *overfitting*.

6.2.3 Testes de hipótese

Para tornar as comparações entre as diversas técnicas utilizadas mais fidedignas, foram realizados testes de hipótese (67). Como se está utilizando o 10-*fold* em que garante-se que as técnicas estão sendo avaliadas sobre os mesmos conjuntos de dados, foram utilizados testes de hipótese do tipo *t-student* emparelhados que aumentam ainda mais a confiabilidade nas comparações. Testes *t-student* são testes que pressupõem normalidade das amostras, ou seja, que seus dados sigam uma distribuição normal. Logo, antes de utilizar os testes *t-student* propriamente dito, faz-se necessário utilizar algum tipo de teste que verifique a normalidade das amostras.

Em nosso caso, as amostras são formadas pelas dez taxas de acertos obtidas durante o 10-*fold*. Para verificar a normalidade dessas amostras, foram aplicados testes *Lilliefors* (68). Caso verifique-se, através do teste *Lilliefors*, que as amostras não seguem uma normal (resultados em itálico nas Tabelas 6.4, 6.5, 6.7 e 6.8), o teste de *Wilcoxon* do posto sinalizado (67) é utilizado em vez do teste *t*. Ambos testes são aplicados a um nível de significância de 0.05. A hipótese nula do teste *t-student* é de que as médias de duas amostras são iguais. Caso a hipótese nula seja rejeitada ao nível de significância utilizado (0.05), significa que a média de uma das amostras é

superior à outra.

Ao comparar métodos através de testes de hipótese, só é possível fazê-lo aos pares. Dessa forma, nos resultados analisados na seção seguinte, foi comparado o método que obteve o melhor desempenho (maior taxa de acerto média) com todos os outros. Os valores nas Tabelas 6.4, 6.5, 6.7 e 6.8 com o símbolo '$-$' são estatisticamente inferiores ao melhor resultado conseguido (maior taxa de acerto média), enquanto os valores com o símbolo '$\sim$' são estatisticamente equivalentes ao melhor resultado apresentado ao nível de significância adotado.

Uma alternativa aos testes de hipótese, que só permitem a comparação aos pares, são os gráficos de caixa que permitem a comparação visual entre todos os métodos ao mesmo tempo.

6.2.4 Gráficos de Caixa

O gráfico de caixa ou gráfico *boxplot* é um gráfico para avaliar a distribuição empírica dos dados. O gráfico de caixa é um instrumento adequado para resumir o conjunto de observações de uma variável contínua. Através desse gráfico, pode-se analisar vários aspectos dos dados como tendência central, variabilidade e simetria além de valores atípicos (*outliers*) (69).

O *boxplot* é formado pelo denominado conjunto de cinco números: o limite inferior (segmento de reta horizontal ligado à base do retângulo), o primeiro quartil (a base do retângulo), a mediana (linha que divide o retângulo), o terceiro quartil (topo do retângulo) e o limite superior (segmento de reta horizontal ligado ao topo do retângulo) (69) (ver Figura 6.6, pág. 74).

A caixa (*box*) contém 50% dos dados. O limite superior da caixa indica o percentil de 75% dos dados e o limite inferior da caixa indica os 25% dos dados. A distância entre esses dois quartis é conhecida como intervalo interquartil. A linha na caixa indica a mediana dos dados e se essa linha não for equidistante dos extremos da caixa, os dados são ditos assimétricos. Os pontos, representados por cruzes, indicam os valores atípicos *outliers*. Os losangos indicam o valor médio das taxas de acerto (ver Figura 6.6, pág. 74).

A vantagem de utilizar os gráficos de caixa é que tais gráficos mostram graficamente a presença central dos dados (mediana), indica se há simetria ou assimetria dos dados com base na posição da mediana na caixa, mostra valores atípicos (*outliers*) se houver e permite a comparação visual de modelos (69).

6.3 Análise de resultados

Nesta seção, serão analisados os experimentos previamente descritos na seção 6.2. Na seção 6.3.1 estão apresentados os resultados dos experimentos quando comparado o método BPSO/k-NN aos métodos TS/k-NN, SA/k-NN e k-NN. Na seção 6.3.2, é verificada a eficácia da arquitetura IBPSO-PD quando comparada às arquiteturas DP (sem seleção de características por módulo) e às redes neurais simples com o uso do algoritmo de seleção de características IBPSO (RNA + IBPSO) e sem o uso do algoritmo IBPSO (RNA). Como são utilizadas as técnicas

de decomposição CT e AG em ambas as arquiteturas modulares (IBPSO-PD e DP), há quatro cenários possíveis: arquitetura proposta com decomposição de tarefas via AG (IBPSO-PD + AG), arquitetura proposta com decomposição de tarefas via CT (IBPSO-PD + CT), arquitetura distribuidor de padrões com decomposição de tarefas via AG (DP + AG) e arquitetura distribuidor de padrões com decomposição de tarefas via CT (DP + CT).

Problemas de seleção de características têm como objetivo selecionar um pequeno subconjunto de características que seja capaz de reter a informação discriminatória presente na base de dados original. Logo, os principais resultados a serem vistos neste trabalho, já que os dois métodos propostos tratam de seleção de características, são a redução de dimensionalidade obtida bem como as taxas de acerto atingidas durante a classificação. Nas duas seções seguintes, constarão as tabelas com as taxas de acerto médias obtidas durante o 10-*fold*, gráficos demonstrando as reduções de dimensionalidade, gráficos de caixa para permitir a comparação de todos os métodos visualmente e os resultados dos testes estatísticos.

6.3.1 Análise dos resultados da técnica BPSO/k-NN

As Tabelas 6.4 e 6.5 contêm os resultados para os experimentos descritos na seção 6.2.1. Os dados contidos nessas duas tabelas devem ser analisados conjuntamente já que se está procedendo à comparação de todas as técnicas entre si. As taxas de acerto estão em % (melhores resultados em negrito) e os desvios-padrão estão entre parênteses. Na Tabela 6.4, o número de vizinhos do algoritmo k-NN foi definido como o valor que propiciou a maior taxa de acerto. O valor de k do classificador k-NN pode ser diferente para cada base.

Tabela 6.4: Taxa de acerto média (%) e desvio-padrão entre parênteses para os experimentos com as técnicas k-NN e TS/k-NN.

	Classificadores	
Bases de dados	**k-NN**	**TS/k-NN**
Diabetes	69.50 (0.0465)$^{-}$	72.24 (0.0600)$^{\sim}$
Glass	68.55 (0.0982)$^{\sim}$	66.32 (0.0912)$^{\sim}$
Heart	65.19 (0.1050)$^{-}$	*72.59 (0.0658)*$^{\sim}$
Ionosphere	84.33 (0.0678)$^{-}$	87.17 (0.0729)$^{\sim}$
Sonar	82.00 (0.1033)$^{-}$	82.86 (0.0796)$^{-}$
Vehicle	*64.76 (0.0515)*$^{\sim}$	67.93 (0.0501)$^{\sim}$

A Figura 6.1 (pág. 69) mostra a mínima (d_{min}) e a máxima (d_{max}) dimensões obtidas pelos três métodos híbridos durante o 10-*fold* para cada base de dados. Os valores estão em porcentagem, i.e., os valores mínimos e máximos divididos pela dimensionalidade original, representada por (d), que por sua vez, representa o conjunto original de características, também dado em porcentagem.

A Tabela 6.5 mostra que as taxas de acerto obtidas pela técnica proposta superam os resultados obtidos pela técnica SA/k-NN e pelas técnicas k-NN e TS/k-NN (ver Tabela 6.4)

Tabela 6.5: Taxa de acerto média (%) e desvio-padrão entre parênteses para os experimentos com as técnicas SA/k-NN e BPSO/k-NN.

	Classificadores	
Bases de dados	**SA/k-NN**	**BPSO/k-NN**
Diabetes	71.89 (0.0736)$^{\sim}$	**73.52** (0.0436)
Glass	66.72 (0.1232)$^{\sim}$	**68.70** (0.1079)
Heart	67.78 (0.0820)$^{\sim}$	**74.44** (0.0750)
Ionosphere	84.62 (0.0573)$^{-}$	**89.75** (0.0426)
Sonar	80.07 (0.1397)$^{\sim}$	**86.57** (0.0746)
Vehicle	66.02 (0.0552)$^{\sim}$	**68.18** (0.0581)

para todas as bases de dados. Analisando a redução de dimensionalidade através da Figura 6.1, constata-se que, para as bases *ionosphere*, *heart*, *vehicle* e *sonar*, as reduções de dimensionalidade mínima e máxima obtidas pelo método BPSO/k-NN foram maiores que as reduções obtidas pelos outros dois métodos híbridos (SA/k-NN e TS/k-NN). Para a base *diabetes*, os melhores resultados em termos de redução de dimensionalidade foram obtidos pelo método TS/k-NN, enquanto para a base *glass*, os melhores resultados foram obtidos pelo método SA/k-NN. Para a base *sonar*, o método BPSO/k-NN atingiu as maiores reduções absolutas considerando tanto o valor máximo quanto o mínimo.

Os métodos também podem ser comparados visualmente através dos gráficos de caixa presentes na Figura 6.2 (pág. 70). Através desses gráficos, chega-se a conclusões semelhantes às obtidas das Tabelas 6.4 e 6.5.

6.3.2 Análise dos resultados da técnica IBPSO-PD

Vários experimentos foram realizados com objetivo de demonstrar a eficácia do método proposto. Nesse caso, uma arquitetura de rede neural modular que seleciona diferentes conjuntos de características por módulo, de nome IBPSO-PD. Oito bases foram utilizadas para esse propósito, apresentadas na Tabela 6.2. A arquitetura proposta é comparada à arquitetura *Pattern Distributor* (DP) e às redes neurais simples (com e sem o uso do algoritmo de seleção IBPSO). Ambas as arquiteturas de RNM utilizam como técnicas de decomposição de tarefas algoritmos genéticos e *crosstalk table*. A quantidade de módulos (m) encontrada pelo método AG é definida automaticamente. Logo, apenas uma configuração de rede (IBPSO-PD + AG) e (DP + AG) foi desenvolvida. Já no método CT, a quantidade de módulos é um parâmetro a ser definido pelo desenvolvedor. Para definir um número de módulos apropriado para os experimentos, a arquitetura (DP + CT) teve seu número m variado de 2 até c (número de classes) para cada base de dados.

A configuração de rede que apresentou a maior taxa de acerto forneceu seu número de módulos para os experimentos realizados nesta seção com a técnica CT. As Figuras 6.3 e 6.4 (págs. 71 e 72) demonstram o comportamento da taxa de acerto quando variado o número de módulos em uma arquitetura (DP + CT) para as bases utilizadas, enquanto a tabela 6.6

mostra a quantidade de módulos retornada pelo AG de maneira automática e a quantidade de módulos definida empiricamente para o método CT. Nas Figuras 6.3 e 6.4, os valores da posição 1 correspondem aos desempenhos de redes neurais simples (redes neurais modulares degeneradas) e os valores inteiros maiores que 1 representam a quantidade de grupos.

Tabela 6.6: Número de módulos utilizados pelas técnicas de decomposição de tarefas.

Bases de dados	**AG**	**CT**
Balance	2	3
Ionosphere	2	2
Iris	2	2
LIBRAS	3	12
Liver	2	2
Semeion	2	7
Vehicle	2	4
Vowel	2	5

Como o objetivo da arquitetura proposta é realizar a seleção de características automática de cada módulo que compõe a topologia da RNM, os principais resultados a serem analisados são a redução de dimensionalidade obtida para cada módulo pelo algoritmo IBPSO bem como a taxa de acerto atingida para cada base de dados. A Figura 6.5 (pág. 73) apresenta a dimensionalidade original (d), o valor mínimo (d_{min}) e o valor máximo (d_{max}) de dimensões após a realização da seleção de características, todos em porcentagem. No caso das abordagens modulares (DP + AG, DP + CT, IBPSO-PD + AG, IBPSO-PD + CT), esses valores da Figura 6.5 representam a quantidade mínima de informação (características) e a quantidade máxima de informação utilizada em cada módulo. No caso das redes neurais simples com seleção de características (RNA + IBPSO), esses valores do gráfico indicam as quantidades mínima e máxima (em porcentagem) de características utilizadas durante as dez iterações do 10-*fold*.

As Tabelas 6.7 e 6.8 mostram as taxas de acerto e os desvios-padrão, entre parênteses, obtidos durante o 10-*fold* para as seis técnicas implementadas. Os dados contidos nessas duas tabelas devem ser analisados conjuntamente uma vez que se está procedendo à comparação de todas as técnicas entre si. As melhores performances estão em negrito. A quantidade de módulos utilizada em cada arquitetura modular pode ser vista na Tabela 6.6.

Para todas as bases analisadas, a arquitetura proposta foi melhor do que as redes neurais simples (com e sem o uso de seleção de características) e do que as redes neurais modulares sem seleção de características nos módulos (arquiteturas DP), exceto para a base *iris*. Para esta base, embora a arquitetura proposta tenha obtido resultado superior aos resultados fornecidos pelas RNAs simples, o melhor resultado, no geral, foi atingido pela rede neural modular sem seleção de características com método de decomposição de tarefas realizado por meio de algoritmo genético (arquitetura DP + AG). Verifica-se também que, para todas as bases, as redes neurais modulares forneceram melhores resultados que as redes neurais simples.

Através da Tabela 6.8, podemos observar que a ideia de aplicar seleção de características

Tabela 6.7: Taxa de acerto média (%) e desvio-padrão entre parênteses para os experimentos com RNAs com e sem seleção de características.

	Rede Neural Simples	
Bases de dados	**RNA**	**RNA + IBPSO**
Balance	*95.57 (2.82)*$^{\sim}$	95.85 (2.38)$^{\sim}$
Ionosphere	92.87 (3.63)$^{\sim}$	*91.17 (3.89)*$^{-}$
Iris	*95.33 (4.50)*$^{\sim}$	91.33 (6.32)$^{-}$
LIBRAS	*65.00 (23.36)*$^{\sim}$	69.17 (9.75)$^{-}$
Liver	67.91 (11.34)$^{\sim}$	57.88 (7.54)$^{-}$
Semeion	91.34 (1.21)$^{\sim}$	88.70 (1.66)$^{-}$
Vehicle	50.50 (6.43)$^{\sim}$	51.33 (12.26)$^{\sim}$
Vowel	32.22 (4.91)$^{-}$	31.11 (4.33)$^{-}$

Tabela 6.8: Taxa de acerto média (%) e desvio-padrão entre parênteses para os experimentos com RNMs com e sem seleção de características nos módulos.

	Redes Neurais Modulares			
Bases de dados	**DP + AG**	**IBPSO-PD + AG**	**DP + CT**	**IBPSO-PD + CT**
Balance	*90.74 (3.92)*$^{-}$	86.06 (7.85)$^{-}$	**96.20** (2.81)	**96.20** (2.81)
Ionosphere	**94.87** (2.95)	**94.87** (2.95)	94.31 (4.84)$^{\sim}$	94.31 (4.84)$^{\sim}$
Iris	**98.00** (3.22)	*97.33 (3.44)*$^{\sim}$	*94.00 (10.16)*$^{\sim}$	*86.00 (14.89)*$^{\sim}$
LIBRAS	*51.94 (29.34)*$^{\sim}$	*60.83 (26.53)*$^{\sim}$	*72.22 (9.07)*$^{\sim}$	**74.44** (9.05)
Liver	**69.09** (9.68)	**69.09** (9.68)	62.14 (10.79)$^{-}$	62.14 (10.79)$^{-}$
Semeion	90.84 (2.44)$^{\sim}$	89.02 (2.02)$^{-}$	**91.40** (2.31)	**91.40** (2.40)
Vehicle	49.46 (6.42)$^{\sim}$	52.05 (9.20)$^{\sim}$	**53.13** (7.78)	**53.13** (7.78)
Vowel	34.75 (4.69)$^{\sim}$	31.31 (4.57)$^{-}$	36.67 (7.87)$^{\sim}$	**38.08** (4.44)

a módulos individuais de uma RNM foi proveitosa, visto que, para sete das oito bases de dados, as taxas de acerto foram maiores com o espaço de características reduzido. Para as bases *LIBRAS* e *vowel*, por exemplo, a arquitetura IBPSO-PD usando CT como técnica de decomposição obteve as melhores taxas de acerto quando comparada com os outros métodos implementados. Para a base *LIBRAS*, apenas 48 características no máximo foram utilizadas nos 12 módulos quando utilizada a técnica CT, em vez do universo original de 91 características, uma redução de dimensionalidade expressiva que prova que nem todas as características são essenciais à tarefa de classificação e que nem todas as características são necessárias aos diferentes módulos de uma RNM, uma vez que a tarefa de cada módulo é menor e mais simples do que o problema original.

Para as bases *balance*, *ionosphere*, *liver*, *semeion* e *vehicle*, as taxas de acerto atingidas pela abordagem modular sem seleção de características (DP) foram similares aos resultados obtidos pela abordagem IBPSO-PD. Entretanto, a quantidade de características utilizada pela abordagem IBPSO-PD foi muito inferior quando comparada à abordagem DP (ver Figura 6.5, pág. 73). Nos experimentos com a base *ionosphere*, por exemplo, a arquitetura IBPSO-PD (usando CT como técnica de decomposição) igualou a performance obtida pelas redes neurais modulares sem seleção de características (DP + CT), com no máximo 41% da quantidade de características original em cada módulo. Já para a base *liver*, a redução foi ainda mais expressiva: apenas 33%, no máximo, da quantidade de características original foi utilizada em cada módulo quando utilizada a técnica AG (IBPSO-PD + AG). Embora a motivação inicial do presente trabalho tenha sido investigar a aplicação de um método de seleção de características para

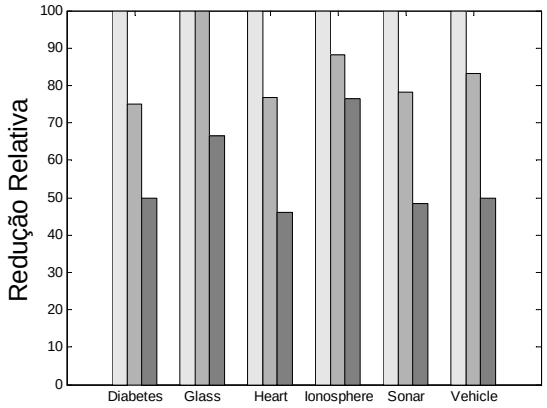
os módulos de uma RNM com o intuito de selecionar as características mais adequadas para cada módulo, foi utilizado também o mesmo método de seleção de características (IBPSO) aplicado a redes neurais simples a fim de verificar se seriam atingidos resultados similares ou superiores aos obtidos pelas redes neurais simples sem seleção de características. Dessa forma, mostrar-se-ia que o método de seleção aplicado a redes neurais simples seria proveitoso uma vez que seriam atingidos resultados iguais ou até superiores aos resultados obtidos pelas redes neurais simples sem seleção de características, porém, fazendo uso de uma quantidade reduzida de características. Entretanto, os resultados presentes na tabela 6.7 mostram que para apenas três das oito bases utilizadas, essa premissa foi verificada. Para a maioria das bases, as redes neurais simples combinadas com o algoritmo de seleção de características IBPSO apresentaram resultados inferiores aos obtidos pelas redes neurais simples sem seleção de características.

Os métodos também podem ser comparados visualmente através dos gráficos de caixa presentes nas Figuras 6.6 e 6.7 (págs. 74 e 75). Através desses gráficos, chega-se a conclusões semelhantes às obtidas das Tabelas 6.7 e 6.8.

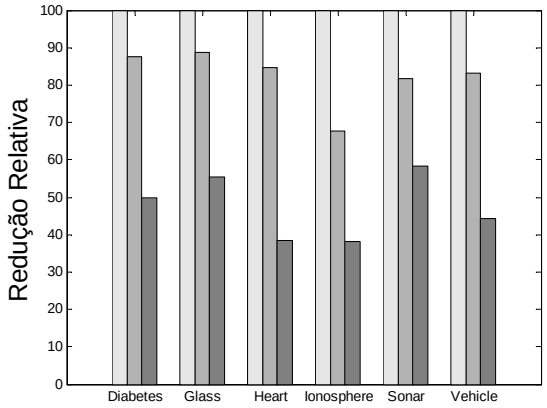
6.4 Considerações Finais

Neste capítulo, tratou-se dos experimentos e da análise de resultados. Primeiramente, foram descritas as bases de dados utilizadas nos experimentos com as duas técnicas propostas: o método híbrido de seleção e ponderação de características baseado no PSO binário e a arquitetura de rede neural modular que seleciona automaticamente as características mais relevantes para cada módulo da topologia da rede. Em seguida, foram definidas as metodologias utilizadas por ambos os métodos. Foram abordados também os testes de hipótese e os gráficos de caixa, duas abordagens utilizadas neste trabalho para proceder à comparação dos métodos propostos com as técnicas que serviram de parâmetro. Na análise de resultados, os experimentos mostraram que a proposta do método híbrido de seleção e ponderação de características baseado no PSO binário obteve resultados superiores aos resultados obtidos pelos métodos híbridos TS/k-NN e SA/k-NN e pelo k-NN clássico que serviram de comparação. Quanto à proposta da arquitetura de rede neural modular que realiza seleção de características de forma automática para os respectivos módulos da rede, bons resultados também foram obtidos. A arquitetura de RNM proposta (IBPSO-PD) obteve resultados superiores às redes neurais simples com e sem seleção de características. Quando comparada com as redes neurais modulares sem seleção de características, a arquitetura proposta quando não superou tais redes em termos de taxa de classificação, igualou-as, porém, fazendo uso de um conjunto de características bastante reduzido enquanto que as redes neurais modulares sem seleção fizeram uso de todo o conjunto de características.

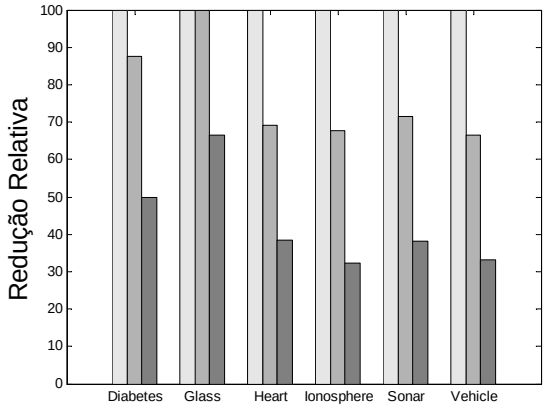
Figura 6.1: Redução de dimensionalidade obtida pelos métodos híbridos.



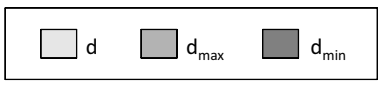
(a) TS/k-NN



(b) SA/k-NN

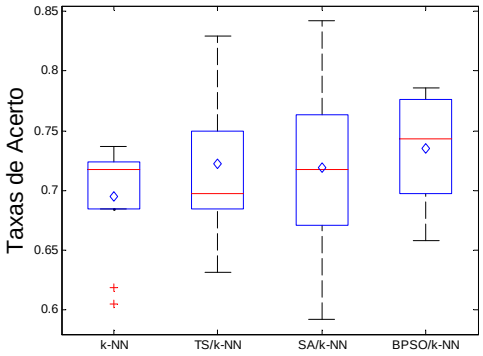


(c) BPSO/k-NN

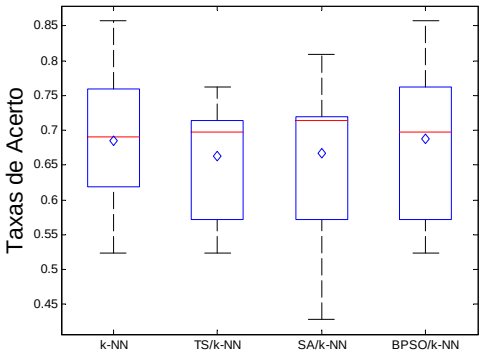


Fonte: o autor

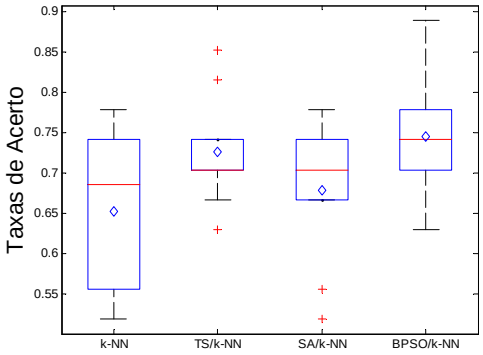
Figura 6.2: Gráficos de caixa para os experimentos com o método BPSO/k-NN. Os losangos representam a taxa de acerto média.



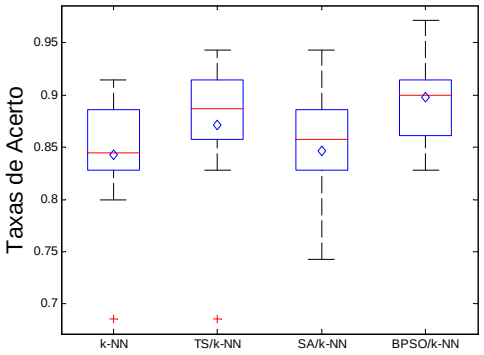
(a) Diabetes



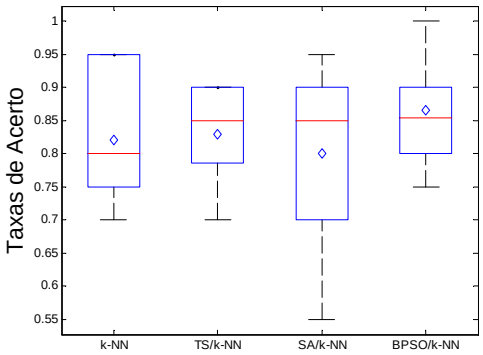
(b) Glass



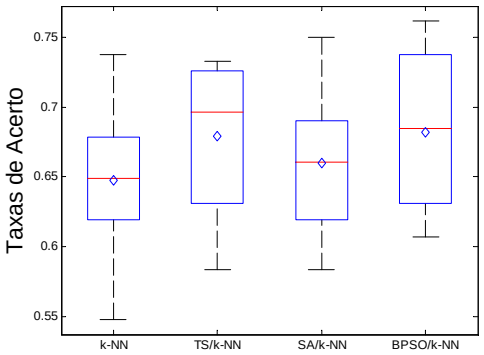
(c) Heart



(d) Ionosphere



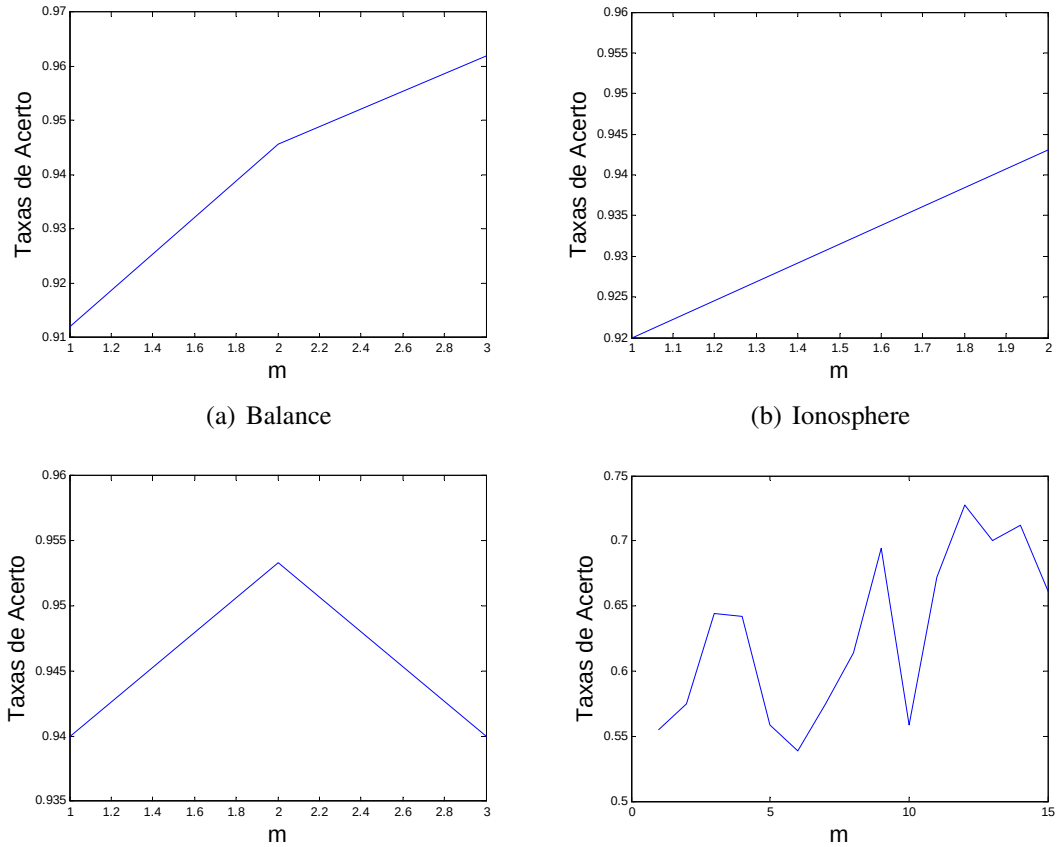
(e) Sonar



(f) Vehicle

Fonte: o autor

Figura 6.3: Variação do número de grupos para determinação do valor *m* utilizado pela técnica CT.



(a) Balance

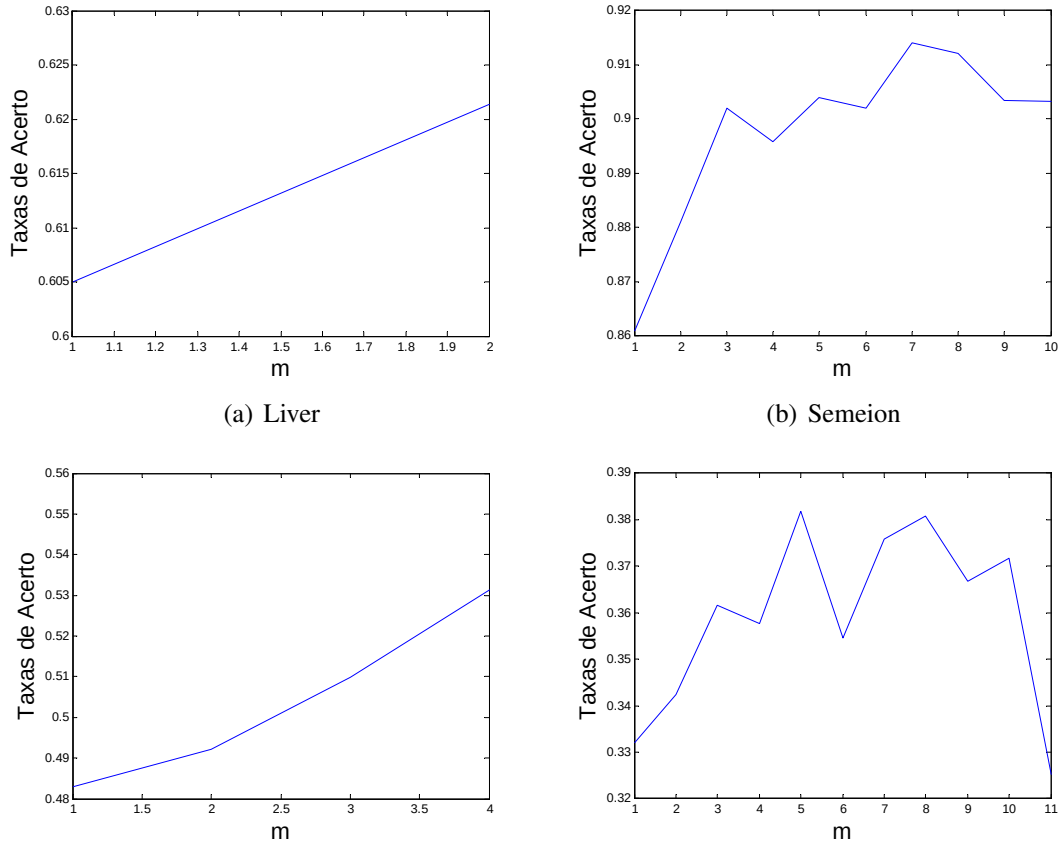
(b) Ionosphere

(c) Iris

(d) LIBRAS

Fonte: o autor

Figura 6.4: Variação do número de grupos para determinação do valor *m* utilizado pela técnica CT.



(a) Liver

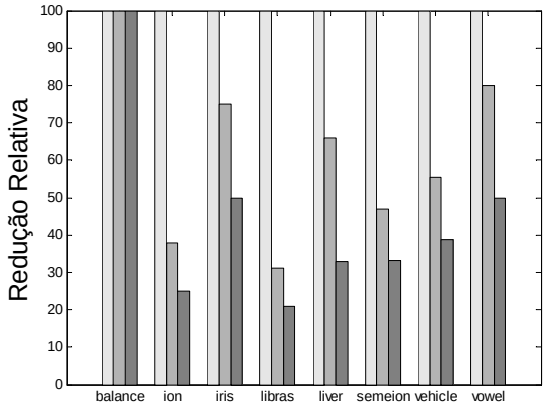
(b) Semeion

(c) Vehicle

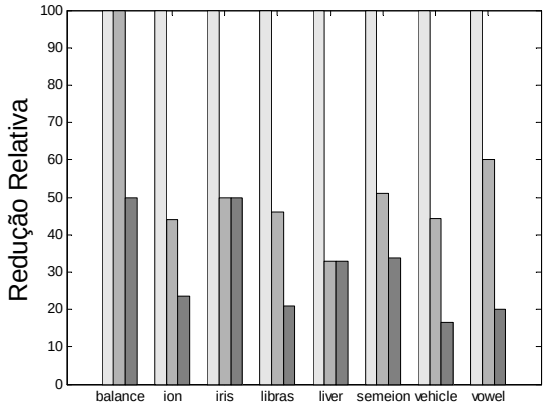
(d) Vowel

Fonte: o autor

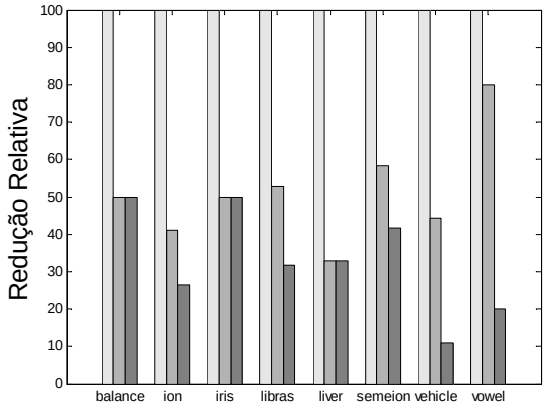
Figura 6.5: Redução de dimensionalidade mínima e máxima em porcentagem.



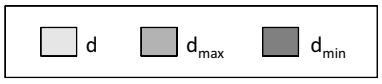
(a) RNA + IBPSO



(b) IBPSO-PD + AG

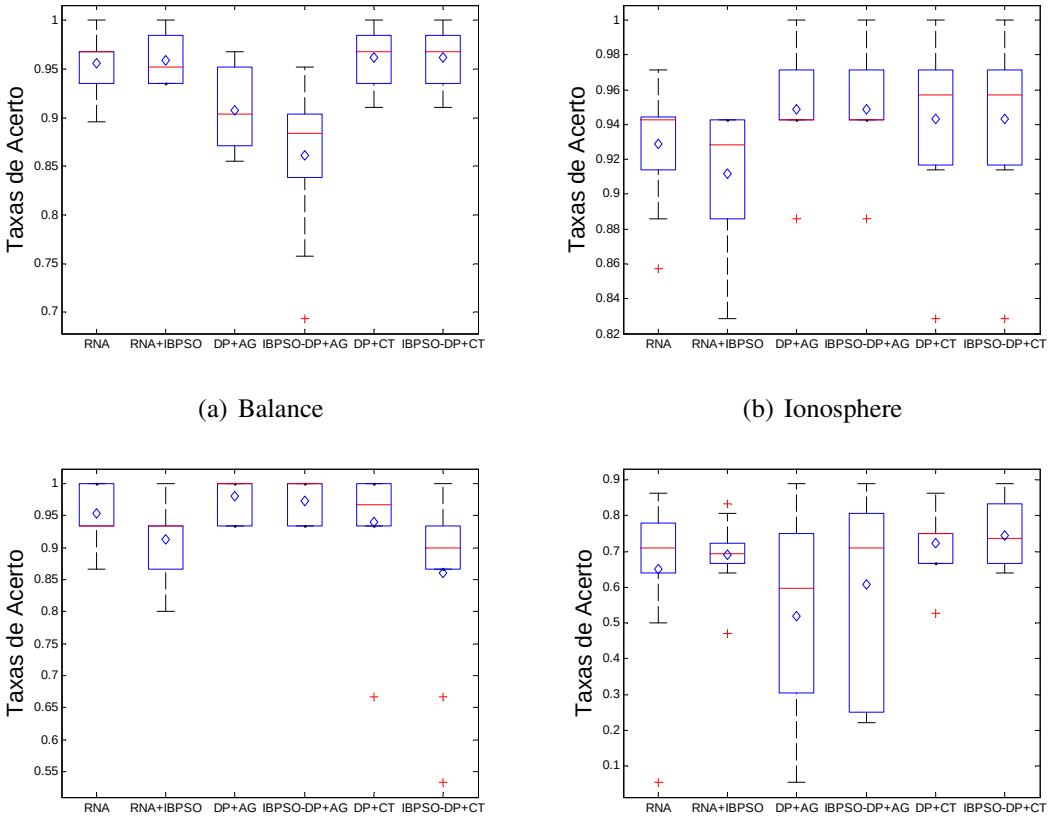


(c) IBPSO-PD + CT



Fonte: o autor

Figura 6.6: Gráficos de caixa para os experimentos com a arquitetura IBPSO-PD. Os losangos representam a taxa de acerto média.



(a) Balance

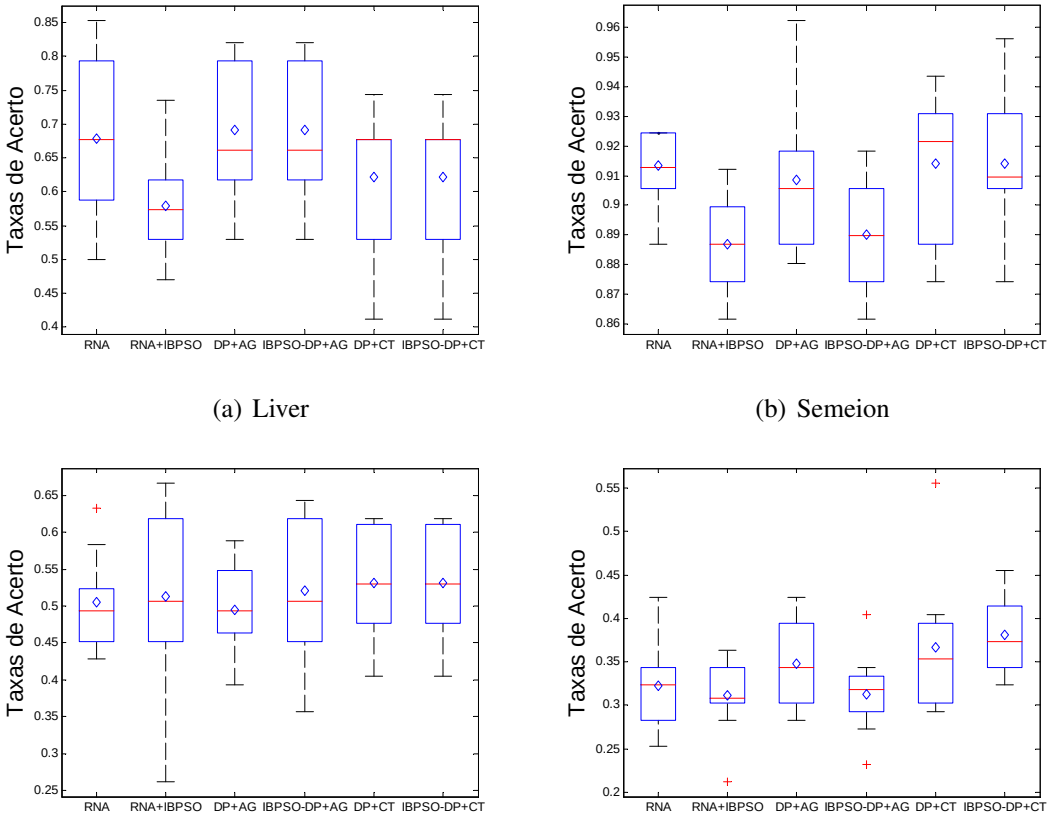
(b) Ionosphere

(c) Iris

(d) LIBRAS

Fonte: o autor

Figura 6.7: Gráficos de caixa para os experimentos com a arquitetura IBPSO-PD. Os losangos representam a taxa de acerto média.



(a) Liver

(b) Semeion

(c) Vehicle

(d) Vowel

Fonte: o autor

7
Conclusão

As redes neurais artificiais têm sido largamente utilizadas para a resolução dos mais diversos problemas, tais como: previsão de séries temporais, processamento de imagens e reconhecimento de padrões. Elas foram propostas inicialmente para emular as redes neurais biológicas. Com forte inspiração biológica, as RNAs mostram vários pontos em comum com as redes neurais naturais, como a presença de unidades de processamento simples, tolerância a falhas e capacidade de aprendizado através de exemplos.

Para aproximar mais ainda as redes neurais artificiais das redes neurais biológicas, foram propostas as redes neurais modulares. Inspiradas na modularização presente no cérebro humano, isto é, na maneira como o cérebro humano faz uso de regiões específicas do córtex cerebral para realizar tarefas simples do dia a dia como ouvir, falar e enxergar, por exemplo, as RNMs fazem uso da modularização para superarem de alguma forma as RNAs.

Para este propósito, o problema original em uma RNM é decomposto, através de técnicas de decomposição, em subproblemas menores e mais simples de serem resolvidos, utilizando uma abordagem de dividir-para-conquistar. Esses diferentes subproblemas são então tratados por diferentes regiões especialistas (módulos). Cada módulo contém um especialista (rede neural simples) que será responsável por solucionar seu respectivo problema. A resposta final da RNM será a combinação das respostas produzidas pelos diferentes módulos ou a resposta produzida pelo módulo vencedor (abordagem competitiva).

Um problema que se observa ao utilizar redes neurais modulares é a utilização de todo o universo original de características pelos diferentes módulos. Ora, se os módulos são responsáveis por tratar problemas menores que o problema original, reconhecer um determinado número de classes k, no contexto de problemas de classificação, onde k é menor que o total de classes original d da base de dados, é provável que informação adicional e desnecessária esteja sendo repassada para os diferentes módulos. Como no presente trabalho foram utilizadas técnicas de decomposição baseadas no agrupamento de classes próximas umas das outras no espaço de características, garante-se que $k < d$.

Com o objetivo de selecionar as características mais relevantes para cada módulo, foi proposta uma arquitetura de rede neural modular (IBPSO-PD), baseada na arquitetura *Pattern Distributor*, que utiliza um algoritmo baseado em enxame de partículas (IBPSO) para selecionar diferentes conjuntos de características por módulo.

A arquitetura proposta descarta características irrelevantes para cada módulo e esta eliminação melhorou a performance de classificação em termos de taxa de acerto. Com um

menor número de características por módulo, as redes especialistas de cada módulo têm seus custos de armazenamento e computacional diminuídos, uma vez que são necessários menos neurônios na camada de entrada e, por consequência, menos conexões entre a camada de entrada e a camada intermediária.

Com menos conexões, as redes neurais de cada módulo têm diminuída a interferência entre os pesos durante a fase de atualização dos mesmos. Os resultados experimentais mostraram que a abordagem proposta ou obteve melhores taxas de classificação ou obteve taxas de classificação iguais quando comparadas aos resultados obtidos pelas redes neurais modulares tradicionais que não fazem uso de seleção nos seus módulos. Porém, no método proposto, tais taxas de classificação foram conseguidas utilizando espaços de características bastante reduzidos.

Quanto ao método híbrido de seleção e ponderação de características baseado no algoritmo BPSO, os experimentos mostraram a eficácia do método proposto, visto que ele atingiu melhores taxas de acerto quando comparado aos métodos k-NN, TS/k-NN e SA/k-NN. Em termos de redução de dimensionalidade, a técnica proposta também obteve bons resultados. Para quase todas as bases de dados, o tamanho do subconjunto de características gerado pelo método proposto foi menor que os subconjuntos gerados pelas outras técnicas híbridas TS/k-NN e SA/k-NN. Estes resultados obtidos mostram que técnicas que realizam seleção e ponderação de características simultaneamente melhoram as taxas de acerto durante a classificação, diminuindo o tamanho do conjunto de características original.

7.1 Trabalhos Futuros

Como visto no capítulo 4, que trata de redes neurais modulares, uma etapa importante na concepção dessas é a fase de decomposição de tarefas. Uma má decomposição pode prejudicar o desempenho da RNM como um todo, visto que a decomposição de tarefas fornece diretamente a quantidade de módulos da topologia final.

Como trabalho futuro, pode-se investigar o uso de outros métodos de decomposição. No trabalho, foram utilizadas as técnicas *crosstalk table* e algoritmos genéticos, que realizam a decomposição agrupando padrões de classes semelhantes. Uma ideia é investigar o uso de técnicas de decomposição que realizam o agrupamento de padrões semelhantes. Isto levaria a módulos redundantes, ou seja, classes presentes em mais de um módulo. A presença de módulos redundantes é ideal quando se quer criar um sistema mais tolerante a falhas e que implique uma possível maior taxa de acerto durante o processo de classificação visto que em vez de apenas um módulo ser capaz de identificar padrões de uma determinada classe, teriam-se dois ou mais módulos com a mesma capacidade discriminatória.

Outra possibilidade é utilizar diferentes algoritmos de seleção embutidos na arquitetura e avaliar o desempenho das arquiteturas geradas com essas técnicas de seleção distintas. A possibilidade de combinações de técnicas de decomposição de tarefas e algoritmos de seleção é grande, visto que a arquitetura é flexível o suficiente para comportar diferentes métodos.

Neste trabalho foram utilizadas redes supervisionadas do tipo *multilayer perceptron* para formar os módulos especialistas. Uma possibilidade de trabalho futuro é investigar o uso de redes GHSOM (*Growing Hieraquical Sel Organizing Maps*) que são mapas auto-organizáveis construtivos hierárquicos (70). São redes não supervisionadas que podem ser utilizadas como método de agrupamento hierárquico divisivo. A ideia por trás das redes GHSOM é que se tem um mapa no mais alto nível que é treinado com todos os padrões de entrada que formam a base de dados. Depois de treinada, avalia-se o erro de cada nodo no mapa. Para cada nodo cujo erro estiver acima de um limiar, cria-se um novo mapa SOM no nível seguinte, que será treinado apenas com os padrões de entrada mapeados no nodo que lhe deu origem. No contexto de redes neurais modulares, cada mapa SOM seria visto como um módulo treinado com um subconjunto dos padrões de entrada. Ao utilizar as redes GHSOM, técnicas de decomposição de tarefas seriam prescindíveis já que a própria rede faria a decomposição de tarefas intrinsecamente ao adicionar de forma construtiva novas redes SOM.

O uso do algoritmo IBPSO também pode ser investigado no módulo distribuidor, bem como em outras arquiteturas de redes neurais modulares como mistura de especialistas, por exemplo. Sendo um trabalho pouco explorado na literatura, as possibilidades de investigação são diversas.

Referências

[1] MCCULLOCH, W.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, Kluwer Academic Publisher, v. 5, n. 4, p. 115–143, 1943.

[2] WIDROW, B.; RUMELHART, D. E.; LEHR, M. A. Neural networks: applications in industry, business and science. *Communications of the ACM*, USA, New York, v. 37, n. 3, p. 93–105, 1994.

[3] GARDNER, D. Introduction: toward neural networks. *The neurobiology of neural networks*, p. 1–20, 1993.

[4] BULLOCK, T. Integrative systems research in the brain: resurgence and new opportunities. *Annual Review of Neuroscience*, v. 16, p. 1–15, 1993.

[5] KOHONEN, T. *Self-organization and associative memory*. [S.l.]: Springer Series in Information Sciences, 1984. 312 p.

[6] MICHELI-TZANAKOU, E. A neural network model of the vertebrate retina. In: CONFERENCE IEEE ENGINEERING IN MEDICINE AND BIOLOGY SOCIETY, 9., 1987, Massachusets. *[Trabalho apresentado]...* Massachusets: [s.n.], 1987. p. 13–16.

[7] AUDA, G.; KAMEL, M. Modular neural networks: a survey. *International Journal of Neural Systems*, v. 9, n. 2, p. 129–151, 1999.

[8] GUAN, S.; LIU, J. Feature selection for modular neural networks based on incremental training. *Journal of Intelligent Systems*, v. 14, n. 4, p. 353–383, 2005.

[9] HAYKIN, S. *Neural networks: a comprehensive foundation*. 2nd. ed. McMaster University, Canada: Pearson Education, 1999.

[10] VERISSIMO, E. et al. Dimensionality reduction in task decomposition for modular neural networks. In: INTERNATIONAL CONFERENCE ON TOOLS WITH ARTIFICIAL INTELLIGENCE, 24., 2012, Athens. *Proceedings...* Athens: IEEE, 2012. v. 1, p. 886–891.

[11] BOUVEYRON, C.; GIRARD, S.; SCHMID, C. High-dimensional data clustering. *Computational Statistics and Data Analysis*, v. 52, p. 502–519, 2007.

[12] CHUANG, L. Y. et al. Improved binary pso for feature selection using gene expression data. *Computational Biology and Chemistry*, v. 32, n. 1, p. 29–38, 2008.

[13] GUAN, S. U.; ZHU, F. M. Modular feature selection using relative importance factors. *International Journal of Computational Intelligence and Applications*, v. 4, n. 1, p. 57–75, 2004.

[14] BARROS, A.; CAVALCANTI, G. D. C. Combining global optimization algorithms with a simple adaptive distance for feature selection and weighting. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2008, Hong Kong. *Proceedings...* Hong Kong: IEEE, 2008. p. 3518–3523.

[15] BELLMAN, R. *Dynamic programming*. Princeton, USA: Princeton University Press, 1957.

[16] LIU, H.; MOTODA, H. *Feature selection for knowledge discovery and data mining*. [S.l.]: Kluwer Academic Publishers, 1998.

[17] MICHIE, D.; SPIEGELHALTER, D. J.; TAYLOR, C. C. *Machine learning, neural and statistical classification*. [S.l.]: Ellis Horwood, 1994.

[18] KOHAVI, R.; JOHN, G. Wrappers for feature subset selection. *Artificial Intelligence*, United Kingdom, Essex, v. 97, p. 273–324, 1997.

[19] MITCHELL, T. M. *Machine learning*. [S.l.]: McGraw-Hill, 1997.

[20] TAHIR, M. A.; BOURIDANE, A.; KURUGOLLU, F. Simultaneous feature selection and weighting using hybrid tabu search/k-nearest neighbor classifier. *Patterns Recognition Letters*, v. 28, n. 4, p. 438–446, 2007.

[21] WETTSCHERECK, D.; AHA, D. W.; MOHRI, T. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, v. 11, n. 1, p. 273–314, 1997.

[22] RAYMER, M. L. et al. Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, v. 4, n. 2, p. 164–171, 2000.

[23] PUNCH, W. F. et al. Further research on feature selection and classification using genetic algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 5., 1993, San Francisco. *Proceedings...* San Francisco: Morgan Kaufmann Publishers, 1993. p. 557–564.

[24] GUVENIR, H. A.; AKKUS, A. Weighted k-nearest neighbor classification on feature projection. In: INTERNATIONAL SYMPOSIUM ON COMPUTER AND INFORMATION SCIENCES, 12., 1997. *Proceedings...* [S.l.]: [s.n.], 1997.

[25] PAREDES, R.; VIDAL, E. A class-dependent weighted dissimilarity measure for nearest neighbor classification problems. *Pattern Recognition Letters*, v. 21, n. 12, p. 1027–1036, 2000.

[26] PAPALAMBROS, P. Y.; WILDE, D. J. *Principles of optimal design: modeling and computation*. [S.l.]: Cambridge University Press, 2000. 412 p.

[27] GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computation Operations Research*, v. 13, n. 5, p. 533–549, 1986.

[28] HANSEN, P. The steepest ascent mildest descent heuristic for combinatorial programming. In: CONGRESS ON NUMERICAL METHODS IN COMBINATORIAL OPTIMIZATION, 1986, Capri. *Proceedings...* Capri: [s.n.], 1986.

[29] GLOVER, F.; LAGUNA, M. *Tabu search*. USA, Boston: Kluwer Academic Publishers, 1997. 408 p.

[30] PHAM, D. T.; KARABOGA, D. Intelligent optimization techniques. *Springer-Verlag*, 2000.

[31] KIRKPATRICK, S.; GELLAT, D. C.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680, 1983.

[32] METROPOLIS, N. et al. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, v. 21, p. 1087–1092, 1953.

[33] DARWIN, C. *On the origin of species*. [S.l.]: John Murray, 1859.

[34] EIBEN, A. E.; SMITH, J. E. *Introduction to evolutionary computing*. [S.l.]: Springer, 2000.

[35] KENNEDY, J.; EBERHART, R. C. Particle swarm optimization. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1995, Perth. *Proceedings...* Perth: IEEE, 1995. v. 4, p. 1942–1948.

[36] ENGELBRECHT, A. P. *Fundamentals of computational swarm intelligence*. University of Pretoria, South Africa: [s.n.], 2005.

[37] BERGH, F. *An analysis of particle swarm optimizers*. Tese (Doutorado), University of Pretoria, South Africa, 2001.

[38] SHI, Y.; EBERHART, R. C. A modified particle swarm optimizer. In: IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE, 1998, Anchorage. *Proceedings...* Anchorage: IEEE, 1998. p. 69–73.

[39] KENNEDY, J.; EBERHART, R. C. A discrete binary version of the particle swarm algorithm. In: INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 1997, Orlando. *Proceedings...* Orlando: IEEE, 1997. v. 5, p. 4104–4109.

[40] GUAN, S.; BAO, C.; NEO, T. Reduced pattern training based on task decomposition using pattern distributor. *IEEE Transactions on Neural Networks*, v. 18, n. 6, p. 1738–1749, 2007.

[41] FISHER, R. A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, v. 7, n. 2, p. 179–188, 1936.

[42] DUDA, R.; HART, P.; STORK, D. *Pattern recognition*. 2nd. ed. [S.l.]: John Wiley & Sons, 2001.

[43] ALVES, V. M. O.; CAVALCANTI, G. D. C. Tree architecture pattern distributor: a task decomposition classification approach. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2009, Atlanta. *Proceedings...* Atlanta: IEEE, 2009. p. 133–140.

[44] KAUFMAN, L.; ROUSSEEUW, P. J. *Finding groups in data: an introduction to cluster analysis*. 9th. ed. [S.l.]: Wiley-Interscience, 1990.

[45] HRYCE, T. A modular architecture for efficient learning. *International Conference on Neural Networks*, USA, California, v. 1, p. 557–562, 1993.

[46] BOLLIVIER, M. de; GALLINARI, P.; THIRIA, S. Cooperation of neural nets for robust classification. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1990, San Diego. *Proceedings...* San Diego: IEEE, 1990. v. 1, p. 113–120.

[47] CHIANG, C.; FU, H. A divide-and-conquer methodology for modular supervised neural network design. In: WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE, 1994, Orlando. *Proceedings...* Orlando: IEEE, 1994. v. 1, p. 119–124.

[48] IWATA, A.; KAWAJIRI, H.; SUZUMURA, N. Classification of hand-written digits by a large scale neural network combnet-ii. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1991, Singapore. *Proceedings...* Singapore: IEEE, 1991. v. 2, p. 1021–1026.

[49] NISHIKAWA, Y. Nn/i: a new neural network which divides and learns environments. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1991, Singapore. *Proceedings...* Singapore: IEEE, 1991. v. 3, p. 684–687.

[50] BOLLIVIER, M. de; GALLINARI, P.; THIRIA, S. Cooperation of neural nets and task decomposition. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1991, Seatle. *Proceedings...* Seatle: IEEE, 1991. v. 2, p. 573–576.

[51] BAXT, W. Improving the accuracy of an artificial neural network using multiple differently trained networks. *Neural Computation*, v. 4, n. 5, p. 772–780, 1992.

[52] HUANG, J.; KUH, A. A neural network isolated word recognition system for moderate sized databases. In: INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, 1993, San Francisco. *Proceedings...* San Francisco: IEEE, 1993. v. 1, p. 387–391.

[53] ROGOVA, G. Combining the results of several neural networks classifiers. *Neural Networks*, v. 7, n. 5, p. 777–781, 1994.

[54] LINCOLN, W.; SKRZYPEK, J. Synergy of clustering back propagation networks. *Advances in neural information processing systems*, USA, California, v. 2, p. 650–657, 1990.

[55] HANSEN, L.; SALAMON, P. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 12, n. 10, p. 993–1003, 1990.

[56] HASHEM, S. Algorithms for optimal linear combinations of neural networks. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, 1997, Houston. *Proceedings...* Houston: IEEE, 1997. v. 1, p. 242–247.

[57] SONMEZ, M. et al. A hierarchical decision module based on multiple neural networks. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, 1997, Houston. *Proceedings...* Houston: IEEE, 1997. v. 1, p. 238–241.

[58] BATITI, R.; COLLA, A. Democracy in neural nets: voting schemes for classification. *Neural Networks*, v. 7, n. 4, p. 691–707, 1994.

[59] WATTA, P.; AKKAL, M.; HASSOUN, M. Decoupled-voting hamming associative memory networks. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, 1997, Houston. *Proceedings...* Houston: IEEE, 1997. v. 2, p. 1188–1193.

[60] JACOBS, R. A. et al. Adaptive mixtures of local experts. *Neural Computation*, USA, Cambridge, v. 3, n. 1, p. 79–87, 1991.

[61] GUAN, S. et al. Output partitioning of neural networks. *Neurocomputing*, v. 68, p. 38–53, 2005.

[62] COVER, T. M.; HART, P. E. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, USA, New Jersey, v. 13, n. 1, p. 21–27, 1967.

[63] WANG, J.; NESKOVIC, P.; COOPER, L. N. Improving nearest neighbor rule with a simple adaptive distance measure. *Pattern Recognition Letters*, v. 28, n. 2, p. 207–213, 2007.

[64] ROBINSON, T. *Dynamic error propagation networks*. Tese (Doutorado) — Cambridge University Engineering Department, 1989.

[65] ASSUNCION, A.; NEWMAN, D. J. *UCI*. 2012. Disponível em: <http://archive.ics.uci.edu/ml/>.

[66] FRIEDMAN, J. H. An overview of predictive learning and function approximation. *Statistics to Neural Networks*, p. 1–55, 1994.

[67] MONTGOMERY, D. C.; RUNGER, G. C. *Estatística aplicada e probabilidade para engenheiros*. 4. ed. [S.l.]: LTC, 2009.

[68] LILLIEFORS, H. W. On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, v. 62, n. 318, p. 399–402, 1967.

[69] VELLEMAN, P. F.; HOAGIN, D. C. *Applications, basics, and computing of exploratory data analysis*. [S.l.]: Duxbury Press, 1981.

[70] DITTENBACH, M.; MERKL, D.; RAUBER, A. The growing hierarchical self-organizing map. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2000, Italy. Italy: IEEE, 2000. p. 15–19.