UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Sobre o Processo de Seleção de Subconjuntos de Atributos – As Abordagens Filtro e *Wrapper*

Daniel Monegatto Santoro

Orientação:
Profa. Dra. Maria do Carmo Nicoletti

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, como parte dos requisitos exigidos para obtenção do grau de Mestre em Ciência da Computação

São Carlos
Abril/2005

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

S237sp

Santoro, Daniel Monegatto.
Sobre o processo de seleção de subconjuntos de atributos - as abordagens filtro e wrapper / Daniel Monegatto Santoro. -- São Carlos : UFSCar, 2005.
141 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2005.

1. Inteligência artificial. 2. Aprendizado do computador. 3. Métodos de busca. 4. Seleção de atributos. I. Título.

CDD: 006.3 (20ª)

AGRADECIMENTOS

Aos meus pais Celso e Lenize, ao meu irmão Diego
e à Ana Paula, por todo apoio e suporte necessários.

Aos meus avós Ernesto, Norma, Geraldo e Therezinha, por
estarem sempre comigo, assim como todos os demais familiares.

À minha namorada Mônica, que em pouco tempo tanto pôde me ensinar.

À minha orientadora Maria do Carmo Nicoletti, por todos os ensinamentos,
conselhos e conversas.

Aos meus grandes amigos de Arujá, Mogi das Cruzes e São Carlos, dos quais cito os mais próximos: Marcos e Dani, Gordo e Paula, Junji e Adriana, Vagner e Kaká, Wilber e Débora, Giba e Dani, Puga, Vini e Fabi, Rafa e Laís, Alex e Jú, Rodrigo e Soraia, Roger, Silvia, Paula, Fernanda, Juliana, Daphni, Beto e Fulvia, Sheila, Jefferson, João Marcos.

Àqueles que dividiram seus lares comigo em São Carlos, dos quais cito Junji,
Adriana, Picachu, Rodrigo, Jefferson, Sá, Fábio, Fred, Fernando e Fabiano.

Aos membros da banca examinadora Marley, Mauro e Estevam, por
contribuírem com este trabalho.

Aos professores e funcionários do Programa de Pós-Graduação
em Ciência da Computação da UFSCar.

Ao CNPq, pelo auxílio financeiro durante todo o
período de estudos.

A todos os outros que me ajudaram durante os
últimos dois anos.

“A vida só pode ser compreendida olhando-se para trás,
mas só pode ser vivida olhando-se para frente”
(Soren Kierkergaard)

RESUMO

Métodos indutivos de aprendizado de máquina aprendem a expressão do conceito a partir de um conjunto de treinamento. Conjuntos de treinamento são, na maioria das vezes, compostos por instâncias descritas por pares atributo-valor e uma classe associada. O conjunto de atributos usado para descrever as instâncias de treinamento tem um forte impacto na expressão induzida do conceito.

As técnicas para a seleção de subconjuntos de atributos no contexto de aprendizado de máquina objetivam identificar os atributos que efetivamente contribuem para a caracterização da classe de uma instância. Essas técnicas podem ser caracterizadas como do tipo *wrapper* (se estão associadas a um método específico de aprendizado de máquina) ou filtro e muitas delas funcionam articuladas a um método de busca (há ainda o tipo integrado, pouco representativo).

Este trabalho aborda o problema de seleção de subconjuntos de atributos por meio da investigação do desempenho de duas famílias de *wrappers* – a família NN (*Neareast Neighbor*) e a DistAl – e de três famílias de filtros – Relief, Focus e LVF. Os vários integrantes da família NN (bem como da família DistAl) diferem entre si com relação ao método de busca utilizado.

O trabalho apresenta e discute os experimentos realizados em vários domínios de conhecimento e seus resultados permitem uma avaliação comparativa de desempenho (precisão e dimensionalidade) dos elementos das várias famílias avaliadas.

ABSTRACT

Inductive machine learning methods learn the expression of the concept from a training set. Training sets are, generally, composed by instances described by attribute-value pairs and an associated class. The attribute set used for describing the training instances has a strong impact on the induced concepts.

In a machine learning environment, attribute subset selection techniques aim at the identification of the attributes which effectively contribute for establishing the class of an instance. These techniques can be characterized as wrappers (if they are associated with a specific machine learning method) or filter and many of them work in conjunction with a search method (there are also embedded feature selection methods, not very representative).

This work approaches the attribute subset selection problem by investigating the performance of two families of wrappers – the NN (Nearest Neighbor) and DistAl families – and three filter families – Relief, Focus and LVF. The many members of the NN family (as well as of the DistAl family) differ among themselves with relation to the search method they use.

The work presents and discusses the experiments conducted in many knowledge domains and their results allow a comparative evaluation (as far as accuracy and dimensionality are concerned) among the members of the families.

SUMÁRIO

1. INTRODUÇÃO 1
1.1 Considerações sobre Aprendizado 1
1.2 Aprendizado de Máquina 2
1.2.1 Caracterização e Principais Conceitos 3
1.2.2 O Processo de Avaliação da Precisão em Aprendizado de Máquina 6
1.3 Contextualização e Objetivos 9

2. SOBRE O PROCESSO DE SELEÇÃO DE SUBCONJUNTOS DE ATRIBUTOS 13
2.1 Considerações sobre o Processo de Seleção de Atributos 13
2.2 O Conceito de Atributo Relevante 17
2.2 Sobre a Categorização de Métodos de Seleção de Atributos 23
2.2.1 A Proposta Blum & Langley 23
2.2.2 A Proposta Dash & Liu 26
2.2.3 Conclusões sobre as Propostas de Categorização de Métodos de Seleção de Atributos 29
2.3 O Modelo *Wrapper* 30
2.4 O Modelo Filtro 33
2.5 Outras Considerações sobre a Seleção de Subconjuntos de Atributos 37

3. O MODELO *WRAPPER* DE SELEÇÃO DE ATRIBUTOS 39
3.1 Introdução 39
3.2 Estratégias de Direção da Busca – *Forward Selection*, *Backward Elimination* e *Random* 41
3.3 Considerações sobre os Métodos de Busca Utilizados 42
3.3.1 Hill-Climbing 43
3.3.2 Random Bit Climber 45
3.3.3 *Best-First* e *Beam Search* 47
3.3.4 Las Vegas 50
3.3.5 Algoritmo Genético 51
3.4 Experimentos e Resultados 52
3.4.1 Método de Avaliação e *Overfitting* 52
3.4.1.1 Bases de Dados Utilizadas nos Experimentos 55
3.4.2 A Família de *Wrappers* NN 57
3.4.2.1 Hill-Climbing 57
3.4.2.2 Random Bit Climber 60
3.4.2.3 Beam Search 62
3.4.2.4 Las Vegas Wrapper 64
3.4.2.5 Algoritmo Genético 66
3.4.2.6 Considerações sobre os *Wrappers* NN 66
3.4.3 A Família de *Wrappers* DistAl 67
3.4.3.1 Hill-Climbing 67
3.4.3.2 Random Bit Climber 69
3.4.3.3 Beam Search 71
3.4.3.4 Las Vegas 73
3.4.3.5 Algoritmo Genético 74
3.4.3.6 Considerações sobre os *Wrappers* DistAl 75
3.5 Considerações sobre os *Wrappers* 76

4. O MODELO FILTRO DE SELEÇÃO DE ATRIBUTOS 78
4.1 A Família Relief 78
4.1.1 O Algoritmo Relief 78
4.1.2 Um Exemplo de Uso do Relief 81
4.1.3 As Variantes Relief-A, Relief-B, Relief-C, Relief-D, Relief-E e Relief-F 83
4.1.4 Considerações sobre a Família Relief 85
4.1.5 Experimentos Realizados com o Relief 86
4.2 A Família Focus 91
4.2.1 O Algoritmo Focus 91
4.2.2 Focus-2 93
4.2.3 FocusD 94
4.2.4 Greedy Focus 98
4.2.5 C-Focus 99
4.2.6 Experimentos Realizados com o C-Focus e C-FocusD 101
4.3 A Família LVF 105
4.3.1 LVF 105
4.3.1.1 O Algoritmo LVF 106
4.3.2 C-LVF 107
4.3.3 Resultados obtidos com o C-LVF 108
4.3.4 LVI 110
4.3.4.1 O Algoritmo LVI e o C-LVI 111
4.3.5 Resultados obtidos com o C-LVI 112
4.4 Conclusões sobre os Filtros 115

5. CONCLUSÕES 117

REFERÊNCIAS BIBLIOGRÁFICAS 119

A. CONSIDERAÇÕES SOBRE OS ALGORITMOS GENÉTICOS 130
A.1 Introdução 130
A.2 Terminologia e Conceitos Básicos 131
A.3 Operadores Genéticos 134
A.4 Outras Considerações sobre os AGs 137

B. OS ALGORITMOS DE APRENDIZADO UTILIZADOS 138
B.1 Redes Neurais Construtivas e DistAl 138
B.2 Nearest Neighbor 140

LISTA DE FIGURAS VIII

LISTA DE TABELAS X

LISTA DE FIGURAS

Figura 1.1. Modelo para a avaliação do conceito induzido em um método de AM. 7

Figura 1.2. Esquema de avaliação de precisão do aprendizado utilizando 10-validação cruzada. 8

Figura 2.1. Representação de atributos com relevância fraca e forte [John et al. 1994] 22

Figura 2.2. Espaço de busca em um problema de seleção de atributos [Kohavi & John 1997] (os 0s representam atributos não selecionados e os 1s atributos selecionados)..... 25

Figura 2.3. Modelo do processo de seleção de um conjunto de atributos com validação [Dash & Liu 1997]. 27

Figura 2.4. Modelo de um método de seleção de subconjunto de atributos do tipo *wrapper* [John et al. 1994]. 31

Figura 2.5. Modelo de um método de seleção de subconjunto de atributos do tipo filtro, usado como pré-processamento a um método de AM. 34

Figura 2.6. *Checklist* para a utilização de um método de seleção de atributos. 38

Figura 3.1 As estratégias de direção da busca *Forward*, *Backward* e *Random*. 42

Figura 3.2. Algoritmo do método de busca *Hill-Climbing*, usado em um *wrapper*. 45

Figura 3.3. Dinâmica de funcionamento do algoritmo do *Hill-Climbing* com a estratégia *Forward*. 45

Figura 3.4. Algoritmo do método de busca *Random Bit Climber*. 47

Figura 3.5. Dinâmica de funcionamento do algoritmo do RBC com a estratégia *Forward*. 47

Figura 3.6. Algoritmo do método de busca *Best-First*. 48

Figura 3.7. Dinâmica de funcionamento do algoritmo *Beam Search*..... 49

Figura 3.8. Pseudo-código do método de busca Las Vegas..... 51

Figura 3.9. O esquema de avaliação utilizado na avaliação dos *wrappers*. 53

Figura 3.10. O esquema de avaliação utilizado pelos *wrappers*. 54

Figura 3.11. Conjunto de variantes dos *wrappers* integrados ao NN. 57

Figura 3.12. Conjunto de variantes dos *wrappers* integrados ao DistAl. 67

Figura 4.1. Pseudocódigo do algoritmo Relief. 81

Figura 4.2. Execução do Relief usando um conjunto de dados artificial..... 82

Figura 4.3. A família de algoritmos Relief e suas principais características. 86

Figura 4.4. Pseudo-código do algoritmo Focus. 92

Figura 4.5. Exemplo de instâncias e conflitos gerados com o Focus-2. 94

Figura 4.6. Subconjuntos de atributos em relação ao número de atributos. 95

Figura 4.7. Dinâmica de uma busca realizada pelo Focus original. 97

Figura 4.8. Dinâmica de uma busca realizada pelo FocusD. 97

Figura 4.9. Pseudo-código da busca realizada pelo FocusD. 98

Figura 4.10. O conceito de 'bastante próximo', utilizado pelo C-Focus. 100

Figura 4.11. Pseudo-código do algoritmo LVF. 107

Figura 4.12. Pseudo-código do algoritmo LVI. 112

Figura A.1. Algoritmo Genético canônico. 133

Figura A.2. Representação gráfica de uma 'roleta' na Seleção por Roleta. 134

Figura A.3. Exemplos de cruzamento para representação binária. 136

Figura B.1. A figura representa a instancia Z, a ser classificada pelo NN. A menor distância (d_1) para uma instância da classe A leva o algoritmo a atribuir a classe A para Z.....140

Figura B.2. Definição formal do algoritmo NN. 141

LISTA DE TABELAS

Tabela 1.1. Resumo de algumas possibilidades quanto à avaliação da precisão de um classificador. 9

Tabela 2.1. Conceito de paridade par. 18

Tabela 2.2. XOR correlacionado. 20

Tabela 2.3. Atributos relevantes e irrelevantes para cada definição (no XOR correlacionado). 21

Tabela 2.4. Comparação entre as funções de avaliação [Dash & Liu 1997]. 29

Tabela 2.5. Principais características das buscas exaustiva, seqüencial e randômica. 30

Tabela 3.1. Resumo das bases de dados utilizadas nos experimentos com *wrappers*. 55

Tabela 3.2 Desempenho do *Hill-Climbing* com o NN. 58

Tabela 3.3. Atributos selecionados e número de avaliações no *Hill-Climbing* com o NN. 59

Tabela 3.4. Desempenho do *Random Bit Climber* com o NN. 61

Tabela 3.5. Atributos selecionados e número de avaliações no *Random Bit Climber* com o NN. 61

Tabela 3.6. Desempenho do *Beam Search* com o NN. 63

Tabela 3.7. Atributos selecionados e número de avaliações no *Beam Search* com o NN. 64

Tabela 3.8. Desempenho do Las Vegas com o NN. 65

Tabela 3.9. Desempenho do Algoritmo Genético com o NN. 66

Tabela 3.10. Desempenho do Hill-Climbing com o DistAl. 68

Tabela 3.12. Desempenho do RBC com o DistAl. 70

Tabela 3.13. Atributos selecionados e número de avaliações no RBC com o DistAl. 71

Tabela 3.14. Desempenho do *Beam Search* com o DistAl. 72

Tabela 3.15. Atributos selecionados e número de avaliações no *Beam Search* com o DistAl. . 73

Tabela 3.16. Desempenho do Las Vegas com o DistAl. 74

Tabela 3.17. Desempenho do Algoritmo Genético com o DistAl. 75

Tabela 3.18. Tempo de processamento na seleção de atributos com o NN e DistAl (em segundos). 77

Tabela 4.1. Base de dados artificial. 81

Tabela 4.2. Desempenho com NN dos subconjuntos de atributos selecionados pelo Relief – 75%, 50% e 25% dos atributos. 87

Tabela 4.3. Número de atributos selecionados pelo Relief – 75%, 50% e 25% dos atributos... 88

Tabela 4.4. Desempenho com NN dos subconjuntos de atributos selecionados pelo Relief – Peso > (Maior/3), Peso > (Média), Peso > 0. 88

Tabela 4.5. Número de atributos selecionados pelo Relief – Peso > (Maior/3), Peso > (Média), Peso > 0. 89

Tabela 4.6. Desempenho com DistAl dos subconjuntos de atributos selecionados pelo Relief – 75%, 50% e 25% dos atributos.. 89

Tabela 4.7. Desempenho com DistAl dos subconjuntos de atributos selecionados pelo Relief – Peso > (Maior/3), Peso > (Média), Peso > 0. 90

Tabela 4.8. Atributos selecionados pelo C-Focus com diferentes graus de similaridade. 102

Tabela 4.9. Precisão com o NN dos subconjuntos de atributos selecionados pelo C-Focus.....103

Tabela 4.10. Precisão com o DistAl dos subconjuntos de atributos selecionados pelo C-Focus. 103

Tabela 4.11. Comparação do número de subconjuntos explorados pelos métodos C-Focus e C-FocusD (GSM = 100%).....104

Tabela 4.12. Atributos selecionados pelo C-LVF e o tempo de processamento influenciado pelo número instâncias da base de dados.....109

Tabela 4.13. Precisão com o NN dos subconjuntos de atributos selecionados pelo C-LVF.....109

Tabela 4.14. Precisão com o DistAl dos subconjuntos de atributos selecionados pelo C-LVF. 110

Tabela 4.15. Atributos selecionados pelo C-LVI, o tempo de processamento influenciado pelo número instâncias da base de dados e a porcentagem utilizada para análise. 114

Tabela 4.16. Precisão com o NN dos subconjuntos de atributos selecionados pelo C-LVI. 115

Tabela 4.17. Precisão com o DistAl dos subconjuntos de atributos selecionados pelo C-LVI. 115

1 CAPÍTULO

INTRODUÇÃO

Este capítulo apresenta os principais conceitos e idéias que subsidiam a área de Aprendizado de Máquina (AM), que é a área na qual esse trabalho se insere, e descreve o problema focalizado na pesquisa realizada, que é o da seleção de subconjuntos de atributos relevantes em conjuntos de treinamentos utilizados em AM.

1.1 Considerações sobre Aprendizado

Este trabalho investiga algumas técnicas que, entre outros resultados, colaboram para que uma máquina realize aprendizado automático de maneira mais eficiente (menor tempo, menor espaço de busca e, possivelmente, melhor precisão). Para uma melhor compreensão do que segue, o restante dessa seção identifica algumas tendências na caracterização de aprendizado como um processo inerente a organismos vivos para, só então, abordar a questão do aprendizado em sistemas computacionais, na Seção 1.2.

Na literatura as definições propostas para aprendizado são diversas, uma vez que é um conceito altamente complexo, que envolve processos e variáveis ainda não muito bem identificados e entendidos. Dada sua importância nas mais variadas áreas do conhecimento humano, aprendizado pode ser abordado e definido com diferentes enfoques tais como o comportamentalista, o cognitivista, o humanista ou o social, entre outros.

A corrente comportamentalista tem seu foco na mudança de comportamento a partir de um estímulo externo e, portanto, possui relação com os sistemas computacionais, o mesmo ocorrendo com a corrente cognitivista, que focaliza a mudança e estruturação dos processos internos da própria mente, de acordo com Smith, em [Smith 1999]. Já as duas outras correntes, isto é, a humanista e a social, estão mais

voltadas para fatores afetivos ou sociais e, por isso, ainda não são tão interessantes do ponto de vista computacional.

Em [Walker 1967] é proposta uma definição extremamente simples e direta, pertinente ao enfoque comportamentalista, que caracteriza aprendizado como uma mudança no desempenho como resultado da experiência. Esta definição destaca três aspectos relevantes, que são a alteração do desempenho em uma determinada tarefa, a necessidade de uma experiência propícia para esta alteração e o fato de ocorrer uma mudança e não necessariamente uma melhoria, o que deixa implícito que o desempenho pode até piorar após o aprendizado, fato que pode ser observado mesmo em sistemas computacionais.

Já a visão de aprendizado da corrente cognitivista tem seu foco voltado para outros aspectos. Segundo propõe [Hartley 1998] o aprendizado resulta de inferências, expectativas e na construção de conexões; ao invés de adquirir hábitos, um aprendiz adquire planos e estratégias e, para isso, um conhecimento anterior é importante. Esta definição cognitivista destaca a alteração do estado inicial do agente que aprende, por meio da elaboração de estratégias que servirão para situações futuras.

Com base nesses dois enfoques é possível concluir que aprendizado envolve a aquisição de conhecimentos que propiciam o desenvolvimento de novas habilidades e estratégias e que (possivelmente) resultam numa alteração do comportamento em eventos futuros.

Em qualquer situação de aprendizado, portanto, é de extrema relevância a maneira como a informação é fornecida a um organismo (ou sistema) bem como a maneira como esta informação será interpretada e/ou transformada pelo aprendiz. Este trabalho de pesquisa investiga métodos capazes de selecionar informações relevantes a serem utilizadas em sistemas computacionais, com foco em sistemas caracterizados como sistemas de aprendizado de máquina.

1.2 Aprendizado de Máquina

Um sistema computacional capaz de aprender precisa, primeiramente, ter a capacidade de adquirir e integrar novos conhecimentos, a fim de otimizar sua capacidade de operação, por meio da utilização de métodos focalizados em sua própria melhoria [Dietterich 1990].

A subárea da Inteligência Artificial (IA) chamada de Aprendizado de Máquina (AM) reúne algumas frentes de pesquisa que buscam, com enfoques diferentes, a melhor maneira de fazer um sistema computacional aprender. As redes neurais, o aprendizado baseado em instâncias, os algoritmos genéticos, os métodos de indução de regras e de árvores de decisão e a programação lógica indutiva são algumas das linhas de pesquisa mais importantes em AM (ver [Mitchell 1997], onde cada um desses modelos é abordado em suas principais características).

1.2.1 Caracterização e Principais Conceitos

A pesquisa em AM é focada basicamente no estudo de métodos computacionais que buscam melhorar a sua performance por meio da mecanização da aquisição de conhecimento a partir da experiência, segundo definição em [Langley & Simon 1995], o que reforça a caracterização de aprendizado apresentada inicialmente. Devido à própria constituição da área de AM, que integra várias áreas do conhecimento humano, na literatura podem ser encontradas diversas maneiras de abordar aprendizado que, nem sempre, são convergentes.

A maneira como os métodos e técnicas da área de AM estão organizados não possui um padrão bem definido e eles podem ser agrupados usando diferentes critérios. Pode-se, por exemplo, adotar como critério o método de inferência utilizado por eles, o que faz com que os métodos e técnicas de AM sejam classificados em *dedutivos* ou *indutivos*. Já se o critério utilizado for a existência (ou não) de conhecimento extra fornecido pelo ambiente, os métodos podem ser agrupados em *supervisionados* e *não-supervisionados*.

É importante ressaltar que os métodos de aprendizado nem sempre se encaixam completamente em uma destas classificações e/ou um dos grupos; esse é o caso, por exemplo, do aprendizado por analogia, que combina ambos os métodos de inferência, o dedutivo e o indutivo. Não pode ser esquecida, também, a tendência cada vez mais evidente na área de AM que é a da hibridização de diferentes métodos, por meio de sua integração e/ou colaboração, na busca por um modelo de aprendizado que combine as vantagens dos métodos e que minimize suas deficiências (ver, por exemplo [Borrajo & Veloso 1997]).

Os métodos de aprendizado dedutivos (ou analíticos) atuam somente nas

informações implícitas do conhecimento inicial [Briscoe & Caelli 1996] e, assim, não são capazes de generalizar o conhecimento a partir de eventos observados. Sistemas dedutivos são adequados a melhor compreender o conhecimento e não a expandi-lo. Os sistemas puramente dedutivos são parte de uma pesquisa restrita, cujos principais representantes são sistemas que se caracterizam por realizar Aprendizado Baseado em Explicação (ver [Carbonell et al. 1991]).

Já os métodos indutivos são mais abrangentes, no sentido de permitirem que um sistema computacional seja capaz de aprender de fato um conceito (ou seja, uma função capaz de associar uma dada entrada a uma determinada saída) e, por isso, são amplamente pesquisados. Os paradigmas de redes neurais, algoritmos genéticos, indução de regras e aprendizado baseado em instâncias são todos indutivos (embora este último possa ser classificado à parte) e representam alguns dos principais modelos de aprendizado de máquina existentes atualmente.

O aprendizado indutivo consiste, basicamente, num processo de generalização a partir de um conjunto de situações concretas em um determinado domínio de conhecimento (situações essas existentes no ambiente e fornecidas ao sistema) de maneira que, após a generalização ter sido feita pelo sistema, sua expressão pode ser usada para a caracterização de novas situações, no domínio em questão. Esse conjunto de situações fornecidas ao sistema como entrada é conhecido como *conjunto de treinamento* e cada uma das situações é chamada de *instância de treinamento* (ou *exemplo de treinamento*).

Geralmente cada instância no conjunto de treinamento é descrita por um conjunto de atributos e seus valores associados. Além disso, cada instância pode conter uma classe associada a elas, que indica uma classificação previamente feita por um especialista humano (ou não) na área de conhecimento em questão. Quando a classe à qual a instância pertence é informada, o aprendizado é caracterizado como supervisionado, caso contrário, como não-supervisionado.

Em problemas de aprendizado mais complexos, a utilização somente de informações sobre as instâncias pode não ser suficiente para a indução de uma expressão representativa do conceito. O uso da Teoria do Domínio é uma alternativa importante nestes casos, já que é integrado ao sistema um conhecimento prévio sobre o domínio do problema em questão (a chamada Teoria do Domínio), o que permite, em muitos casos, a construção do conceito pretendido [Nicoletti 1994].

É importante notar que a forma como são representadas as instâncias, o conceito induzido e também a Teoria do Domínio são especialmente relevantes e definem a abrangência do aprendizado.

A *linguagem de descrição de instâncias* é empregada para a descrição de instâncias do conjunto de treinamento e, geralmente, consiste de uma linguagem proposicional baseada na especificação de atributos e valores de atributos, e uma classe associada. A *linguagem de descrição de conceitos* é a linguagem empregada para a representação do conceito induzido e, via de regra, consiste na expressão proposicional de uma regra de decisão e suas variantes, tais como árvores de decisão.

Como comentado anteriormente, métodos que investem no uso de Teoria do Domínio são métodos que se propõem a lidar com o aprendizado de conceitos mais complexos que, conseqüentemente, exigem uma linguagem de representação mais sofisticada e poderosa para serem expressos. Esse é o caso da área de Programação Lógica Indutiva (PLI), que emprega um subconjunto da lógica de primeira ordem, aquele formado por cláusulas de Horn, para representar conjunto de treinamento, conceitos e Teoria do Domínio (ver [Nicoletti 1994] para uma discussão sobre esse assunto).

Um aspecto importante em AM é aquele relativo à maneira como os dados são apresentados ao método de aprendizado. É chamado de *aprendizado não-incremental* aquele aprendizado onde todo o conjunto de treinamento deve ser fornecido de uma só vez, já que o conceito é induzido utilizando todas as instâncias de treinamento simultaneamente. Um método de aprendizado não-incremental não é capaz de continuar aprendendo após o conceito ter sido induzido, usando novas instâncias de treinamento, a não ser que todo o processo indutivo seja realizado novamente. Já o aprendizado conhecido como *incremental* é construído exemplo a exemplo, conforme as instâncias são apresentadas ao método de aprendizado, o que permite um aprendizado contínuo de um determinado conceito, cuja representação vai sendo alterada à medida que novas instâncias são apresentadas.

Considerando ambas as maneiras de se fornecer as instâncias ao método de aprendizado, é possível observar que a maneira incremental é mais adequada para um aprendizado contínuo, já que é mais eficiente revisar uma hipótese existente do que refazer o aprendizado desde o início, cada vez que uma nova instância é observada [Utgoff 1989].

1.2.2 O Processo de Avaliação da Precisão em Aprendizado de Máquina

Em AM é bastante importante determinar quão representativa é a expressão do conceito induzido por algum método de aprendizado. A pesquisa e uso de métodos de aprendizado, portanto, sempre contemplam a avaliação do conceito induzido pelo método, de maneira a poder validar e referendar o seu uso em situações subseqüentes. Nesse contexto, por avaliação entende-se uma estimativa da caracterização correta (isto é, classe correta) de uma instância qualquer.

Tendo em vista a avaliação, as pesquisas na área de aprendizado geralmente adotam a seguinte estratégia: o conjunto de treinamento disponibilizado inicialmente é particionado em dois, um que efetivamente será usado como conjunto de treinamento para a indução da expressão do conceito e outro, geralmente chamado de *conjunto de teste*, que será usado para avaliar a expressão aprendida pelo método.

A forma mais simples de realizar esta tarefa é dividir o conjunto original em duas partes, uma para o conjunto de treinamento (usualmente uma quantidade equivalente à pelo menos 80% do conjunto de dados inicial, embora esse valor possa variar, dependendo da aplicação) e outra para determinar a precisão do conceito induzido nas classificações, contendo o restante das instâncias. Esta técnica é chamada de validação *holdout* ou então teste de estimativa simples [Reich & Barai 1999].

O conjunto de treinamento é usado para o aprendizado do conceito e, então, o conceito induzido é avaliado nas instâncias do conjunto de teste, como mostra o esquema da Figura 1.1. A avaliação simplesmente compara o resultado (classe) fornecido pela expressão induzida do conceito *versus* a classe que é parte da descrição da instância de teste. Em caso de igualdade, o conceito induzido fez uma classificação correta da instância, caso contrário, fez uma classificação incorreta.

A precisão de um conceito induzido por um determinado método de aprendizado é obtida dividindo-se o número de instâncias do conjunto de teste que foram classificadas corretamente pelo total de instâncias de tal conjunto.

É importante notar que, dependendo do domínio, alguns erros na classificação podem ser mais críticos que outros. Como apontado em [Dietterich 2003], na detecção de doenças, caso um paciente doente seja classificado erroneamente como saudável, tem-se um caso de *falso negativo*, o que caracteriza um erro inaceitável (dado que o paciente pode ter sua situação agravada com este erro). Já no caso de outro paciente que

esteja de fato saudável, se ele for classificado como doente, em uma situação caracterizada como *falso positivo*, sua situação não será tão prejudicial quanto a anterior, uma vez que as conseqüências deste tipo de erro, provavelmente, não serão fatais. Este aspecto é fundamental para considerar quais erros serão toleráveis e quais não.

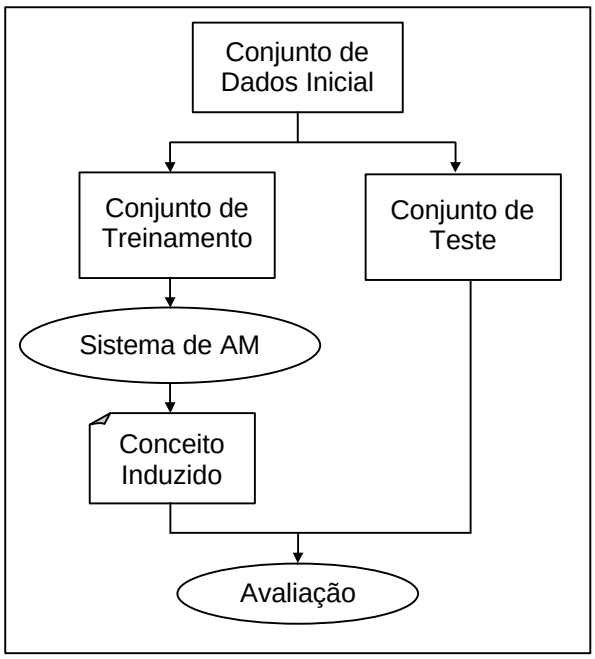


Figura 1.1. Modelo para a avaliação do conceito induzido em um método de AM.

Quanto à divisão dos dados originais em conjunto de treinamento e conjunto de teste, costuma-se não fazer apenas uma simples divisão, como explicado inicialmente na validação *holdout*, já que neste caso o resultado obtido pode não refletir a real precisão do conceito induzido dado um determinado conjunto de treinamento. Diz-se que este método não utiliza os dados eficientemente [Kohavi 1995], pois realiza o treinamento e o teste sobre apenas uma amostra, que pode, eventualmente, não ser significativa.

Uma forma bastante difundida de contornar este problema é utilizar uma estratégia conhecida por n-validação cruzada (ou K-validação cruzada), que se caracteriza por realizar n divisões no conjunto de dados inicial, gerando n subconjuntos de instâncias de tamanhos aproximadamente iguais. O processo exige que sejam feitos n treinamentos e os correspondentes testes; cada treinamento é realizado deixando um dos subconjuntos de instâncias separado para a realização do correspondente teste de precisão. O resultado final é calculado como a média aritmética dos n valores de precisão obtidos

(ver [Blum et al. 1999] para maiores detalhes).

Por exemplo, quando se tem um conjunto com 1000 instâncias e n = 10 (10-validação cruzada), formam-se 10 subconjuntos de 100 instâncias. São então realizados 10 experimentos, cada um deles envolvendo treinamento com 9 subconjuntos e teste com o subconjunto restante. Cada vez o treinamento é realizado com 900 instâncias e o teste correspondente realizado com as 100 instâncias que não participaram do treinamento. A precisão final é obtida fazendo a média aritmética dos valores de precisão obtidos (ver Figura 1.2).

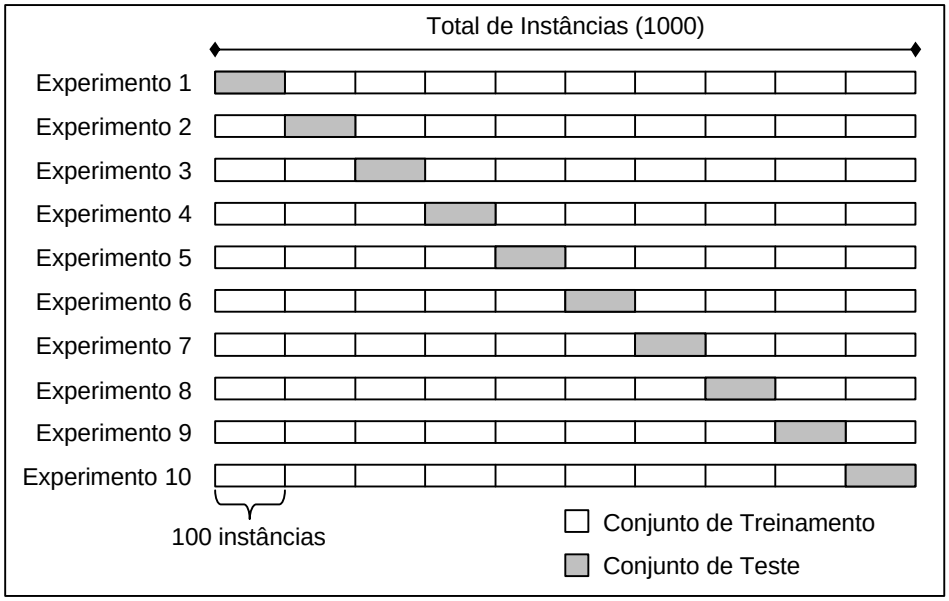


Figura 1.2. Esquema de avaliação de precisão do aprendizado utilizando 10-validação cruzada.

Valores mais altos de n normalmente garantem maior proximidade com a situação real. Quando n é igual ao total de instâncias presentes tem-se um caso particular chamado de *leave-one-out* [Kearns & Ron 1997], ou seja, para cada instância é realizado um treinamento com todas as outras instâncias e o teste de precisão é realizado apenas com a instância não incluída. No total são realizados tantos treinamentos/testes quantas forem as instâncias. O custo computacional dessa estratégia, entretanto, pode ser bastante alto e praticamente inviabiliza seu uso em um número significativamente grande de aplicações.

Neste trabalho foi adotada a 10-validação cruzada para a avaliação da precisão dos métodos de aprendizado utilizados, de forma a não exigir demais do sistema computacional e, também, não perder confiabilidade no resultado final obtido, conforme

sugerido em [Kohavi 1995], que realizou estudo bastante minucioso sobre os métodos de estimativa da precisão de classificadores e apontou a 10-validação cruzada como uma técnica que pode ser até melhor que o método *leave-one-out*.

Existem ainda outras formas de realizar esta separação a fim de avaliar a precisão do conceito induzido tais como a validação *bootstrap*, como descrita em [Efron & Tibshirani 1993] e também o *random subsampling cross-validation* [Kohavi 1995]. Um resumo das possibilidades mais relevantes para este trabalho quanto à avaliação da precisão de um classificador é apresentada na Tabela 1.1, baseada em [Jain et al. 2000].

Tabela 1.1. Resumo de algumas possibilidades quanto à avaliação da precisão de um classificador.

Método de validação	Descrição	Comentários
Holdout	Os dados são divididos em duas partes, uma para treinamento e outra para testes; conjunto de teste e treinamento são totalmente distintos.	Partições diferentes podem estimar a precisão do classificador diferentemente.
Leave-one-out	O conjunto de treinamento utiliza (n - 1) instâncias, com o teste sendo realizado na instância de fora; o processo é repetido n vezes para cada instância de teste que fica fora do conjunto de treinamento.	Muita variação entre os experimentos; alto custo computacional.
n-validação cruzada	Divide o total de instâncias em n subconjuntos, com quantidades similares de instâncias; utiliza (n - 1) subconjuntos para treinamento e o subconjunto restante para testes; o processo é repetido n vezes para cada subconjunto de teste que fica fora do conjunto de treinamento.	Uma solução intermediária entre os métodos *leave-one-out* e *holdout*.

Em algumas situações, como no problema de seleção de atributos, o conjunto de dados costuma ser dividido em três partes, uma delas sendo o conjunto de validação, além dos conjuntos de treinamento e teste. Este conjunto de validação auxilia na avaliação da qualidade final do subconjunto de atributos encontrado, enquanto os conjuntos de treinamento e teste auxiliam na busca por um bom subconjunto. Este conceito de validação é melhor explorado na Subseção 3.4.1.

1.3 Contextualização e Objetivos

Um aspecto crítico para um bom desempenho de qualquer técnica de aprendizado de máquina indutivo, sobretudo em situações reais, é aquele relativo aos dados a serem utilizados para alimentar o algoritmo de aprendizado.

O senso comum acredita que a utilização de uma maior quantidade de informação é mais vantajosa já que, idealmente, seria possível um aprendizado mais refinado. No entanto, tal situação muitas vezes não pode ser atendida, ou por limitações computacionais ou por deficiência na obtenção dos dados, ou ainda, pode não ser vantajosa, em casos onde os dados são redundantes ou são totalmente irrelevantes. Infelizmente em grande parte dos casos não é possível conhecer, antecipadamente, quais os dados necessários para a obtenção do melhor resultado [Dash & Liu 1997].

A questão do volume de dados a serem utilizados por um método de aprendizado pode ser interpretada como uma questão de dimensionalidade, considerando dois possíveis aspectos a serem tratados, a saber, o total de instâncias e o total de atributos.

Assim sendo, busca-se a redução de qualquer um desses aspectos, a fim de minimizar o volume de dados a serem tratados e aumentar a capacidade de generalização dos métodos de aprendizado devido à eliminação de dados irrelevantes ou redundantes.

Diversas técnicas de redução de dimensionalidade dos dados têm sido propostas e estudadas nos últimos anos, principalmente no contexto de aprendizado indutivo supervisionado [de Castro et al. 2004] [Dash & Liu 1997] [Langley 1994] [Liu & Motoda 1998]. Com a utilização destas técnicas, foi possível observar, entre outras melhorias, um aumento na eficiência do aprendizado bem como um aumento na compreensibilidade dos resultados aprendidos [Talavera 1999].

As abordagens para este problema são diversas, mas pode-se separá-las em dois grandes grupos, que se diferenciam basicamente com relação ao aspecto no qual investem:

- **Redução do número de atributos:** grupo de técnicas que focalizam a redução do número de atributos (ou selecionam os atributos relevantes) que descrevem as instâncias de treinamento; é o grupo que concentra a maior quantidade de técnicas que são referenciadas como *seleção de subconjuntos de atributos, seleção de atributos, seleção automática de atributos* ou *extração de atributos*;
- **Redução do número de instâncias:** grupo de técnicas que focalizam a redução do número de instâncias que irão participar do treinamento. Geralmente realizam amostragem seletiva das instâncias, evitando a utilização de instâncias que não acrescentam informações já obtidas a partir de outras instâncias já consideradas.

O principal objetivo deste trabalho de pesquisa é a investigação de métodos e técnicas que visam minimizar o conjunto de atributos no contexto de aprendizado indutivo de máquina. O trabalho focaliza e investiga métodos relativos às duas principais abordagens de seleção de atributos existentes na literatura, conhecidas como *wrapper* e filtro [Almuallim & Dietterich 1991] [Kira & Rendell 1992] [Yang & Honavar 1998].

A abordagem *wrapper* é sempre realizada articulada a um método de aprendizado enquanto que a abordagem filtro é sempre feita independentemente. Esta simples diferença torna as duas abordagens bastante distintas, tanto com relação à eficiência final do aprendizado, quanto com relação ao desempenho computacional, além de outros aspectos que serão considerados oportunamente.

Ambos os métodos são empregados com sucesso na tarefa de reduzir a quantidade de atributos necessários para o aprendizado de forma a manter ou mesmo aumentar a capacidade de generalização dos métodos de aprendizado. Tal sucesso é um fato esperado se for considerado que diversos algoritmos de aprendizado de máquina, como o C4.5 [Quinlan 1993] e o CN2 [Clark & Niblett 1989], sofrem uma certa degradação da precisão de predição quando atributos desnecessários fazem parte do conjunto de treinamento, como apontado em [Kohavi & Sommerfield 1995]. Nesta mesma referência, é comentado que alguns algoritmos, tal como o Naive-Bayes não sofrem significativamente com tal problema, mas acabam por perder eficiência quando atributos redundantes ou com relações entre eles estão presentes.

Outros objetivos deste trabalho são: o estudo do conceito de atributo relevante; a avaliação do impacto de diferentes métodos de busca nos métodos *wrapper* em relação à otimização na precisão de classificação utilizando algoritmos de AM, redução da dimensionalidade e eficiência computacional; avaliação da influência de diferentes estratégias de direcionamento na busca; e a avaliação de métodos filtro em relação à redução da dimensionalidade e manutenção da precisão de classificação.

Na literatura atualmente pode ser observado um grande esforço dedicado à otimização das diversas técnicas e algoritmos de aprendizado, bem como à integração das diversas propostas existentes [Dietterich 1997] [Ji & Ma 1999] [Saitta 2001] [Wurst et al. 2002]. Com o progresso e refinamento dos métodos de aprendizado, cada vez mais se torna perceptível o crescimento da área de AM [Mitchell 1997], possibilitando a aplicação de técnicas de AM em aplicações comerciais, com resultados bastante

satisfatórios em situações reais.

A seleção de atributos é uma técnica que, sem dúvida, vem contribuindo para um aumento na aplicação prática de métodos de AM, sobretudo em situações de aprendizado com grande número de atributos, tornando possível o aprendizado em situações antes impossíveis [Liu et al. 2002]. Em [Guyon & Elisseeff 2003] é reportado a utilização de métodos de seleção de atributos aplicados em situações de aprendizado com algumas centenas de atributos e até mesmo a dezenas de milhares de atributos.

Em todos os experimentos realizados o sistema utilizado foi um Pentium 4 2.8 GHz, com 512MB DDR400, em uma placa-mãe com chipset Intel i865PE. Os algoritmos foram todos implementados em C++ e compilados com o Borland C++ Builder 5.5. Estes dados são relevantes quando são constatados os tempos de processamento dos algoritmos.

Com o objetivo de investigar o processo de seleção de atributos e de algumas das várias técnicas que a operacionalizam, esse trabalho está organizado da seguinte maneira. O Capítulo 2 caracteriza e formaliza o conceito de atributo relevante conforme abordado na literatura, contextualiza detalhadamente o problema da redução do conjunto de atributos, discute as principais técnicas existentes na área e evidencia os principais métodos de busca existentes na literatura, em conexão com o problema de seleção de atributos.

O Capítulo 3 aborda métodos de seleção de atributos do tipo *wrapper* e investiga o impacto que o processo de busca e a estratégia de direcionamento, que são partes integrantes do método, desempenham. São avaliados experimentalmente cinco métodos de busca combinados com até três estratégias de direcionamento no contexto de seleção de atributos do tipo *wrapper*.

No Capítulo 4 são investigadas três famílias de métodos de seleção de atributos do tipo filtro com o objetivo de evidenciar a qualidade da seleção realizada por esses métodos, suas abrangências, limitações e potencialidades. São feitas avaliações e análises comparativas de desempenho dos métodos filtros *versus wrappers*.

O Capítulo 5 condensa os principais resultados obtidos com a pesquisa e identifica várias linhas de pesquisa como continuidade do trabalho realizado.

2 CAPÍTULO
SOBRE O PROCESSO DE SELEÇÃO DE SUBCONJUNTOS DE ATRIBUTOS

Este capítulo aborda os conceitos básicos relativos ao processo de seleção de atributos. São descritas as motivações e características dos principais modelos e métodos de seleção de atributos propostos na literatura, destacando-se os modelos *wrapper* e filtro. Situações onde a seleção de atributos deve ou não ser utilizada são consideradas e discutidas.

2.1 Considerações sobre o Processo de Seleção de Atributos

A seleção do conjunto de atributos é um problema que vem sendo estudado há algumas décadas, conforme comentado em [Aha & Bankert 1995], sobretudo na área de reconhecimento de padrões, onde pesquisadores têm atuado desde a década de 70 [Mucciardi & Gose 1971].

A pesquisa relacionada especificamente à seleção de atributos articulada a AM é relativamente mais recente e se intensificou a partir da década de 90 [Almuallim & Dietterich 1991] [Doak 1992] [John et al. 1994] [Kira & Rendell 1992].

Com a explosão no volume de dados armazenados e a consolidação da Mineração de Dados (MD) como área de pesquisa, a seleção de atributos se evidenciou também como um problema dessa área, conforme comentado em [Han et al. 1996]. Muitas das técnicas utilizadas em MD, entretanto, podem ser caracterizadas como técnicas de reconhecimento de padrões utilizando métodos estatísticos.

Apesar da pesquisa relativa à seleção de subconjuntos de atributos estar acontecendo associada a diferentes áreas de pesquisa, alguns conceitos e objetivos são compartilhados por elas, principalmente no que diz respeito à redução do volume de dados. Embora na área de reconhecimento de padrões a ênfase na redução do volume de

dados e o aumento da precisão do classificador sejam os únicos resultados esperados [Portinale & Saitta 2002], em AM outros fatores tais como a compreensibilidade do conceito aprendido e aumento da robustez do método de aprendizado são também relevantes [Talavera 1999].

Uma importante consideração feita em [John et al. 1994] é a de que boa parte da pesquisa realizada em reconhecimento de padrões assume como válido o princípio da monotonicidade, que estabelece que o aumento do conjunto de atributos só melhora a performance de um classificador, o que nem sempre acontece em situações práticas de aprendizado de máquina. Este fato é discutido em [Portinale & Saitta 2002], também no contexto de AM, e tais autores comentam que "...acurácia geralmente é não monotônica com relação à adição de atributos, e mesmo quando o algoritmo de aprendizado é robusto com relação a atributos irrelevantes (como por exemplo a abordagem Naive-Bayes [Mitchell 1997]) é, usualmente, significativamente afetado pela correlação entre atributos."

Muito embora sejam intuitivamente semelhantes em muitos aspectos, as definições formais para o problema da seleção de atributos são diversas e dependem do enfoque adotado. Dash & Liu em [Dash & Liu 1997] apresentam quatro diferentes definições para a seleção de atributos, identificadas pelos autores como conceitualmente diferentes.

1) **Idealizada:** o objetivo é encontrar o menor subconjunto de atributos necessário e suficiente para descrever o conceito alvo [Kira & Rendell 1992];
2) **Clássica:** o objetivo é selecionar um subconjunto de M atributos a partir de um conjunto de N atributos, com $|M| < |N|$, de forma que M é o melhor subconjunto dentre todos os possíveis subconjuntos com tamanho $|M|$, de acordo com uma função de avaliação [Narendra & Fukunaga 1977];
3) **Melhora da taxa de precisão:** o objetivo é selecionar um subconjunto de atributos de forma a aumentar a precisão da classificação em relação ao conjunto de todos os atributos ou, então, reduzir o conjunto de atributos inicial sem diminuir significativamente a taxa de precisão de um método de aprendizado em um determinado domínio [Koller & Sahami 1996];
4) **Aproximação da distribuição original das classes:** o objetivo é selecionar um subconjunto pequeno de atributos de forma que a distribuição das classes utilizando apenas estes atributos seja a mais próxima possível da distribuição utilizando todos os atributos [Koller & Sahami 1996].

É importante notar também que a seleção de atributos pode ser vista como uma especialização dos métodos de atribuição de pesos aos atributos, uma área de pesquisa também bastante ativa [Wettschereck et al. 1997], onde os atributos recebem pesos proporcionais à sua importância na representação do conceito. O processo de seleção de atributos é um caso particular do processo de atribuição de pesos a atributos, onde atributos relevantes e não relevantes estão associados a pesos 1 e 0 respectivamente.

Grande parte da pesquisa relacionada à seleção de atributos tem como objetivo a melhoria de um método de AM quanto à sua taxa de precisão. No contexto de AM, entretanto, existem outros fatores a serem considerados. Em [Michie et al. 1994], são relacionadas quatro características associadas a métodos de AM, sendo que três delas podem ser diretamente otimizadas com a utilização de métodos de seleção do conjunto de atributos:

1) **Precisão:** a capacidade de classificação de instâncias não observadas é uma das principais motivações do estudo de técnicas para a seleção de atributos em AM. Quando se realiza a seleção de atributos, se espera que atributos irrelevantes e redundantes sejam removidos, o que normalmente leva a um aumento da precisão do método de aprendizado;
2) **Velocidade na classificação:** uma maior rapidez na classificação de novas instâncias pode ser determinante em muitos casos. Esta característica é amplamente otimizada com as técnicas de seleção de atributos, pois o conceito induzido tende a ser menos complexo após a redução da quantidade de atributos. A seleção de atributos leva à redução do volume de dados, com a conseqüente redução da carga computacional exigida;
3) **Tempo de aprendizado:** a velocidade do aprendizado é importante em casos onde o ambiente de aprendizado muda constantemente; esta característica nem sempre é otimizada, já que um tempo adicional deve ser levado em consideração, relativo à redução do número de atributos. Por outro lado, entretanto, em qualquer situação, quanto maior o número de atributos que descrevem as instâncias de treinamento, maior o tempo de aprendizado do conceito representado por elas (por isso, os filtros podem eventualmente melhorar o tempo de aprendizado, já os *wrappers* não, devido à metodologia adotada neste, que envolve a utilização de um método de aprendizado);

4) **Compreensibilidade:** em muitos domínios de conhecimento é fundamental que a expressão do conceito induzido possa ser entendida pelo usuário. A compreensibilidade é normalmente melhorada com a redução dos atributos e a conseqüente simplificação do conceito induzido.

O processo de seleção de atributos é particularmente importante em situações em que instâncias são descritas usando um número grande de atributos e se desconhece a relevância de cada um desses atributos na representação do conceito.

Métodos de seleção podem evitar/minimizar o problema conhecido como 'maldição da dimensionalidade' em muitas situações de aprendizado [Koller & Sahami 1996]. Por esta expressão entende-se a crescente complexidade de se encontrar uma boa modelagem para um conceito devido à expansão no espaço de busca provocada pela presença de mais atributos.

Com a redução do número de atributos realizada por um método de seleção de atributos, o sistema de aprendizado fica menos sobrecarregado, apesar da inclusão de um outro método ao processo, aquele que faz a seleção dos atributos relevantes. Com a redução do número de atributos, além da redução da sobrecarga computacional, o conceito induzido torna-se mais compreensível, sobretudo quando utilizados métodos simbólicos de AM (por exemplo, árvores de decisão).

A redução de atributos pode colaborar, também, com a eliminação dos processos associados à obtenção dos valores dos atributos julgados desnecessários (irrelevantes ou redundantes). Isso significa, por exemplo, eliminar sensores ou exames médicos, implicando a redução de custos. Da mesma forma, em situações onde os dados devem ser armazenados, a redução de atributos (e também de instâncias) implica redução em espaço de armazenamento.

Por outro lado, não deve ser esquecido que a redução no número de atributos pode, eventualmente, implicar perda no poder de discriminação do classificador e, conseqüentemente, uma menor precisão do sistema.

Em geral a seleção de atributos colabora com a otimização da capacidade de generalização do método de aprendizado, já que são selecionados apenas os atributos relevantes, eliminando-se aqueles que não contribuem ou que atrapalham o aprendizado.

2.2 O Conceito de Atributo Relevante

O conceito de atributo relevante pode ter diversas interpretações. Para a identificação de atributos relevantes é fundamental, primeiro, que se estabeleça o conceito de atributo relevante e, também, se defina relevância do atributo em relação a que.

Um levantamento bibliográfico sobre definições de atributo relevante realizado e descrito em [John et al. 1994], evidenciou algumas definições na literatura. Esses autores notaram, entretanto, que para um problema simples (XOR correlacionado), os atributos caracterizados como relevantes usando cada uma das definições variavam consideravelmente. Este fato motivou os autores a redefinirem o conceito de atributo relevante, propondo dois graus de relevância, identificados como fraco e forte.

No que segue, as definições de relevância existentes na literatura são apresentadas, é mostrada a aplicação das definições de relevância em dois exemplos, o de paridade par e de XOR correlacionado, e são apresentados e discutidos os conceitos de relevância fraca e forte. Para tal, as seguintes notações, convenções e suposições são adotadas.

- A entrada para um algoritmo de aprendizado supervisionado é um conjunto de m instâncias de treinamento;
- Cada instância de treinamento $X \in F_1 \times F_2 \times ... \times F_n$, onde F_i $(1 \leq i \leq n)$ é o domínio de valores do atributo X_i;
- Instâncias de treinamento são pares $<X,y>$, onde $y \in Y$ é um rótulo (ou saída) (isto é, a classe da instância X) e Y é o conjunto dos possíveis rótulos que descrevem as instâncias;
- Dada uma instância X, o valor do atributo X_i em X é denotado por x_i;
- A tarefa de um algoritmo indutivo de AM é induzir uma estrutura (por exemplo, árvore de decisão) tal que, dada uma nova instância, é possível prever sua classe y;
- É adotada uma medida de probabilidade p no espaço $F_1 \times F_2 \times ... \times F_n \times Y$.

Na discussão que segue não são feitas suposições sobre os tipos de atributos ou classes.

Definição 1 [Almuallim & Dietterich 1991]. Um atributo X_i é relevante para um conceito C se X_i comparece em toda fórmula booleana que representa C e é irrelevante caso contrário.

A Definição 1 caracteriza relevância supondo que todos os atributos e classes são booleanos e que não existe ruído.

Definição 2 [Gennari et al. 1989]. Um atributo X_i é relevante se e somente se existe algum x_i e y para os quais $p(X_i = x_i) > 0$ tal que

$$p(Y = y \mid X_i = x_i) \neq p(Y = y)$$

A Definição 2 estabelece que um atributo X_i é relevante se o conhecimento de seu valor pode mudar a estimativa para a classe Y, ou seja, a classe é condicionalmente dependente de X_i. Note que a Definição 2 falha na identificação de atributos relevantes no conceito de paridade par, como mostrado no Exemplo 2.1.

Exemplo 2.1.
Considere o conceito de paridade par definido por três atributos booleanos X_1, X_2 e X_3 representado na Tabela 2.1.

Tabela 2.1. Conceito de paridade par.

X_1	X_2	X_3	Classe
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

De acordo com a Definição 2, nenhum atributo é relevante para a caracterização do conceito paridade par, uma vez que $p(Y = y \mid X_i = x_i) \neq p(Y = y)$ não acontece, como mostram os cálculos a seguir:

$$p(Y = 0) = 4/8 = 1/2$$
$$p(Y = 1) = 4/8 = 1/2$$

$p(Y = 0 \mid X_1 = 0) = 2/4 = 1/2$
$p(Y = 1 \mid X_1 = 0) = 2/4 = 1/2$
$p(Y = 0 \mid X_2 = 0) = 2/4 = 1/2$
$p(Y = 1 \mid X_2 = 0) = 2/4 = 1/2$
$p(Y = 0 \mid X_3 = 0) = 2/4 = 1/2$
$p(Y = 1 \mid X_3 = 0) = 2/4 = 1/2$

$p(Y = 0 \mid X_1 = 1) = 2/4 = 1/2$
$p(Y = 1 \mid X_1 = 1) = 2/4 = 1/2$
$p(Y = 0 \mid X_2 = 1) = 2/4 = 1/2$
$p(Y = 1 \mid X_2 = 1) = 2/4 = 1/2$
$p(Y = 0 \mid X_3 = 1) = 2/4 = 1/2$
$p(Y = 1 \mid X_3 = 1) = 2/4 = 1/2$

Para contornar este problema, a Definição 2 foi alterada, com a inclusão de duas novas suposições:

- Seja S_i o conjunto de todos os atributos exceto X_i, ou seja, $S_i = \{X_1, X_2, ..., X_{i-1}, X_{i+1}, ..., X_m\}$;
- Seja s_i uma atribuição de valores a todos os atributos em S_i.

Definição 3. Um atributo X_i é relevante se e somente se existe algum x_i, y e s_i para os quais $p(X_i = x_i) > 0$ tal que

$$p(Y = y, S_i = s_i \mid X_i = x_i) \neq p(Y = y, S_i = s_i)$$

A Definição 3 também falha na identificação dos atributos relevantes, identificando todos os atributos como relevantes no conceito de paridade par, como mostra o Exemplo 2.2.

Exemplo 2.2

De acordo com a Definição 3, todos atributos são relevantes para a caracterização do conceito paridade par, uma vez que $p(Y = y, S_i = s_i \mid X_i = x_i) \neq p(Y = y, S_i = s_i)$ é verificado em todas as situações, como mostram os cálculos a seguir:

$$p(Y=0, X_2 = 0, X_3 = 0) = 1/2$$
$$p(Y=1, X_2 = 0, X_3 = 0) = 1/2$$

$$p(Y=0, X_2 = 0, X_3 = 0 \mid X_1=0) = 1 \qquad p(Y=0, X_2 = 0, X_3 = 0 \mid X_1=1) = 0$$
$$p(Y=1, X_2 = 0, X_3 = 0 \mid X_1=0) = 0 \qquad p(Y=1, X_2 = 0, X_3 = 0 \mid X_1=1) = 1$$

$$p(Y=0, X_1 = 0, X_3 = 0) = 1/2$$
$$p(Y=1, X_1 = 0, X_3 = 0) = 1/2$$

$$p(Y=0, X_1 = 0, X_3 = 0 \mid X_2 = 0) = 1 \qquad p(Y=0, X_1 = 0, X_3 = 0 \mid X_2 = 1) = 0$$
$$p(Y=1, X_1 = 0, X_3 = 0 \mid X_2 = 0) = 0 \qquad p(Y=1, X_1 = 0, X_3 = 0 \mid X_2 = 1) = 1$$

$$p(Y=0, X_1 = 0, X_2 = 0) = 1/2$$
$$p(Y=1, X_1 = 0, X_2 = 0) = 1/2$$

$$p(Y=0, X_1 = 0, X_2 = 0 \mid X_3 = 0) = 1 \qquad p(Y=0, X_1 = 0, X_2 = 0 \mid X_3 = 1) = 0$$
$$p(Y=1, X_1 = 0, X_2 = 0 \mid X_3 = 0) = 0 \qquad p(Y=1, X_1 = 0, X_2 = 0 \mid X_3 = 1) = 1$$
$$\ldots \qquad \ldots$$

Definição 4. Um atributo X_i é relevante se e somente se existe algum x_i, y e s_i para os quais $p(X_i = x_i, S_i = s_i) > 0$ tal que

$$p(Y = y \mid X_i = x_i, S_i = s_i) \neq p(Y = y \mid S_i = s_i)$$

De acordo com a Definição 4, o atributo X_i é relevante se a probabilidade da classe (dados todos atributos) pode mudar quando o valor de X_i não é considerado.

Conforme apontado em [John et al. 1994], para o exemplo do XOR correlacionado (mostrado no Exemplo 2.3), todas as definições anteriores dão resultados inesperados.

Exemplo 2.3.
Sejam X_1, X_2, X_3, X_4 e X_5 atributos booleanos. O espaço de instâncias é tal que X_2 e X_3 são negativamente correlacionados com X_4 e X_5 respectivamente, ou seja, $X_4 = \overline{X}_2$ e $X_5 = \overline{X}_3$. Existem apenas oito instâncias possíveis nesse espaço e assume-se que elas são equiprováveis. Considere que o conceito alvo é dado por:

$$Y = X_1 \oplus X_2 \ ^{1}$$

Note que o conceito alvo tem uma expressão booleana equivalente dada por $Y = X_1 \oplus \overline{X}_4$. A Tabela 2.2 mostra as oito possíveis instâncias do conjunto de instâncias e o valor de classe associado.

Para a expressão do conceito, os atributos X_3 e X_5 são completamente irrelevantes já que não participam da expressão que define o conceito alvo. O atributo X_1 é indispensável e um dos atributos X_2 ou X_4 pode ser descartado. Quando um deles é descartado o outro se torna essencial.

Tabela 2.2. XOR correlacionado.

X_1	X_2	X_3	X_4	X_5	$X_1 \oplus X_2$ ou $X_1 \oplus \overline{X}_4$
0	0	0	1	1	0
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	0	0	0

A Tabela 2.3 mostra, para o conceito em questão e para cada uma das definições, quais são os atributos considerados relevantes.

[1] O símbolo ⊕ denota a operação XOR (ou exclusivo).

Tabela 2.3. Atributos relevantes e irrelevantes para cada definição (no XOR correlacionado)

Definição	Relevante	Irrelevante
Definição 1	X_1	X_2, X_3, X_4, X_5
Definição 2	Nenhum	Todos
Definição 3	Todos	Nenhum
Definição 4	X_1	X_2, X_3, X_4, X_5

Considerando a Definição 4, o único atributo relevante é o X_1, já que:

$p(Y = 0 \mid X_1 = 0, X_2 = 0, X_3 = 0, X_4 = 1, X_5 = 1) \neq p(Y = 0 \mid X_2 = 0, X_3 = 0, X_4 = 1, X_5 = 1)$

P(...) = 1 P(...) = 0,5

$p(Y = 1 \mid X_1 = 0, X_2 = 0, X_3 = 0, X_4 = 1, X_5 = 1) \neq p(Y = 1 \mid X_2 = 0, X_3 = 0, X_4 = 1, X_5 = 1)$

P(...) = 0 P(...) = 0,5

Os autores John, Kohavi e Pfleger em [John et al. 1994] cogitam que são necessários dois graus de relevância. A Definição 4 estabelece a relevância forte. Relevância forte implica o atributo ser indispensável; não pode ser removido sem que haja perda na acurácia da predição do conceito.

Definição 5 (Relevância fraca). Um atributo X_i é fracamente relevante se e somente se não for fortemente relevante e se existe um subconjunto de atributos $S'_i \leq S_i$ para o qual existe algum x_i, y e s'_i com $p(X_i = x_i, S'_i = s'_i) > 0$ tal que

$$p(Y = y \mid X_i = x_i, S'_i = s'_i) \neq p(Y = y \mid S'_i = s'_i)$$

Relevância fraca implica o atributo poder algumas vezes contribuir para a acurácia na predição. Atributos são relevantes se são fracamente ou fortemente relevantes e são irrelevantes caso contrário. Atributos irrelevantes nunca contribuem para a acurácia na predição.

Para o Exemplo 2.3, então, atributo X_1 é fortemente relevante; atributos X_2 e X_4 são fracamente relevantes e X_3 e X_5 são irrelevantes.

De acordo com [Blum & Langley 1997], as noções de relevância forte (Definição 4) e relevância fraca (Definição 5) podem ser caracterizadas como relevância em relação à uma distribuição a qual pode ser interpretada como a noção de 'relevância com relação à amostra'.

Como comentado em [John et al. 1994], algoritmos tais como o Focus, proposto em [Almuallim & Dietterich 1991] e tratado em detalhes no Capítulo 4 dessa dissertação, encontram um conjunto mínimo de atributos que são suficientes para caracterizar o conceito. Dado um volume suficiente de instâncias de treinamento, esses algoritmos irão selecionar todos os atributos fortemente relevantes, nenhum atributo irrelevante e apenas um pequeno conjunto de atributos fracamente relevantes que são suficientes para caracterizar o conceito. Algoritmos tais como o Relief [Kira & Rendell 1992] [Kononenko 1994], tratado em detalhes no Capítulo 4 dessa dissertação, tentam aproximar eficientemente o conjunto de atributos relevantes.

A Figura 2.1 mostra um diagrama extraído de [John et al. 1994] no qual os autores procuram representar o conjunto de atributos que os métodos Focus e Relief buscam aproximar.

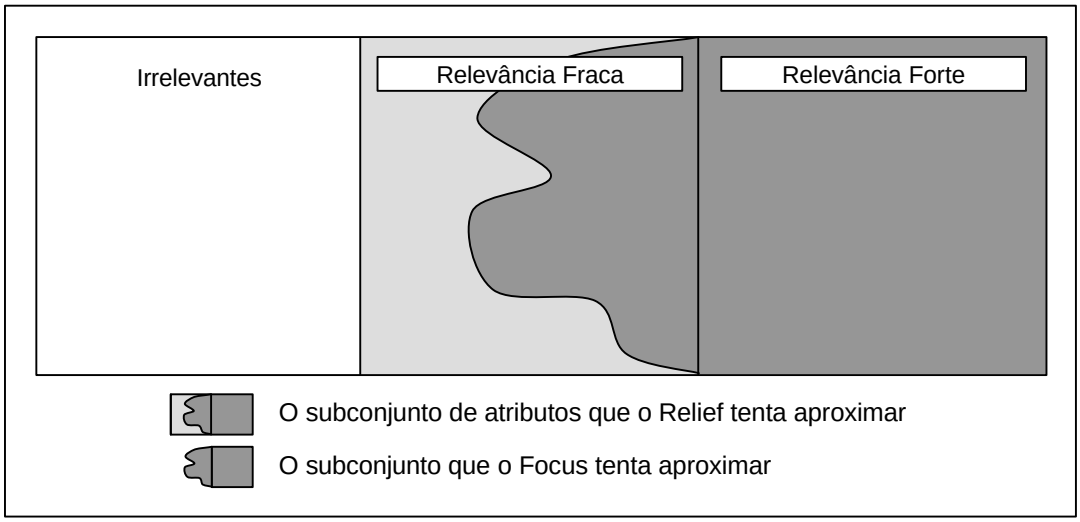


Figura 2.1. Representação de atributos com relevância fraca e forte [John et al. 1994]

É importante notar que as definições de relevância de atributos não implicam obrigatoriamente que todo processo de seleção de atributos que busca a melhor precisão na classificação deverá conter apenas os atributos considerados relevantes e desconsiderar todos os atributos irrelevantes. Em alguns casos os atributos não relevantes incluem um *bias* que auxilia na precisão da predição (ver [Kohavi & John 1997]).

2.2 Sobre a Categorização de Métodos de Seleção de Atributos

Como comentado anteriormente, na literatura existe uma grande diversidade de métodos para a seleção de atributos. Por isso, é fundamental que haja uma taxonomia para organizá-los de maneira sistemática, facilitando assim a caracterização de cada um deles, bem como a análise das diferenças e similaridades entre eles.

Na literatura existem algumas propostas de agrupamento de métodos/técnicas para seleção de atributos que são, via de regra, subsidiadas por diferentes critérios. Duas delas são de interesse nesse trabalho devido, principalmente, à sua abrangência. A proposta descrita em [Blum & Langley 1997] caracteriza o processo de seleção de atributos como uma busca heurística e a proposta descrita em [Dash & Liu 1997] diferencia os métodos/técnicas de seleção de atributos usando como critério tanto a maneira como os subconjuntos de atributos são avaliados quanto o tipo de busca realizada. Essas duas propostas de classificação de métodos de seleção de atributos são resumidas nas duas subseções seguintes.

2.2.1 A Proposta Blum & Langley

Em [Blum & Langley 1997] o processo de seleção de atributos é visto como um processo de busca heurística. A proposta (referenciada como proposta Blum & Langley) parte do princípio que existem vários estados no espaço de estados definido por variações de um conjunto de atributos. O método de seleção de atributos deve ser capaz de encontrar o estado (subconjunto de atributos) que melhor satisfaz o objetivo proposto (por exemplo, maximizar a precisão do classificador). O processo de busca pode então ser caracterizado em quatro aspectos:

1) **Estado inicial**: referente ao(s) subconjunto(s) de atributos a partir do(s) qual(is) a busca vai iniciar; implica também a definição dos operadores usados para gerar os próximos estados. Três modos são cogitados:
   1.1) **Todos os atributos**: iniciar a busca a partir do estado representado por todos os atributos. A busca então os remove sucessivamente, em uma estratégia conhecida *backward elimination*;
   1.2) **Nenhum atributo**: iniciar a busca a partir do estado representado por nenhum atributo. A busca os adiciona progressivamente, em uma estratégia conhecida como *forward selection*;

1.3) **Subconjunto de atributos randômico:** iniciar a busca a partir do estado representado por atributos selecionados randomicamente, variando a cada execução. Um exemplo pode ser encontrado em [Dvijver & Kittler 1982] onde é proposto um operador de busca que adiciona k atributos e retira um. A busca pode também ser conduzida por um algoritmo genético que, via operadores de cruzamento e mutação, induz outros tipos de conectividade no espaço de estados;

2) **Organização da busca**: caracteriza a forma como a busca é organizada. São cogitados dois modos básicos e algumas variantes:

2.1) **Exaustiva**: é a alternativa mais simples que verifica exaustivamente todos os estados possíveis. Não é uma alternativa viável em muitos casos, já que o tamanho do espaço de busca é de 2^n estados, n sendo o número total de atributos. Na Figura 2.2 é mostrado todo o espaço de busca para um conjunto com apenas quatro atributos;

2.2) ***Greedy***: é uma maneira mais viável de percorrer o espaço de busca em situações reais. A cada estado são consideradas as possíveis mudanças locais ao estado, uma é selecionada e, então, uma nova iteração é considerada. Um método de busca como o *Hill-Climbing* pode ser empregado;

3) **Estratégia de avaliação do subconjunto de atributos**: diz respeito à maneira como os subconjuntos de atributos são avaliados. Uma medida comumente usada envolve medir a habilidade do atributo em discriminar entre as classes que ocorrem no conjunto de treinamento. Essas estratégias podem ser divididas em três grandes grupos que se diferenciam pela maneira como a estratégia de seleção de atributos interage com o método de AM básico (utilizado para a indução do conceito):

3.1) **Filtro**: a relevância dos atributos é avaliada independentemente – não envolve o uso de um método de AM;

3.2) ***Wrapper***: avalia o subconjunto de atributos articulado ao método de AM que realizará a indução do conceito;

3.3) **Integrada**: caracterizam processos de seleção de atributos que são parte integrante do próprio método de AM. Estes métodos de AM são aqueles que, de alguma maneira, dão preferência a alguns dos atributos em detrimento de outros. Podem ser citados o ID3 [Quinlan 1986], C4.5 [Quinlan 1993], CART [Breiman et al. 1984] ou até mesmo as redes neurais artificiais, embora neste último caso a abordagem seja mais ampla, como num algoritmo que atribui pesos aos atributos;

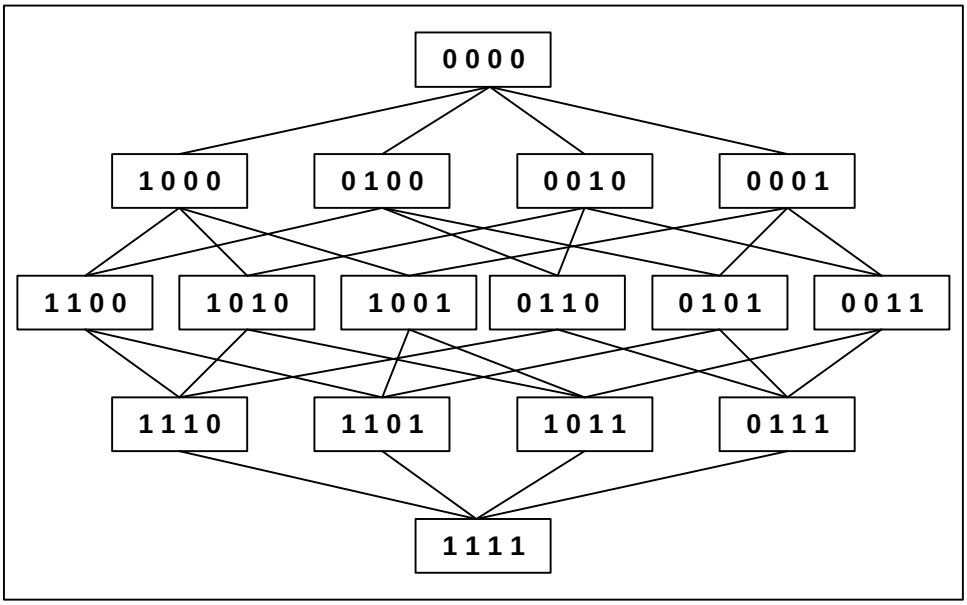


Figura 2.2. Espaço de busca em um problema de seleção de atributos [Kohavi & John 1997] (os 0s representam atributos não selecionados e os 1s atributos selecionados).

4) **Critério de parada**: a busca pelo subconjunto de atributos pode ser interrompida de acordo com alguns critérios pré-estabelecidos:

4.1) **Quantidade de atributos:** após a identificação do melhor subconjunto contendo um número pré-determinado atributos;

4.2) **Não há melhora na precisão da classificação:** quando a inserção de um novo atributo não resulta em nenhuma melhoria na precisão da classificação (quando for utilizada uma estratégia de busca heurística como o *forward selection*, por exemplo);

4.3) **Há degradação na precisão da classificação:** quando a inserção ou remoção de um atributo degrada a precisão da classificação;

4.4) **Seleção do melhor, após busca completa:** a busca é realizada sobre todos os possíveis subconjuntos de atributos para, ao final ser selecionado o melhor subconjunto**.**

O modelo integrado está em um nível diferente dos outros dois modelos considerados, o filtro e o *wrapper* e, por isso, não há um estudo específico em relação a eles, já que são algoritmos de aprendizado com características especiais.

É interessante ressaltar que mesmo os algoritmos de aprendizado que incluem alguma forma de seleção de atributos podem ser beneficiados com o uso das abordagens *wrapper* ou filtro, algo observado em diversos estudos, tais como os que otimizam o aprendizado de árvores de decisão [Almuallim & Dietterich 1991] ou de redes neurais [Yang & Honavar 1998].

2.2.2 A Proposta Dash & Liu

A revisão dos métodos de seleção de atributos descrita em [Dash & Liu 1997] (referenciada como proposta Dash & Liu) é mais abrangente e informativa que a de Blum & Langley mas, em contrapartida, não aborda aspectos teóricos em profundidade. Embora ambas tenham algumas similaridades, a proposta Dash & Liu não trata o processo de seleção de atributos apenas como uma busca; ela incorpora alguns elementos específicos da própria seleção de atributos, o que permite uma análise mais minuciosa em algumas situações.

A proposta Dash & Liu caracteriza os métodos via quatro aspectos fundamentais descritos a seguir e mostrados no diagrama da Figura 2.3.

1) **Procedimento de geração dos subconjuntos de atributos**: o procedimento de geração é o aspecto que engloba o método de busca e o estado inicial da busca. Os métodos de seleção devem seguir uma estratégia específica para a geração de novos subconjuntos de atributos. Para isso é necessário que seja definido um estado inicial para o conjunto de atributos. Este pode ser iniciado com todos os atributos, com nenhum atributo ou com atributos selecionados randomicamente. O procedimento de geração também define a forma como acontece a busca pelo conjunto de atributos, inserindo e/ou removendo atributos, conforme a configuração inicial. Os processos de seleção de atributos podem buscar o

conjunto de atributos em todo o espaço de busca (completa), utilizando heurísticas que buscam regiões de busca mais prováveis ou, então, randomicamente com, por exemplo, um algoritmo genético, assim como sugerido na proposta Blum & Langley.

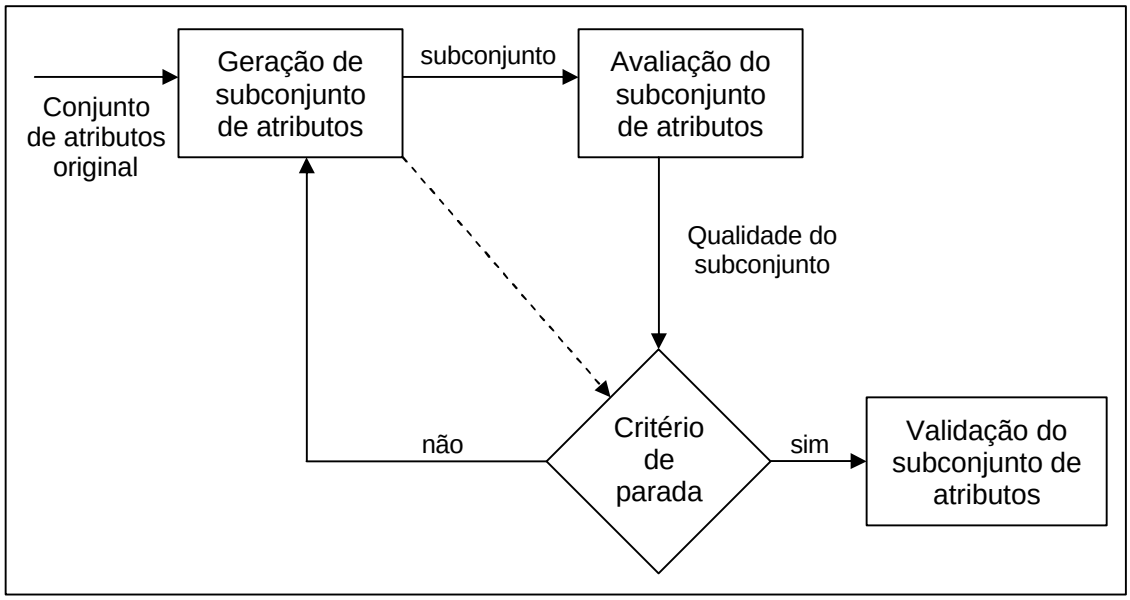


Figura 2.3. Modelo do processo de seleção de um conjunto de atributos com validação [Dash & Liu 1997].

2) **Função de avaliação**: é uma medida que avalia cada subconjunto de atributos criado no processo de geração de subconjuntos, com vistas à seleção do subconjunto mais adequado aos objetivos do processo de seleção. Baseado nos métodos de seleção existentes, a abordagem Dash & Liu caracteriza as funções de avaliação em cinco grupos descritos a seguir.

2.1) **Medidas de distância:** também conhecidas como medidas de separabilidade ou discriminação. Para um problema com duas classes, um atributo X é preferido em relação a um atributo Y se X induz uma diferença entre as probabilidades condicionais de duas classes maior do que Y. Se a diferença for zero, os atributos X e Y são considerados indiscerníveis;

2.2) **Medidas de informação:** determinam o ganho de informação com o uso de um atributo. O ganho de informação de um atributo X é definido como a diferença entre a incerteza antes e após a utilização do atributo X. Um atributo X é preferido em relação a um atributo Y se o ganho de informação do atributo X for maior que o do atributo Y;

2.3) **Medidas de dependência:** também conhecidas como medidas de correlação. São usadas para predizer o valor de uma variável a partir do valor de outra;

2.4) **Medidas de consistência**: define-se um conjunto de dados como totalmente consistente quando não há instâncias com mesmos valores para os atributos e classes diferentes. Busca-se obter o menor número de atributos possíveis que satisfaça uma taxa de inconsistência aceitável, ao menos próxima ao grau de inconsistência quando todos os atributos são utilizados;

2.5) **Taxa de erro do classificador**: utiliza o algoritmo de aprendizado que será utilizado posteriormente na classificação para avaliar os subconjuntos de atributos. Referente à abordagem *wrapper*.

3) **Critério de parada:** é o critério que deve ser utilizado para interromper a busca pelo subconjunto de atributos. Esses critérios podem sofrer influências da função de avaliação (o processo termina quando novos subconjuntos gerados não obtêm melhores classificações) ou do processo de geração (o processo termina após a identificação de um determinado número de atributos).

4) **Validação do subconjunto de atributos:** trata-se da validação do subconjunto de atributos obtido, que pode ser realizada usando um conjunto de dados separado, ou então, comparando os resultados com os obtidos por outros métodos de seleção de atributos. Esta característica não faz parte do processo de seleção em si, mas é fundamental para garantir a efetividade do mesmo.

A maneira de analisar as funções de avaliação utilizada na proposta Dash & Liu é baseada na revisão realizada em [Ben-Bassat 1982], que propôs, no início da década de 80, as medidas de incerteza (similar à de informação), distância e dependência. A revisão em [Dash & Liu 1997] aqui descrita trata-se, portanto, de uma extensão da abordagem de [Ben-Bassat 1982], já que insere duas outras medidas que surgiram mais recentemente.

Na Tabela 2.4, extraída de [Dash & Liu 1997], são comparadas as diferentes funções de avaliação quanto à generalidade (se o subconjunto de atributos selecionado pode ser utilizado em conjunto com diferentes métodos de AM), complexidade temporal (tempo necessário para a seleção do subconjunto de atributos) e acurácia (quão precisa é a classificação utilizando o subconjunto de atributos selecionado). No caso da acurácia,

apenas em relação à taxa de erro do classificador é possível determinar que a acurácia será alta, já que o subconjunto selecionado é otimizado para o método de AM utilizado posteriormente.

Tabela 2.4. Comparação entre as funções de avaliação [Dash & Liu 1997].

Função de Avaliação	Generalidade	Complexidade Temporal	Acurácia
Medidas de Distância	Sim	Baixa	–
Medidas de Informação	Sim	Baixa	–
Medidas de Dependência	Sim	Baixa	–
Medidas de Consistência	Sim	Média	–
Taxa de erro do classificador	Não	Alta	Muito alta

As informações presentes na Tabela 2.4 permitem dizer que a função de avaliação 'taxa de erro do classificador', apesar de garantir uma alta acurácia, deve ser evitada quando há limitação de tempo. É informado também que se a função de avaliação usada for a taxa de erro do classificador o processo perde em generalidade. Experimentos descritos em [Santoro & Nicoletti 2005], entretanto, evidenciam que em algumas situações, atributos selecionados usando a taxa de erro de um determinado classificador podem ser utilizados com sucesso por outros.

2.2.3 Conclusões sobre as Propostas de Categorização de Métodos de Seleção de Atributos

Considerando ambas as propostas de categorização, a descrita em [Blum & Langley 1997] e a descrita em [Dash e Liu 1997], pode-se afirmar que um método de seleção de atributos possui duas características principais, a saber, uma função de avaliação que avalia cada subconjunto de atributos e um processo de busca pelo melhor subconjunto de atributos, normalmente aquele que obtém o melhor resultado usando a função de avaliação.

No caso dos filtros, a característica fundamental a ser investigada é a maneira como os atributos são avaliados, enquanto que nos *wrappers* é necessária uma maior atenção aos métodos de busca, já que são utilizados algoritmos de aprendizado para a avaliação dos conjuntos de atributos. É interessante notar que a proposta Dash & Liu não enfatiza tanto a divisão entre *wrapper* e filtro, embora destaque também as diferenças entre ambos os modelos.

Na Tabela 2.5 as buscas exaustiva, seqüencial e randômica são analisadas quanto à eficiência (capacidade de encontrar boas soluções), complexidade (relativo ao espaço de busca explorado), e suas vantagens e desvantagens, que apontam para as situações onde devem ser preferenciais.

Tabela 2.5. Principais características das buscas exaustiva, seqüencial e randômica.

Busca	Eficiência	Complexidade	Vantagens	Desvantagens
Exaustiva	Sempre encontra solução ótima	Exponencial	Alta precisão	Alta complexidade
Seqüencial	Boa, se não for necessário realizar *backtracking*	Quadrática	Simples e rápida	Não realiza *backtracking*
Randômica	Boa, utilizando bons parâmetros de controle	Normalmente baixa	Escapa de mínimos locais	Difícil determinar bons parâmetros

2.3 O Modelo *Wrapper*

A abordagem de seleção de atributos computacionalmente mais exigente é aquela que utiliza um algoritmo de aprendizado para avaliar a qualidade de cada subconjunto de atributos (o mesmo que será utilizado posteriormente no aprendizado), em um modelo conhecido como *wrapper* [John et al. 1994].

Um algoritmo *wrapper* típico realiza a busca no mesmo espaço de subconjuntos de atributos que os métodos filtro e integrado (ver Figura 2.2). Avalia os diversos subconjuntos, entretanto, induzindo um classificador a partir do conjunto de treinamento e usa a acurácia deste classificador (no conjunto de teste) como medida para avaliar o subconjunto de atributos. A Figura 2.4, extraída de [John et al. 1994], mostra um diagrama do modelo *wrapper*.

A utilização do modelo *wrapper* é relativamente recente em pesquisas na área de AM. Conceitualmente, entretanto, esta proposta faz parte da literatura em Estatística e Reconhecimento de Padrões há muito tempo. Trabalhos antigos, embora não definissem os métodos como *wrappers*, já utilizam o mesmo conceito definido por esta abordagem. Os autores Siedlecki & Sklansky, em [Siedlecki & Sklansky 1988], afirmam que o único modo legítimo de avaliar os subconjuntos de atributos deve ser por meio da taxa de erro do classificador para o qual o subconjunto está sendo identificado.

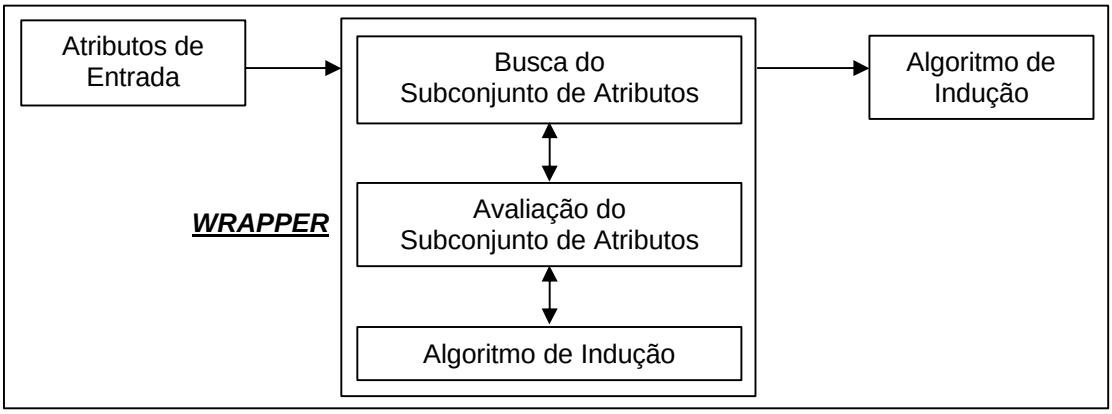


Figura 2.4. Modelo de um método de seleção de subconjunto de atributos do tipo *wrapper* [John et al. 1994].

De fato o modelo *wrapper* é conceitualmente mais interessante que o filtro na perspectiva de otimizar a precisão de classificação, já que os subconjuntos de atributos são avaliados utilizando o mesmo critério usado para estimar a precisão do algoritmo de aprendizado. No entanto, utilizar o próprio algoritmo de aprendizado para avaliar cada subconjunto de atributos requer tamanho poder computacional que esta estratégia não pode ser utilizada com um desempenho satisfatório em qualquer situação. Os *wrappers* devem ser evitados especialmente quando o algoritmo de aprendizado é demasiadamente complexo ou, então, quando o volume de dados é muito grande. Esta segunda restrição é um limitador que se opõe a um dos objetivos da seleção de atributos, que é a redução do volume de dados [Liu & Setiono 1996a].

Normalmente algoritmos de aprendizado que exigem menor poder computacional são utilizados, como o NN ou árvores de decisão mas, mesmo nestes casos, técnicas de otimização na busca são necessárias, para garantir um melhor desempenho [Blum & Langley 1997].

No modelo *wrapper* a forma como a precisão da classificação é avaliada é determinante, já que esta será utilizada para avaliar a qualidade de cada subconjunto de atributos. A n-validação cruzada é a técnica de avaliação de precisão recomendada em [John et al. 1994] para estar associada a um *wrapper* e, por isso, é adotada neste trabalho. Em [Kohavi & John 1997] é proposta a utilização da 5-validação cruzada, com a possibilidade de mais de uma execução da avaliação sobre o mesmo subconjunto de atributos caso o desvio padrão entre os cinco primeiros conjuntos avaliados seja maior que 1%. Para a realização dos experimentos neste trabalho adotou-se a 5-validação cruzada uma única vez para a avaliação da precisão do classificador durante o processo de busca.

Normalmente os modelos *wrapper* propostos na literatura dão bastante ênfase aos métodos de busca utilizados, já que a forma de avaliação depende apenas do algoritmo de aprendizado. Para a implementação de sistemas *wrapper* eficazes, é também fundamental a escolha de um algoritmo de aprendizado que não seja computacionalmente oneroso e que seja flexível na utilização de uma quantidade variável de atributos.

Buscas completas são utilizadas apenas em alguns trabalhos, já que a avaliação de cada possível subconjunto deve ser feita, demandando alto poder computacional. Em [Liu & Yu 2002], que é uma versão revisada de [Dash & Liu 1997], é descrito um estudo recente das atuais possibilidades existentes na área de seleção de atributos, onde são destacados apenas quatro *wrappers* que utilizam busca completa, o *Aproximate Monotonic Branch & Bound* (AMB&B) [Foroutan & Sklansky 1987], o *Beam Search* [Doak 1992] e dois métodos propostos em [Ichino & Sklansky 1984].

O AMB&B é um derivado do *Branch & Bound* (B&B). O B&B é um algoritmo que realiza uma busca em profundidade, iniciando com todos os atributos para, então, removê-los um a um. Como o B&B é baseado no princípio da monotonicidade, caso a remoção de um atributo não seja vantajosa (situação detectada por meio da redução do valor da função objetivo), os ramos que derivam deste atributo não são explorados. No caso do AMB&B, ramos com valor de função objetivo próximos, mesmo que inferiores ao valor máximo, também são explorados, permitindo a utilização deste algoritmo em domínios não monotônicos, utilizando funções objetivo que não sigam o principio da monotonicidade.

O *Beam Search*, por sua vez, é uma variação do *Best-First*, com a adição de um vetor que delimita o escopo da busca. Este vetor contém uma quantidade (pré-definida) de ramos a serem investigados (os mais promissores). Trata-se, portanto de uma versão menos abrangente do método *Best-First*, contendo-se apenas aos ramos com valores de função objetivo maiores. Se o vetor não for limitado o método realiza uma busca exaustiva (como o *Best-First*); se o vetor for limitado em apenas um elemento a busca é equivalente à busca *Hill-Climbing*.

Métodos *wrapper* que realizam uma busca heurística, por sua vez, são utilizados em diversos trabalhos. Estratégias como o *Wrapper Sequential Forward Generation* (WSFG) e *Wrapper Sequential Backward Generation* (WBSG) [Devijver & Kittler 1992] são pioneiras e seguem os modelos básicos de busca já definidos anteriormente

(*forward selection* e *backward elimination*), sendo ambos os métodos base para diversas outras formas de busca heurística. O *Sequential Backward Selection-Slash* (SBS-Slash) [Caruana & Freitag 1994] segue a mesma heurística do WBSG, mas de forma mais agressiva, eliminando mais de um atributo a cada etapa. Já o *Bi-Directional Search* (BDS) e o *(P,Q) Sequential Search* (PQSS) [Doak 1992] podem eliminar e adicionar atributos conforme a necessidade, diferentemente dos outros algoritmos seqüenciais citados. Diversos outros métodos derivados dos WSFG e WBSG são, em geral, apenas versões otimizadas dos métodos originais.

As buscas randômicas (ou buscas não-determinísticas) também estão sendo bastante exploradas no modelo *wrapper*. São cinco os métodos identificados em [Liu & Yu 2002]. Os Algoritmos Genéticos (AGs) e o *Simulated Annealing* são não-determinísticos tanto na geração do subconjunto inicial de atributos quanto na própria busca. Já o método identificado como RGSS (*Random Generation Plus Sequential Selection*) [Doak 1992] gera apenas um subconjunto inicial de atributos randômico para, então, aplicar uma busca heurística como o WSBG ou o WSFG. Outros algoritmos são o LVW [Liu & Setiono 1996b], que utiliza o algoritmo de Las Vegas para a geração dos subconjuntos de atributos, e o RMHC-PF1 (*Random Mutation Hill Climbing – Prototype and Feature Selection*) que seleciona randomicamente atributos e instâncias e avalia os subconjuntos de atributos com o algoritmo NN, sendo que novos subconjuntos são gerados por meio da mutação de um vetor que armazena os atributos selecionados anteriormente.

É interessante ressaltar que quando o modelo *wrapper* é utilizado, o algoritmo de aprendizado utilizado é tratado como uma caixa preta, onde o conhecimento de seu funcionamento interno não é necessário. Por esta razão é que os estudos propostos utilizando o modelo *wrapper* focalizam principalmente a estratégia de busca utilizada, bem como a forma como é avaliada a precisão da classificação.

2.4 O Modelo Filtro

A principal característica de um método de seleção de atributos identificado como filtro é a de ser um processo independente, que pode ser acoplado a qualquer outro processo (de aprendizado, por exemplo), conforme mostra a Figura 2.5. São métodos pouco custosos computacionalmente, quando comparados com métodos *wrapper* e, conseqüentemente, são bastante populares.

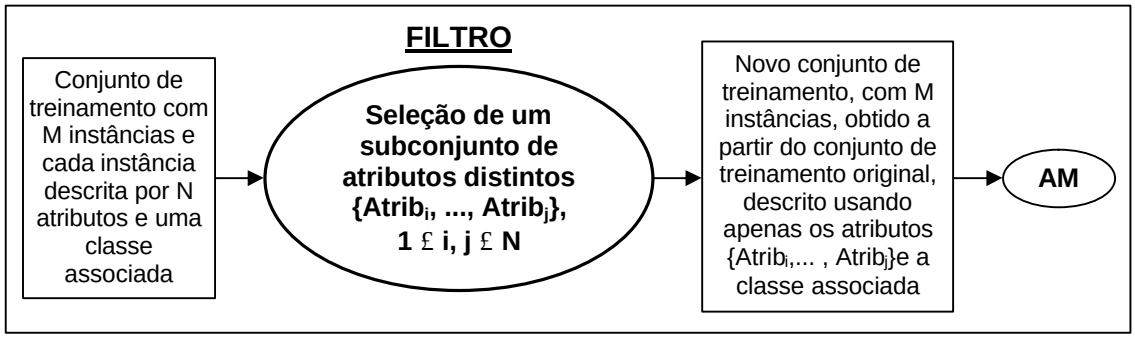


Figura 2.5. Modelo de um método de seleção de subconjunto de atributos do tipo filtro, usado como pré-processamento a um método de AM.

Considere um conjunto de instâncias C tal que |C| = M, no qual cada instância é descrita por um conjunto de N atributos A = {Atrib$_1$, Atrib$_2$, ..., Atrib$_N$}. Métodos de seleção de atributos do tipo filtro[2] transformam o conjunto C em um conjunto C', tal que |C'| = M e cada instância em C' é descrita por um conjunto com N' atributos distintos, A' = {Atrib$_i$, ..., Atrib$_j$}, |A'| = N', 1 ≤ i, j ≤ N e N' < N.

Na proposta Dash & Liu são apontados quatro critérios de avaliação de conjuntos de atributos utilizados atualmente, que podem ser usados em modelos filtro. As medidas usadas na avaliação são:

- **Distância**: discriminação entre atributos;
- **Informação**: ganho de informação com adição ou remoção de um atributo;
- **Dependência**: relação entre atributos e classes;
- **Consistência**: redução do conjunto de atributos mantendo a consistência original.

Estas medidas são utilizadas associadas a métodos de busca ou, então, são utilizadas na construção de um *ranking* onde, para cada atributo, é encontrado um valor de relevância, para então serem selecionados aqueles atributos que ultrapassam um determinado valor de relevância.

Na literatura pode ser evidenciado que a maioria dos algoritmos consagrados de seleção atributos são do tipo filtro, com destaque para o Relief [Kira & Rendell 1992] e o Focus [Almuallim & Dietterich 1991], e as suas otimizações tais como o Relief-F [Kononenko 1994] e os Focus-2 e C-Focus [Almuallim & Dietterich 1992] [Arauzo et al. 2003] respectivamente, que estarão sendo abordados em detalhes no Capítulo 4.

[2] Há também o conceito de filtro para a seleção de instâncias, que não faz parte dos objetivos de pesquisa desta dissertação.

Enquanto o Relief utiliza uma medida de distância e o Focus utiliza uma medida de consistência, métodos como o DTM [Cardie 1993] e o MDLM [Sheinvald et al. 1990] utilizam a medida de informação. O Preset [Modrzejewski 1993] e o POE+ACC [Mucciardi & Gose 1971] usam como medida a dependência entre os atributos e as classes. Inúmeros outros filtros já foram propostos, tais como o LVF [Liu & Setiono 1996a], o *Branch & Bound* [Narendra & Fukunaga 1977], um método proposto em [Koller & Sahami 1996] ou ainda o LVI [Liu & Setiono 1998], um método incremental, baseado no LVF e desenvolvido para melhorar o desempenho computacional em grandes bases de dados. A seguir são brevemente apresentados estes métodos.

O DTM (*Decision Tree Method*) realiza a seleção de atributos utilizando o C4.5 sobre o conjunto de dados. São selecionados os atributos presentes na árvore de decisão, descartando aqueles atributos ausentes da árvore. Desta maneira, este método não realiza uma busca completa, mas sim heurística, usando a seleção de atributos integrada ao C4.5.

O MDLM (*Minimum Description Length Method*) opta por uma busca completa no espaço de busca, com intuito de eliminar todos os atributos desnecessários. A teoria utilizada como base para este método é o MDLC (*Minimum Description Length Criterion*) [Rissanen 1978], que identifica subconjuntos de atributos que podem ser expressos por outros conjuntos com a mesma representatividade. Todos os possíveis subconjuntos de atributos são analisados, resultando na seleção do menor subconjunto a satisfazer o MDLC. Desta forma é possível eliminar subconjuntos de atributos que não contribuem para a descrição das instâncias pelo critério do MDLC.

O POE+ACC (*Probability Of Error & Average Correlation Coefficient*) é um método que utiliza uma busca heurística para a seleção de atributos. O primeiro passo é a seleção do atributo com a menor probabilidade de erro na classificação, para então ser selecionado o atributo que proporciona a soma mínima entre a probabilidade de erro na classificação e um coeficiente de correlação médio (ACC). Este ACC é referente à correlação entre o novo atributo e aqueles já selecionados. A proposta é escolher atributos que diminuam a probabilidade de erro e que não possuam forte correlação entre eles. O processo só pára após a seleção de uma quantidade de atributos definida pelo usuário.

Outro método de seleção de atributos que utiliza como medida a dependência entre os atributos é o Preset, baseado em alguns conceitos da Teoria dos Conjuntos Aproximados [Pawlak 1984]. É selecionado um reduto, que é caracterizado como um subconjunto de atributos capaz de classificar as instâncias do conjunto de treinamento da mesma forma que o conjunto total de atributos. Os atributos selecionados são aqueles do reduto.

O LVF utiliza uma medida de consistência mais elaborada que o Focus, permitindo pequenas inconsistências para tratar melhor os casos com ruído. Além disso, se diferencia do Focus por utilizar como método de busca o algoritmo conhecido como Las Vegas, que gera subconjuntos aleatórios de atributos (o usuário determina a quantidade de subconjuntos gerados). Cada subconjunto que satisfaz a condição de consistência é informado ao usuário e apenas subconjuntos com o mesmo número ou menos atributos que o melhor encontrado até então são gerados durante a busca.

O LVI é uma versão aprimorada do LVF, com a finalidade de tratar bases de dados com grande número de instâncias. O próprio LVF é utilizado como função de avaliação, mas é utilizada uma pequena amostra das instâncias disponíveis (entre 10% e 20%) para que o subconjunto de atributos seja encontrado e depois verificado sobre todas as instâncias disponíveis. Caso haja inconsistência nesta verificação, o LVF é executado novamente sobre a amostra de instâncias inicial, incrementado com as instâncias que apresentaram inconsistência. Por este motivo, o algoritmo é chamado de incremental.

O método de Koller & Sahami [Koller & Sahami 1996], procura analisar se os atributos dão alguma informação adicional em relação à que já se obtém com o restante dos atributos. Trata-se de uma medida de informação, que utiliza *Markov Blanket*, para realizar tal avaliação, utilizando uma busca heurística (ver [Koller & Sahami 1996] para mais detalhes).

O *Branch & Bound* é basicamente um mecanismo de busca, que utiliza o conjunto de todos os atributos como estado inicial, para removê-los um a um. É assumido o conceito de monotonicidade, que define que um subconjunto de atributos não pode ser melhor que outro subconjunto maior que o contém. A busca realizada é completa, mas ramos da árvore que apresentam valores inferiores na função de avaliação não são explorados, devido à utilização da monotonicidade. Também por este motivo, as medidas de avaliação utilizadas devem seguir este princípio, como exemplo a distância de Mahalanobis [Duran & Odell 1974] ou de Bhattacharya [Narendra & Fukunaga 1977].

2.5 Outras Considerações sobre a Seleção de Subconjuntos de Atributos

Até aqui as vantagens da redução do conjunto de atributos foram apresentadas e discutidas; é importante, entretanto, notar que a redução do conjunto de atributos pode não ser interessante em algumas situações, como em casos onde não há muitos atributos ou, então, quando se sabe sobre a relevância deles.

Outro ponto importante a ser considerado é que os métodos de seleção de atributos existentes não são de uso geral. Comumente cada um deles apresenta algumas restrições com relação aos tipos de dados aos quais pode ser aplicado. Em [Dash & Liu 1997] são apontadas três características associadas aos dados que devem ser levadas em consideração quando da escolha de um método de seleção de atributos (no mesmo estudo podem ser encontrados alguns métodos de seleção de atributos classificados quanto a estas características):

- **Tipo dos dados**: é necessário observar os tipos de valores que os atributos podem assumir: discreto, contínuo, nominal e booleano. Alguns métodos podem não ser capazes de tratar todos estes tipos de dados. Outra consideração é sobre as classes, já que alguns métodos não conseguem lidar com múltiplas classes, mas apenas com classificações binárias;
- **Dimensão dos dados:** quanto à dimensão, devem ser consideradas tanto a capacidade do algoritmo lidar com um conjunto de treinamento muito pequeno quanto com um grande volume de dados;
- **Presença de ruído:** os métodos podem ser capazes ou não de manipular dados com ruído ou, então, dados conflitantes.

Em [Guyon & Elisseeff 2003] é proposto um *checklist* a ser realizado por aqueles que pretendem empregar a redução do conjunto de atributos, que é apresentado resumidamente na Figura 2.6 e que, apesar de trivial e simplista, pode ser usado como guia quando se considera o problema de seleção de atributos pela primeira vez.

1. **Há conhecimentos sobre o domínio?**
   Se sim, construa um conjunto de atributos com estes conhecimentos.
2. **Os atributos possuem as mesmas proporções (isto é, o mesmo intervalo de variação de valores)?**
   Se não, é interessante normalizá-los.
3. **Há suspeitas de interdependência de atributos?**
   Se sim, aumente o conjunto de atributos construindo atributos agrupados.
4. **É necessário eliminar as variáveis de entrada (por razões de custo, velocidade ou compreensão dos dados)?**
   Se não, construa atributos agrupados ou somas ponderadas dos atributos.
5. **É necessário avaliar os atributos separadamente (para conhecer sua influência no sistema ou por o número de atributos ser muito grande exigindo um filtro prévio)?**
   Se sim, utilizar um *ranking* para os atributos; se não utilizar qualquer método.
6. **É realmente necessário um classificador?**
   Se não, pare.
7. **Há suspeitas de que os dados estão com ruído (têm padrões de entrada sem sentido, saídas ruidosas ou classificações erradas)?**
   Se sim, detecte as instâncias atípicas utilizando os atributos rankeados obtidos no passo 5; reavalie estes atributos ou os exclua.

Figura 2.6. *Checklist* para a utilização de um método de seleção de atributos.

É importante comentar que a seleção de atributos já foi explorada com outras intenções além da redução do conjunto de dados, como o fizeram Caruana & Sá em [Caruana & de Sa 1998] e [Caruana & de Sa 2003], que propõem utilizar os atributos não selecionados como atributos de saída em aprendizado multitarefa, o que mostra que o tema pode ser explorado em situações outras que aquelas inicialmente cogitadas.

No Capítulo 3 o modelo *wrapper* de seleção de atributos é investigado, sendo implementados cinco métodos de busca em conjunto com dois diferentes algoritmos de AM. No Capítulo 4 são estudados em profundidade três famílias de métodos do tipo filtro.

3 CAPÍTULO

O MODELO *WRAPPER* DE SELEÇÃO DE ATRIBUTOS

Este capítulo investiga diferentes esquemas de implementação do modelo *wrapper*, no contexto de seleção de subconjuntos de atributos, envolvendo cinco métodos de busca (*Hill-Climbing*, *Random Bit Climber*, *Beam Search*, Las Vegas e Algoritmo Genético) e dois diferentes algoritmos de AM (*Neareast Neighbor*, DistAl).

Em virtude do foco deste trabalho ser a investigação de métodos para a seleção de subconjuntos de atributos no contexto de AM, detalhes sobre os métodos de aprendizado usados estão no Apêndice B.

3.1 Introdução

Como comentado anteriormente, a seleção de atributos do tipo *wrapper* é aquela caracterizada por utilizar um algoritmo de aprendizado para avaliar a qualidade dos subconjuntos durante o processo de seleção de atributos. A qualidade de cada subconjunto é medida como a precisão do classificador induzido pelo método de aprendizado, usando o subconjunto de atributos em questão.

O modelo *wrapper* assume um método de busca subjacente que identifica, no espaço de atributos, aqueles subconjuntos a serem considerados. É fundamental que o método de busca permita encontrar da forma mais eficiente (com eficácia) o melhor subconjunto de atributos, já que o processo de busca nesse espaço é caracterizado como um problema de complexidade NP [Amaldi & Kann 1998]. Por este motivo o estudo de alguns dos principais métodos de busca utilizados no processo de seleção de atributos do tipo *wrapper* é um dos focos deste capítulo.

A abordagem *wrapper* não pressupõe o conhecimento do algoritmo de

aprendizado, que é tratado como uma caixa preta pelo método de busca. Quando da implementação, entretanto, é necessário que a interface do algoritmo seja conhecida para poder utilizá-lo articulado ao método de busca adotado. Efetivamente, são necessárias informações sobre o formato de entrada dos dados e o formato de saída, para capturar a precisão de classificação, bem como dados mais específicos (quando disponíveis), tais como a quantidade de nós de uma árvore de decisão ou o número de regras, que também podem ser utilizados na busca.

Nos experimentos realizados e descritos neste capítulo foram utilizados dois algoritmos de aprendizado, a saber: o *Nearest Neighbor* (1-NN) [Cover & Hart 1967] e o DistAl [Yang et al. 1999]. Os resultados da seleção de atributos utilizando os dois algoritmos de aprendizado apresentaram diferentes comportamentos, como será analisado posteriormente.

Para o uso de qualquer método de busca, é necessário definir primeiro a organização do espaço onde tal busca vai acontecer. No problema de seleção de atributos aqui estudado, cada estado desse espaço representa um subconjunto de atributos, descrito por um vetor de valores booleanos com n posições, sendo n o total de atributos que descrevem uma instância de treinamento. O valor de cada posição do vetor indica se o atributo correspondente pertence ao subconjunto (1) ou não (0).

Como já mencionado, o objetivo da busca é encontrar o subconjunto de atributos que promove a maior precisão de classificação. Note que não necessariamente existe um único destes subconjuntos. Encontrar o melhor subconjunto é uma tarefa extremamente custosa computacionalmente, visto que o espaço de busca é de 2^n estados, para n sendo o número total de atributos. Por este motivo, as buscas completas em geral não são viáveis quando é utilizado o modelo *wrapper*, sobretudo quando o espaço é descrito por muitos atributos.

Neste capítulo são abordados cinco métodos de busca empregados no modelo *wrapper*, que foram investigados nesta dissertação, a saber: o *Hill-Climbing* [Vafaie & De Jong 1993], o *Random Bit Climber* [Davis 1991], o *Beam Search*/*Best First* [Aha & Bankert 1995], o Las Vegas [Liu & Setiono 1996b] e Algoritmos Genéticos [Goldberg 1989]. Dependendo do método de busca, ele pode ainda ser caracterizado em função da estratégia de direção de busca adotada. A Seção 3.2 descreve as possíveis estratégias de direção e a Seção 3.3 os métodos de busca utilizados neste trabalho.

3.2 Estratégias de Direção da Busca – *Forward Selection*, *Backward Elimination* e *Random*

Conforme mencionado no Capítulo 2, a busca por subconjunto de atributos tem como uma de suas características o estado a partir da qual ela se inicia (chamado de estado inicial). Na literatura [Guyon & Eliseef 2003], é frequentemente chamado de *Forward Selection* (neste trabalho referenciado como *Forward*) a estratégia de busca que tem como estado inicial um conjunto vazio de atributos, ao qual, à medida que a busca progride, vão sendo adicionados atributos. Já o termo *Backward Elimination* (neste trabalho referenciado como *Backward*) caracteriza uma estratégia de busca que tem como estado inicial o conjunto com todos os atributos que, à medida que a busca progride, vão sendo eliminados.

As duas estratégias são bastante exploradas na literatura e a *Forward* é freqüentemente indicada como melhor opção para a seleção de atributos do tipo *wrapper* já que utiliza menos atributos no estágio inicial [Guyon & Eliseef 2003], o que torna o processo de construção e avaliação do classificador mais rápido.

Ainda na referência [Guyon & Eliseef 2003], é argumentado que quando a estratégia *Backward* é utilizada, os métodos de busca tendem a obter um subconjunto de atributos que resulta em maior precisão de classificação, por serem capazes de selecionar atributos que, em conjunto, melhoram a precisão (atributos inter-relacionados).

É importante enfatizar que os termos *Forward* e *Backward*, se referem às estratégias que definem a direção na qual a busca será realizada, adicionando ou eliminando atributos, respectivamente. Já os métodos de busca *Sequential Forward Selection* (SFS) e *Sequential Backward Elimination* (SBE) [Kittler 1978] são métodos de seleção de atributos que utilizam as estratégias *Forward* e *Backward* respectivamente, em conjunto com o método de busca *Hill-Climbing*, que será abordado na Subseção 3.3.1.

Uma terceira estratégia, chamada neste trabalho de *Random*, parte de um estado inicial qualquer (isto é, com atributos selecionados randomicamente). Neste caso, cada processo de busca se inicia a partir de um estado inicial diferente (usualmente), gerando resultados finais também provavelmente diferentes. Esta estratégia se caracteriza tanto por adicionar, quanto remover os atributos, sempre em busca do estado que proporciona

maior precisão de classificação (busca similar à realizada pelo *Bi-Directional Search*). Os métodos de busca que utilizam esta estratégia consideram, portanto, a adição e remoção de atributos a cada iteração.

Nos experimentos realizados, que serão descritos e discutidos na Seção 3.4, são utilizadas as três estratégias em conjunto com os métodos de busca *Hill-Climbing* e *Beam Search*. Com o método *Random Bit Climber*, foram utilizados diferentes estados iniciais (todos os atributos, nenhum atributo e atributos selecionados randomicamente), mas sua busca não pode ser caracterizada como *Forward* ou *Backward* (apenas como *Random*), já que durante a busca atributos podem ser adicionados e removidos. Já o Algoritmo Genético e o método de busca Las Vegas iniciam a busca a partir de qualquer estado inicial e implementam uma estratégia de busca própria. Na Figura 3.1 é mostrado um esquema gráfico das três diferentes estratégias de busca em um espaço de busca definido por quatro atributos, onde cada caixa com quatro círculos indica um subconjunto; os círculos brancos indicam atributos não selecionados e os círculos sombreados indicam atributos selecionados.

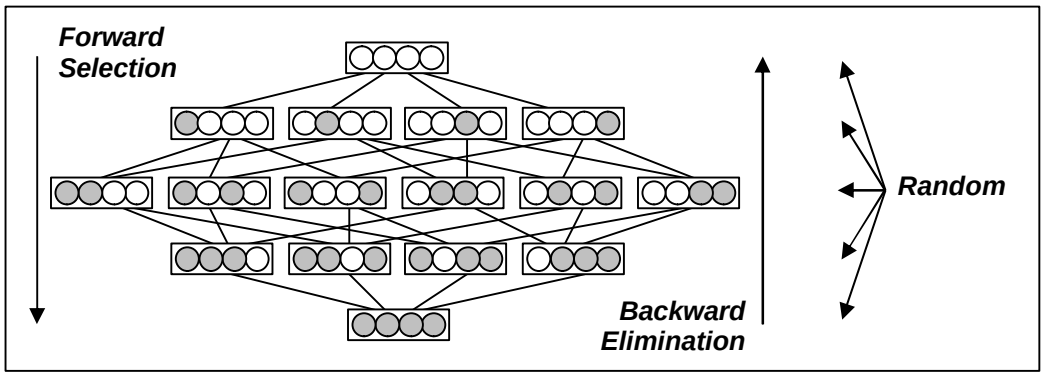
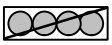
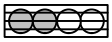


Figura 3.1 As estratégias de direção da busca *Forward*, *Backward* e *Random*.

3.3 Considerações sobre os Métodos de Busca Utilizados

Nesta seção são apresentados os métodos de busca utilizados na seleção de atributos do tipo *wrapper*. No que segue, cada um dos cinco métodos é descrito e comentado, sendo discutidas suas vantagens e desvantagens. São apresentados, também, o pseudo-código de cada um deles, um diagrama mostrando a dinâmica de funcionamento de cada um e um *trace* dos algoritmos. No contexto deste trabalho a palavra 'expansão' é usada para designar os possíveis estados que podem ser obtidos a partir do estado corrente.

Nas Figuras 3.3, 3.5 e 3.7, que representam um diagrama de dinâmica do processo de busca, a seguinte convenção foi adotada:

- Cada estado é representado como um diagrama  , onde cada círculo representa um atributo. O círculo sombreado indica que o atributo correspondente faz parte do estado. No diagrama apresentado neste parágrafo estão selecionados o segundo e quarto atributos;
- Os números sobre a parte superior direita dos estados representa a qualidade de cada estado, sendo 10 o estado com maior 'qualidade' e 0 o com menor 'qualidade';
- As letras sobre a parte superior esquerda dos estados é a identificação de cada estado, para ser referenciada no *trace* do algoritmo, apresentado em conjunto com o diagrama. O estado IN é o estado inicial que, em todos os diagramas, é o conjunto de atributos vazio;
- As linhas que ligam os estados representam as ligações entre os estados com apenas um atributo alterado;
- As setas tracejadas indicam os estados selecionados em cada iteração do *loop* principal na execução do algoritmo;
- Os estados cortados () são aqueles avaliados pelo método de busca mas que não geram estados filhos, por serem piores que outros já encontrados;
- Os estados com o fundo preto () são aqueles avaliados e mantidos no vetor *open* (no caso do método de busca *Beam Search*);
- Os estados com faixas laterais () são aqueles avaliados e mantidos no vetor *closed* (no caso do método de busca *Beam Search*);
- Os estados pontilhados () são aqueles não avaliados/explorados pelo método de busca.

3.3.1 Hill-Climbing

O *Hill-Climbing* é um dos métodos de busca mais simples empregados na seleção de atributos. Há apenas um estado corrente, que é inicializado conforme a estratégia de busca utilizada. A busca é realizada por meio da criação de novos estados a partir do estado corrente (i.e. expansão do estado corrente) que, no caso mais simples de geração de novos estados, é efetivada com a adição e/ou remoção de cada um dos atributos

existentes (este foi o modelo de geração de novos subconjuntos utilizado neste trabalho).

Dentre os estados resultantes da expansão do estado corrente, aquele com maior precisão é então definido como estado corrente, desde que seja melhor que o estado que o gerou. O processo então se repete com o novo estado corrente sendo expandido, só sendo interrompido, quando a partir do estado corrente qualquer estado gerado não tenha precisão de classificação melhor que o estado corrente.

É interessante notar que a forma como novos estados (denominados estados filhos) são criados a partir do estado corrente depende da estratégia de direção adotada. Caso seja utilizada a estratégia *Forward*, os estados filhos são todos aqueles com um atributo a mais selecionado. Já no *Backward*, os estados filhos são todos aqueles com um atributo a mais removido. No caso da estratégia *Random*, os estados filhos devem ser gerados nas duas direções, com a remoção e adição de cada atributo possível, envolvendo assim a avaliação de subconjuntos de atributos com menos e mais atributos.

Uma das características que diferencia o *Hill-Climbing* de outros similares tais como o *Random Bit Climber*, por exemplo, é que um atributo já modificado (adicionado ou removido) não é modificado novamente durante a busca e, consequentemente, não há *backtracking*. Desta forma, a busca fica bastante direcionada e tende a ser pouco custosa computacionalmente em relação a outros métodos de busca tal como o *Best-First*/*Beam Search*, por exemplo.

Em contrapartida, esta característica de forte direcionamento do *Hill-Climbing* muitas vezes acaba encontrando um máximo local, pois os atributos são adicionados um a um, levando à não captura de atributos inter-relacionados (sobretudo quando utilizada a estratégia *Forward*). O pseudo-código do algoritmo *Hill-Climbing*, adaptado de [Kohavi & John 1997] é descrito na Figura 3.2 e a dinâmica de funcionamento do método é mostrada na Figura 3.3, assim como um acompanhamento das principais variáveis durante a busca (*trace* do algoritmo).

No pseudo-código da Figura 3.2, a função f(o) representa a precisão de classificação do classificador induzido pelo método de aprendizado, usando um conjunto de treinamento descrito pelos atributos de o.

```
hill_climbing(estado_inicial, v)

v ← estado_inicial;          /* o estado corrente v recebe o estado_inicial */
repita
    expandir ← falso;        /* só haverá repetição se houver melhora nos filhos */
    filhos ← expansão(v);    /* expande v em seus filhos, por meio da adição/remoção
                                de atributos de v */
    avalia(filhos);          /* realiza a predição de classificação dos filhos com um
                                algoritmo de aprendizado */
    v' ← maior(filhos);      /* v' armazena o filho com maior f(o) */
    se ( f(v') > f(v) ) então   /* se o melhor filho v' é maior que o seu gerador v... */
      v ← v';                /* ... o estado v' passa a ser o estado corrente */
      expandir ← verdadeiro; /* ... e o processo se repete */
    fim se
enquanto (expandir)
retorna (v);
```

Figura 3.2. Algoritmo do método de busca *Hill-Climbing*, usado em um *wrapper*.

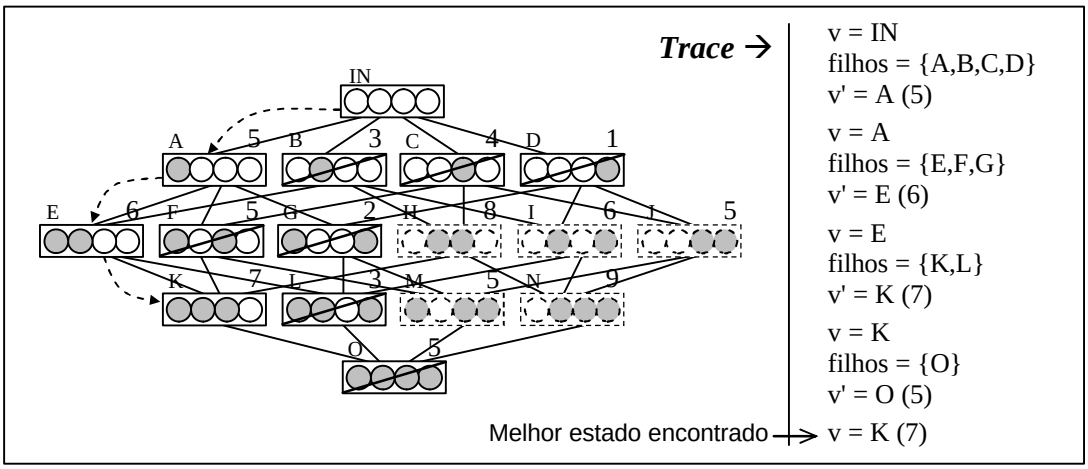


Figura 3.3. Dinâmica de funcionamento do algoritmo do *Hill-Climbing* com a estratégia *Forward*.

3.3.2 Random Bit Climber

O método *Random Bit Climber* (RBC) foi proposto em [Davis 1991] como um método de busca específico para vetores booleanos, como é o caso do problema de seleção de atributos aqui tratado. O RBC pode ser abordado como uma variação do *Hill-Climbing*, variação esta que não prioriza a escolha entre possíveis melhores estados, aceitando qualquer melhoria encontrada durante a busca. Não há na literatura qualquer trabalho que usa o RBC como método de busca para a seleção de atributos[3].

O RBC é um método de busca simples que, devido ao fato de direcionar a busca em função de uma ordem aleatória de inversão dos bits e usar *backtracking*, não se

[3] Uma variante do RBC chamada *Random Sample Climbing* (RSC), entretanto, foi utilizado em [Guerra-Salcedo et al. 1999] em conjunto com um algoritmo genético híbrido.

qualifica para a implementação das estratégias de direcionamento da busca *Forward* e *Backward*. Nos experimentos realizados com o RBC descritos na Seção 3.4, no entanto, foram utilizados três diferentes estados iniciais, o conjunto vazio (nenhum atributo), o conjunto original de atributos (todos os atributos) e um subconjunto de atributos randomicamente selecionados.

Assim como o *Hill-Climbing*, esse método considera apenas um estado corrente, que pode ser iniciado com um estado inicial qualquer. A busca é caracterizada pela inversão dos bits que definem a seleção dos atributos, sendo esta ordem de inversão definida randomicamente. A ordem de inversão de bits deve ser efetivada completamente para que uma nova ordem randômica seja criada.

Durante a busca, caso uma inversão de bit provoque um subconjunto com melhor precisão que o melhor encontrado até então, este novo subconjunto passa a ser o estado corrente e as inversões de bit são realizadas sobre este subconjunto até que todos os bits sejam invertidos a partir do estado onde a ordem de inversão dos bits foi definida.

Ao final da inversão de todos os bits na ordem designada, o melhor subconjunto gerado até então sofre novas inversões, seguindo uma nova ordem de inversão. O algoritmo só pára quando a inversão de todos os bits do melhor subconjunto encontrado até então não fornece qualquer melhoria, isto é, quando um máximo local é encontrado.

O RBC tem como característica principal a capacidade de percorrer o espaço de busca rapidamente, já que qualquer melhoria encontrada durante a busca é aceita, continuando a busca sem avaliar melhores possibilidades locais. Desta forma, bons resultados podem ser encontrados rapidamente sendo avaliados poucos subconjuntos de atributos. A utilização de uma ordem randômica, por sua vez, pode levar a resultados não ótimos, embora aceitáveis em geral (como verificado nos experimentos realizados na Seção 3.4).

O pseudo-código do algoritmo RBC, adaptado de [Davis 1991] é descrito na Figura 3.4. Um diagrama que mostra a dinâmica de funcionamento do método é mostrado na Figura 3.5, em conjunto com o *trace* do algoritmo.

```
random_bit_climber(estado_inicial, o, v)

atributos ← | o |;              /* é armazenado o número de atributos do conjunto original */
v ← estado_inicial;            /* v armazena o subconjunto corrente estado_inicial */
melhor-aval ← f(v);            /* é armazenada a avaliação do melhor subconjunto */
repita
    ordem ← gera-ordem( );     /* é gerado um vetor com a ordem de inversão dos atributos */
    expandir ← falso;          /* só haverá repetição se houver melhora nos filhos */

    para (i=0;i<atributos;i++)          /* loop executado atributos vezes */
        filho ← flip(v, ordem[i]);      /* o melhor subconjunto gerado tem o bit definido
                                           em ordem[i] invertido */
        filho-aval ← f(filho);          /* o subconjunto gerado é avaliado */
        se (filho-aval > melhor-aval) então   /* se o subconjunto é melhor que seu gerador */
            v ← filho;                  /* ... armazena o novo estado */
            melhor-aval ← filho-aval;   /* ... armazena a avaliação deste estado */
            expandir ← verdadeiro;      /* ... e o processo se repete */
        fim se;
    fim para;
enquanto (expandir)
retorna (v);
```

Figura 3.4. Algoritmo do método de busca *Random Bit Climber*.

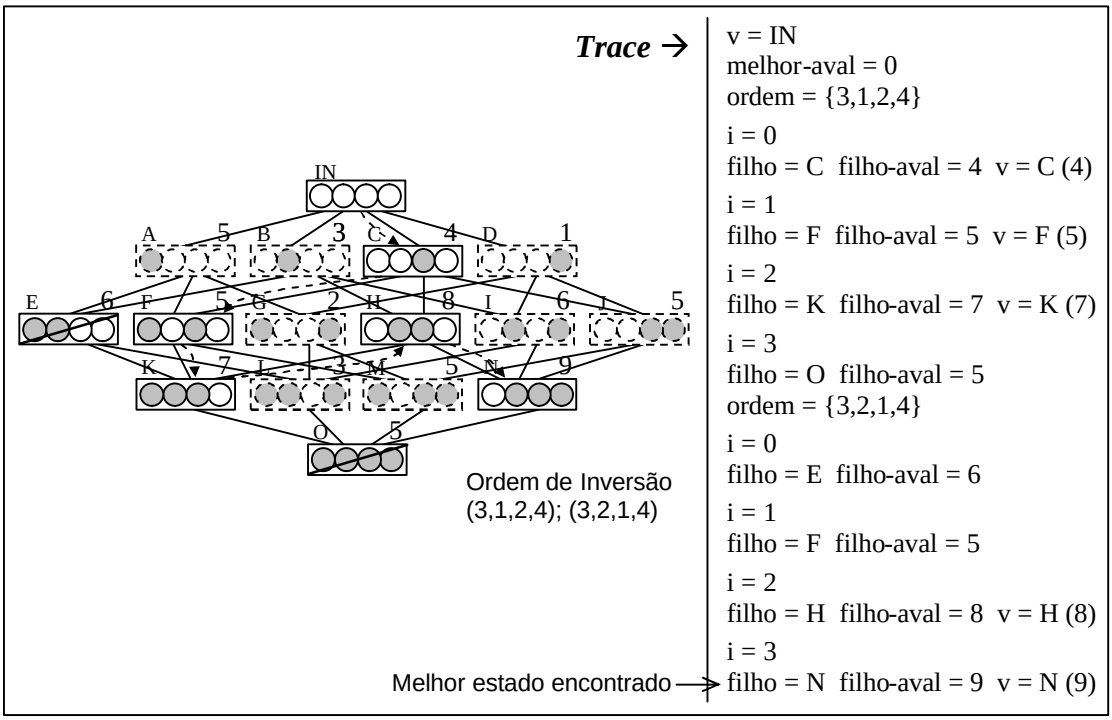


Figura 3.5. Dinâmica de funcionamento do algoritmo do RBC com a estratégia *Forward*.

3.3.3 *Best-First* e *Beam Search*

O método *Best-First* é mais robusto que o *Hill-Climbing* [Ginsberg 1993] e por robustez neste caso entende-se a capacidade de evitar mais eficientemente máximos locais. Esta qualidade existe devido à geração de novos estados não apenas a partir do melhor

subconjunto de atributos encontrado (estado corrente), mas também a partir daqueles que apresentam boa precisão de classificação (e que ainda não foram expandidos durante o processo de busca).

Para isso, o método *Best-First* utiliza dois vetores, um chamado de *Open,* que armazena os subconjuntos gerados que não foram expandidos ainda (armazena também seus valores de precisão), e outro chamado de *Closed*, com os subconjuntos que já foram expandidos, de forma a evitar uma nova expansão desnecessária. O vetor *Open* é sempre ordenado de forma que o subconjunto com maior precisão permaneça na primeira posição, sendo o primeiro candidato a ser expandido.

A expansão é sempre realizada a partir do melhor subconjunto já gerado presente no vetor *Open* (i.e., seu primeiro elemento), e é similar à realizada no *Hill-Climbing*, que é efetivada com a adição e/ou remoção de cada atributo, conforme a estratégia de busca utilizada.

O processo de busca pode abranger todos os subconjuntos de atributos possíveis, caso nenhuma restrição seja imposta. No entanto, é comum limitar a busca, terminando o processo quando k expansões seguidas não melhoram a precisão de classificação. O algoritmo *Best-First*, adaptado de [Kohavi e John 1997] é mostrado na Figura 3.6.

```
best_first( estado_inicial, tamanho_beam, k, melhor)

open ← estado_inicial;      /* o vetor open é iniciado com o estado_inicial */
melhor ← estado_inicial;    /* o estado_inicial é definido como o melhor subconjunto */
closed ← Ø;                 /* o vetor closed é iniciado vazio */
expansoes ← 0               /* para limitar a busca, é adicionado um contador de expansões */
repita
    v ← maior(open);            /* v é definido como o melhor subconjunto do vetor open */
    remove(v, open);            /* o subconjunto v é removido do vetor open */
    adiciona(v, closed);        /* o subconjunto v é adicionado ao vetor closed */

    se f(v) > f(melhor) então  /* se a qualidade de v é melhor que a de melhor... */
      melhor ← v;               /* ...melhor passa a ser v */
      expansoes ← 0;            /* ...e as expansões são zeradas */
    fim se
    expansões ← expansões+1;      /* acrescenta as expansões realizadas  */
    filhos ← expansão(v);         /* aplica operadores de adição/remoção de atributos em
                                     v, gerando novos subconjuntos, e os armazena no vetor
                                     filhos */
    avalia(filhos);               /* realiza a avaliação dos filhos, desde que não estejam
                                     nos vetores open ou closed  */
    adiciona(filhos, open);       /* adiciona os subconjuntos filhos avaliados ao vetor open */
    limita(open, tamanho_beam);   /* no método Beam Search o vetor open é limitado, para
                                     armazenar um determinado número de subconjuntos;
                                     para o Best-first este valor é infinito (não há limitação) */
enquanto (expansões < k)
retorna (melhor);
```

Figura 3.6. Algoritmo do método de busca *Best-First*.

O *Beam Search* é uma versão modificada do *Best-First*, onde o tamanho do vetor dos melhores subconjuntos ainda não expandidos (o vetor *Open*) é limitado, focalizando assim a busca apenas nos melhores subconjuntos gerados durante a busca (a variável tamanho_beam limita o tamanho do vetor *Open*). Com esta limitação, a busca não é completa, já que alguns estados nunca são expandidos. Mesmo com esta restrição no vetor *Open*, também é recomendável limitar o número de expansões, assim como sugerido no caso do método *Best-First*, a fim de limitar a abrangência e, principalmente, o tempo de busca.

Na Figura 3.7 é mostrado um diagrama da dinâmica de funcionamento do método *Beam-Search*, que inclui três diferentes estágios do processamento, bem como um *trace* do algoritmo. A variável tamanho_beam é definida como três neste contexto.

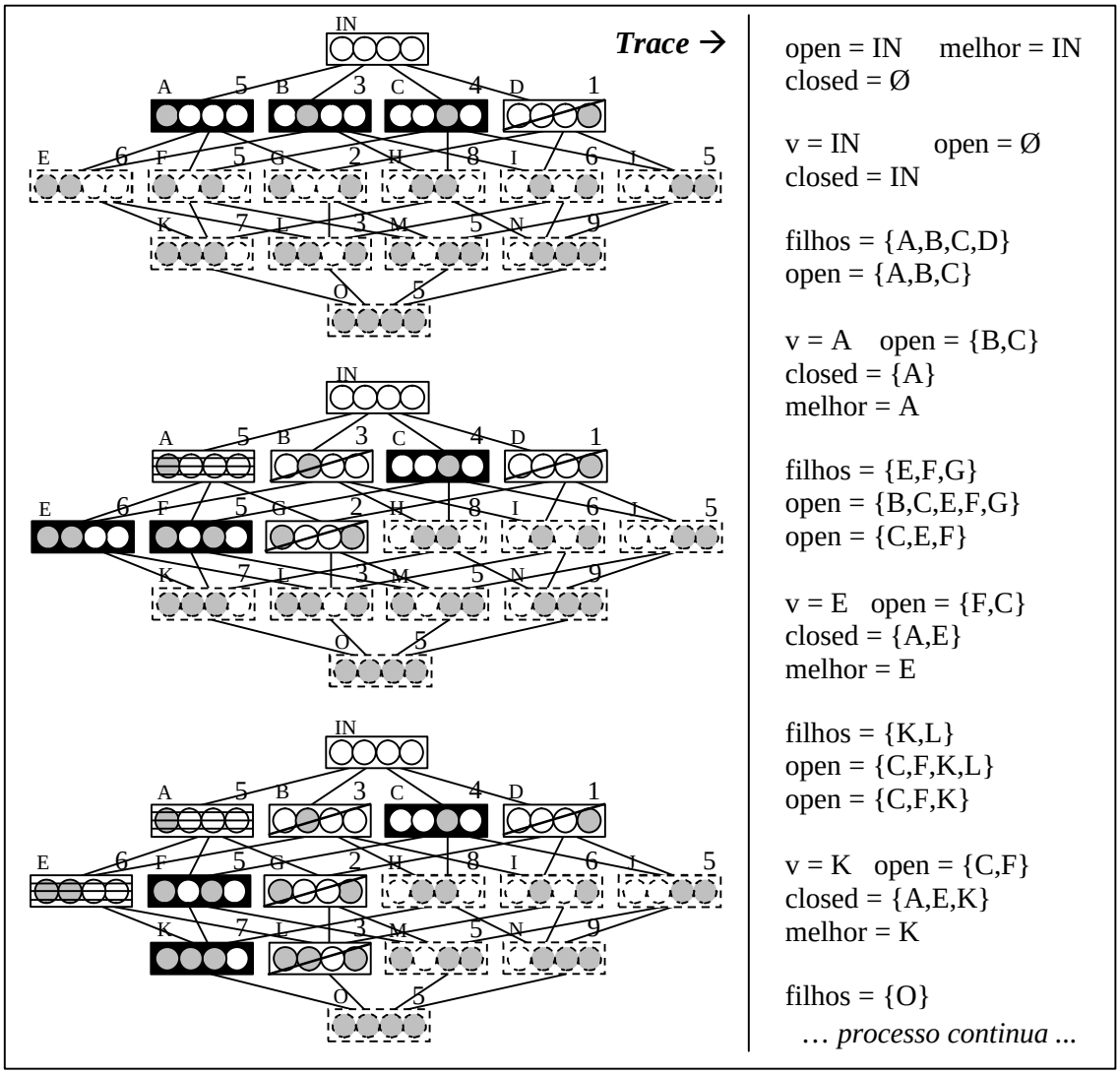


Figura 3.7. Dinâmica de funcionamento do algoritmo *Beam Search*.

3.3.4 Las Vegas

O método de busca do tipo Las Vegas [Brassard & Bratley 1996] usa a geração randômica de subconjuntos. Seu funcionamento é extremamente simples, num processo onde são gerados sucessivos subconjuntos randômicos, sem qualquer heurística de otimização para a busca. A utilização do método de busca Las Vegas para a seleção de atributos foi proposta em [Liu & Setiono 1996b].

Durante o processo de busca, cada subconjunto gerado randomicamente é avaliado e o melhor subconjunto gerado durante o processo é o único armazenado. O processo de geração aleatória de subconjuntos é realizado até que K subconjuntos em seqüência sejam gerados sem que seja encontrado um subconjunto que seja melhor do que o melhor subconjunto encontrado durante a busca.

Os autores propõem que o valor de K seja definido como $60 \times n$, sendo n a quantidade de atributos que descrevem o espaço de busca. A idéia central deste algoritmo, segundo seus autores, é não se preocupar com a escolha dos atributos que, em métodos como o *Hill-Climbing*, acabam por interferir de forma determinante na busca.

Liu e Setiono, em [Liu & Setiono 1996b] garantem que o subconjunto ótimo de atributos é encontrado se não houver restrição de tempo de execução do algoritmo. Minha opinião é que a afirmação dos autores é excessiva, dado que no processo de seleção de atributos o espaço de busca cresce exponencialmente. Quando o espaço de busca é bastante grande o Las Vegas se torna computacionalmente inviável, da mesma forma que uma busca completa, já que não utiliza qualquer heurística de busca para auxiliar no processo.

Por este motivo, o chamado *Las Vegas Wrapper* (LVW) não é recomendado para situações com um número de atributos muito grande ou integrado a um algoritmo de aprendizado muito custoso computacionalmente, como alguns experimentos descritos na Seção 3.4 sugerem. O pseudo-código do algoritmo Las Vegas é mostrado na Figura 3.8.

```
las_vegas(K, melhor)

k ← 0;                                  /* k armazena a quantidade de subconjuntos
                                           gerados sem haver melhoria */
melhor ← emptySet( );                   /* melhor recebe um subconjunto vazio */
atMelhor ← numAtributos(melhor);        /* atMelhor armazena o número de atributos
                                           selecionados no subconjunto melhor */
repita
    v ← randomSet( );                   /* v recebe um subconjunto aleatório */
    at ← numAtributos(v);               /* at armazena o número de atributos */
                                           selecionados no subconjunto v */
    se (f(v) > f(melhor)) ou (f(v) = f(melhor) e at < atMelhor) então
                                /* Se a qualidade de v é maior que do subconjunto melhor
                                   ou é igual, mas com menos atributos selecionados... */
        melhor ← v;             /* ... o subconjunto melhor passa a ser v */
        atMelhor ← at;          /* ... e o número de atributos selecionados é atualizado */
        imprime ("Melhor até então:" + melhor);     /* melhor é impresso */
        k ← 0;                                      /* k é zerado, pois houve melhora */
    fim se
    senão               /* caso contrário... */
        k ← k + 1;      /* ...incrementa o contador de subconjuntos gerados k */
    fim senão
enquanto (k < K)        /* repetir a operação enquanto o número de subconjuntos gerados
                           sem melhoria for menor que o valor K, definido inicialmente */
retorna (melhor);
```

Figura 3.8. Pseudo-código do método de busca Las Vegas.

3.3.5 Algoritmo Genético

Os AGs têm sido usados na busca por subconjuntos de atributos em diversos estudos realizados com *wrappers* [Chang & Lipmann 1991] [Guerra-Salcedo et al. 1999] [Vafaie & Imam 1994] [Yang & Honavar 1998]. Estes estudos mostram que os AGs são ótimos métodos de busca para a seleção de atributos, sobretudo quando o espaço de busca é definido por um número elevado de atributos. Os principais conceitos relativos a AGs estão no Apêndice A desta dissertação.

Na seleção de atributos, os cromossomos dos AGs correspondem aos subconjuntos que, a cada geração, sofrem seleção, cruzamento e mutação, a fim de gerar subconjuntos com maior precisão de classificação. Nos experimentos realizados neste trabalho de pesquisa, foi utilizada seleção do tipo roleta com elitismo de 1, cruzamento de um ponto e mutação simples.

3.4 Experimentos e Resultados

Nesta seção são apresentados os experimentos realizados com os *wrappers*, envolvendo dois métodos de aprendizado e cinco métodos de busca, dos quais dois deles apresentam três diferentes estratégias de direção da busca (e um apresenta três diferentes estados iniciais). Para todos os algoritmos que envolvem condições randômicas (métodos de busca RBC, Las Vegas e AG; estratégia *Random*) os experimentos foram repetidos dez vezes e os valores apresentados correspondem à média aritmética destas execuções. As onze bases de dados utilizadas estão descritas na Subseção 3.4.1.1. No total foram realizados mais de 1600 experimentos com os *wrappers*.

3.4.1 Método de Avaliação e *Overfitting*

Nos experimentos realizados com a finalidade de comparar a eficiência real de cada método de seleção de atributos, é importante utilizar a boa prática de dividir os dados disponíveis em duas partes [Reunanen 2003], uma para a realização do processo de seleção e outra para a validação desta seleção. Isto deve ser feito para verificar se o subconjunto de atributos selecionado apresenta bons resultados também com dados não usados pelo método de seleção, tanto na indução do conceito quanto na classificação.

O problema do *overfitting*, definido em [Cunningham 2000] como o problema do mecanismo de aprendizado se adaptar às peculiaridades do conjunto de treinamento em detrimento da capacidade de generalização, muitas vezes não teve a devida atenção em diversas pesquisas relacionadas à seleção de atributos, conforme apontado em [Kohavi & Sommersfeld 1995] [Kohavi & John1997] e [Loughrey & Cunningham 2005].

De fato, alguns trabalhos, como [Yang & Honavar 1998] não separam uma parte do conjunto de dados para a validação da seleção de atributos, levando a resultados muito otimistas quanto à melhoria da precisão de classificação obtida com a seleção de atributos.

Nota-se que a separação de uma parte dos dados para validação é necessária apenas quando os métodos seleção de atributos estão sendo avaliados quanto à sua capacidade[4]. No caso dos *wrappers*, o conjunto de dados é, em geral, separado inicialmente em dois

[4] Quando um método de seleção de atributos é utilizado na prática recomenda-se a utilização de todos os dados disponíveis para a realização do processo seleção de atributos, desde que o método seja comprovadamente adequado ao domínio em questão.

conjuntos, um para a avaliação da qualidade dos subconjuntos de atributos durante a busca (utilizado pelo método de seleção de atributos) e outro para a validação do subconjunto de atributos final (selecionado pelo método), que não participa do processo de seleção de atributos. O conjunto utilizado pelo método de seleção de atributos é também dividido para estimar a qualidade de classificação, podendo ser utilizada a n-validação cruzada ou qualquer outro método de estimativa de precisão.

Nos experimentos realizados nesta dissertação, foi adotado o esquema proposto em [Reunanen 2003], onde metade das instâncias disponíveis são separadas para a validação e a outra metade participa do processo de seleção dos atributos. Esta divisão (feita randomicamente), apesar de reduzir significativamente o conjunto de dados utilizado para a seleção de atributos, garante que o processo de validação terá uma quantidade considerável de instâncias para ser realizada (desde que o conjunto original tenha um número suficiente de instâncias). A Figura 3.9 mostra um diagrama do esquema adotado.

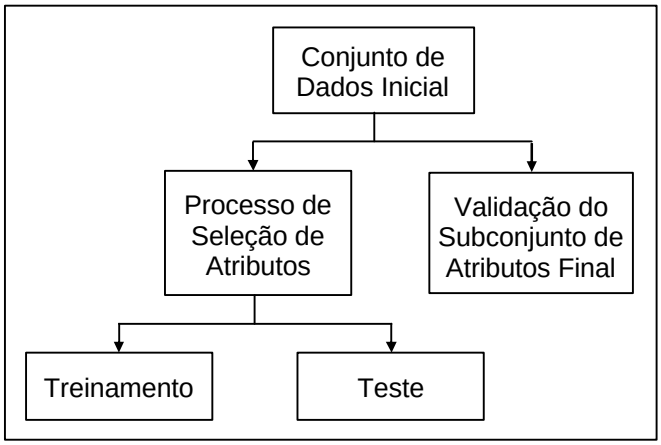


Figura 3.9. O esquema de avaliação utilizado na avaliação dos *wrappers*.

A n-validação cruzada foi utilizada tanto durante o processo de seleção quanto no de validação. Para o processo de seleção foi utilizada a 5-validação cruzada e na validação foi utilizada a 10-validação cruzada. Esta diferença nos métodos de estimativa de precisão se justifica devido ao interesse na redução da carga computacional durante o processo de busca pelo melhor subconjunto.

Outra consideração importante é que o método de estimativa de precisão utilizado na seleção dos atributos é totalmente independente da validação, já que a validação só é realizada para verificar se o subconjunto encontrado apresenta bons resultados também com dados que não participaram do processo de seleção de atributos (aprendizado).

Neste caso, optou-se por utilizar a 10-validação cruzada a fim de garantir maior acurácia na estimativa de precisão. Os dados relativos à precisão dos algoritmos de aprendizado disponibilizados nas tabelas de desempenho são referentes à validação utilizando 10-validação cruzada, o mesmo método de avaliação utilizado para avaliar a precisão do conjunto com todos os atributos (utilizando o mesmo conjunto de instâncias).

Nas tabelas de desempenho apresentadas nesta seção, os valores em negrito indicam os casos onde a precisão de classificação foi igual ou superior àquela obtida com o conjunto original de atributos. As tabelas que mostram a quantidade de atributos selecionados possuem os métodos com menor número de atributos destacados em itálico.

Na Figura 3.10 é apresentado um esquema geral dos *wrappers*, incluindo o processo de avaliação final, que utiliza um conjunto de validação que não participa do processo de seleção de atributos. O *wrapper* em si é envolvido pela caixa identificada como *Wrapper* que inclui um método de busca, que gera subconjuntos de atributos, e um método de AM que utiliza 5-validação cruzada para avaliar os subconjuntos gerados. Os subconjuntos de atributos selecionados são representados entre colchetes.

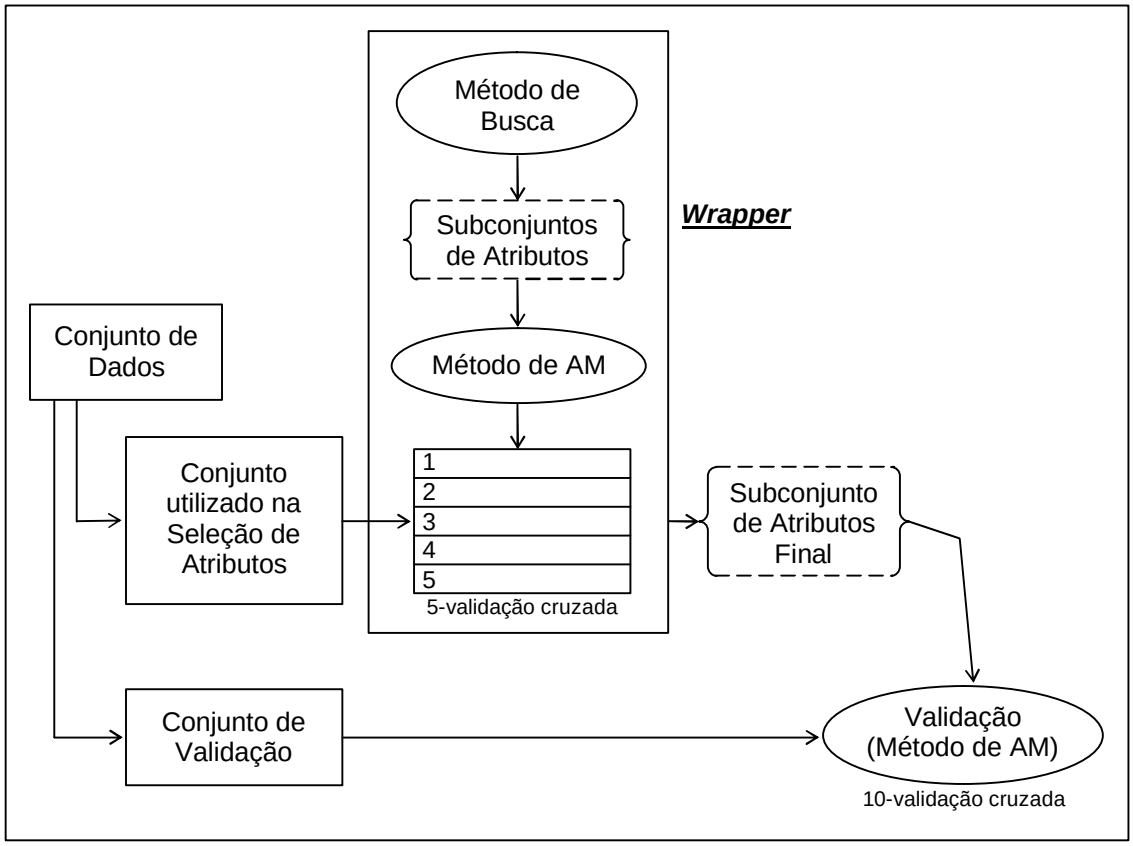


Figura 3.10. O esquema de avaliação utilizado pelos *wrappers*.

3.4.1.1 Bases de Dados Utilizadas nos Experimentos

Todas as bases de dados (*databases*) foram extraídas do UCI Machine Learning Repository [Blake & Merz 1998]. Foram escolhidas bases de dados com características bastante distintas, tanto com poucos quanto com muitos atributos e com o número de classes entre 2 e 7. Sem exceção, todas as bases de dados possuem atributos numéricos (contínuos e/ou discretos).

Na Tabela 3.1 são apresentadas as principais características das onze bases de dados empregadas nos experimentos e, em seguida, elas são brevemente descritas. Como comentado na Subseção 3.4.1, as bases de dados foram divididas em duas partes, uma para a seleção de atributos e outra para a validação. Para isso, foram separadas 50% das instâncias para a seleção e 50% para a validação, salvo nos casos onde o número de instâncias superava 2000, onde o número de instâncias para a seleção foi limitado em 1000, de forma a reduzir o tempo computacional exigido no processo de seleção de atributos.

Nos experimentos que seguem, as bases de dados foram agrupadas usando como critério o número de atributos que as descrevem, em:

- Baixa dimensionalidade (4-15): aquela que tem um número de atributos entre 4 e 15;
- Média dimensionalidade (16-34): aquela que tem um número de atributos entre 16 e 34;
- Alta dimensionalidade (60): aquela que tem entre 35 e 60 atributos (apenas uma base).

A escolha das bases de dados procurou contemplar diversidade em relação a número de atributos e número de classes.

Tabela 3.1. Resumo das bases de dados utilizadas nos experimentos com *wrappers*.

Database	Instâncias			Atributos	Classes
	Total	Seleção	Validação		
Iris	150	75	75	4	3
Breast	683	341	342	10	2
Glass	214	107	107	10	6
Wine	178	89	89	13	3
Vehicle	846	423	423	18	4
Segment	2310	1000	1310	19	7
Waveform	5000	1000	4000	21	3
WDBC	569	284	285	30	2
WPBC	194	97	97	33	2
Ionosphere	351	175	176	34	2
Sonar	208	104	104	60	2

Íris: base de dados com apenas 4 atributos para a classificação de 3 diferentes flores. É amplamente utilizada em trabalhos de AM e estatística, sendo datada de 1936. Possui 2 das 3 classes linearmente separáveis. Os atributos são numéricos e possuem certa redundância.

Breast: base de dados com 10 atributos para a identificação de câncer de mama (benigno ou maligno). Um dos atributos é um número de identificação do exame (possivelmente irrelevante). Das 699 instâncias originais, foram excluídas 16, por conterem atributos ausentes.

Glass: base de dados com 10 atributos para a classificação de 6 tipos de vidro. Os atributos são referentes à composição do vidro, refração e número de identificação (possivelmente irrelevante).

Wine: base de dados com 13 atributos para a classificação de 3 tipos de vinho, relativos à localização da região de cultivo. Os atributos são referentes às análises químicas dos vinhos.

Vehicle: base de dados com 18 atributos para a classificação de 4 tipos de veículos a partir de uma silhueta. Os atributos são referentes às propriedades geométricas das silhuetas.

Segment: base de dados com 19 atributos para a segmentação de imagens em 7 possíveis classes. Os atributos contêm informações sobre uma região da imagem (3x3), que deve ser classificada para a posterior segmentação.

Waveform: base de dados com 21 atributos para a classificação de 3 diferentes ondas, geradas por computador. São 5000 instâncias e os valores podem ter ruído.

WDBC: base de dados com 30 atributos para o diagnóstico de câncer de mama (maligno ou benigno). São 10 características das células (como área, raio do núcleo, perímetro, etc), com média, desvio padrão e maior caso para cada uma delas (informações disponíveis devido à metodologia adotada para a extração dos dados).

WPBC: base de dados com 33 atributos para o prognóstico de câncer de mama (recorrente ou não-recorrente). Entre as características das células (como área, raio do núcleo, perímetro, etc), 10 delas possuem média, desvio padrão e maior caso (informações disponíveis devido à metodologia adotada para a extração dos dados).

Ionosphere: base de dados com 34 atributos para identificar se um radar capta sinais da ionosfera válidos ou não. São informações sobre 17 características com dois atributos para representar cada uma (portanto, com diversos atributos potencialmente irrelevantes).

Sonar: base de dados com 60 atributos extraídos de um sonar para a classificação de minas (metálicas) ou pedras. Os atributos contêm informações sobre a intensidade energética de respostas a diferentes faixas de freqüência.

3.4.2 A Família de *Wrappers* NN

Nesta subseção estão descritos os experimentos e discutidos os resultados dos *wrappers* que implementam o *Nearest Neighbor* (NN) como algoritmo de aprendizado. O chamado Wrapper NN deu origem a diversas variantes devido aos diferentes métodos de busca e estratégias de direção empregados. As próximas subseções são relativas às variantes criadas considerando cada um dos cinco métodos de busca investigados neste trabalho. A Figura 3.11 mostra o conjunto de variantes utilizando o NN que foram avaliadas.

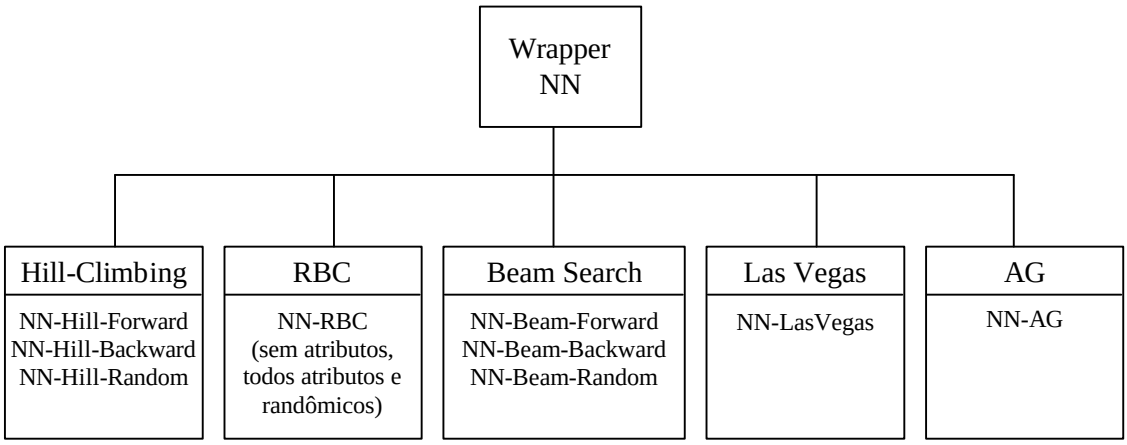


Figura 3.11. Conjunto de variantes dos *wrappers* integrados ao NN.

3.4.2.1 Hill-Climbing

Nas Tabelas 3.2 e 3.3 são mostrados os resultados dos experimentos de avaliação de cada uma das estratégias (*Forward*, *Backward* e *Random*) usando o NN como algoritmo de aprendizado e o *Hill-Climbing* como método de busca, referenciados como as variantes NN-Hill-Forward, NN-Hill-Backward e NN-Hill-Random.

Estes experimentos, realizados com as bases de dados descritas na Subseção 3.4.1.1 mostram que, na média não há melhoria significativa na precisão de classificação em relação ao conjunto original de atributos utilizando nenhuma das estratégias em conjunto com o *Hill-Climbing*. Com as variantes NN-Hill-Backward e NN-Hill-Random há, na média, uma sensível melhoria, como pode ser evidenciado na Tabela 3.2; já com a variante NN-Hill-Forward há uma degradação da precisão de classificação. De fato a média geral desta variante é reduzida por desempenhos bastante indesejáveis nas bases de dados Sonar, Wine e Breast, o que de certa forma reflete deficiências da estratégia *Forward* associada com o *Hill-Climbing*.

Tabela 3.2 Desempenho do *Hill-Climbing* com o NN.

Database	Todos Atributos	NN-Hill-Forward	NN-Hill-Backward	NN-Hill-Random
Iris	94,67 ± 6,94	**94,67** ± 6,46	**94,67** ± 6,46	**94,67** ± 6,46
Breast	96,20 ± 3,68	91,81 ± 4,77	95,03 ± 2,00	94,42 ± 2,52
Glass	91,59 ± 8,23	**94,39** ± 4,94	87,85 ± 9,92	**92,06** ± 6,89
Wine	97,75 ± 4,99	88,76 ± 10,83	93,26 ± 7,87	94,49 ± 6,96
Vehicle	69,74 ± 10,40	67,14 ± 8,53	**70,21** ± 8,99	67,38 ± 9,17
Segment	97,02 ± 1,46	**97,25** ± 0,82	**97,48** ± 1,25	**97,13** ± 0,99
Waveform	77,15 ± 2,09	76,03 ± 1,30	76,93 ± 1,62	75,15 ± 1,82
WDBC	95,44 ± 3,71	**96,14** ± 2,61	**96,14** ± 3,07	95,12 ± 3,99
WPBC	63,92 ± 14,72	**72,16** ± 16,80	**73,20** ± 6,90	**72,78** ± 14,04
Ionosphere	88,64 ± 5,37	**90,91** ± 4,88	**88,64** ± 5,52	**91,31** ± 7,13
Sonar	80,77 ± 8,10	61,54 ± 12,62	**80,77** ± 12,66	78,94 ± 11,93
Média	**86,63**	**84,62**	**86,74**	**86,68**
(4-15)	95,05	92,41	92,70	93,91
(16-34)	81,99	83,27	83,77	83,15
(60)	80,77	61,54	80,77	78,94

Como os resultados evidenciam, a NN-Hill-Forward não se mostrou uma boa alternativa quando se deseja maximizar a precisão de classificação. Já quando o foco é minimizar o número de atributos, os resultados com a NN-Hill-Forward foram mais interessantes quando comparados aos da NN-Hill-Backward e NN-Hill-Random, como mostra a Tabela 3.3. A mesma tendência de valores menores pode ser notada com relação ao número de estados visitados (subconjuntos avaliados), que contribui para um menor tempo de processamento.

Tabela 3.3. Atributos selecionados e número de avaliações no *Hill-Climbing* com o NN.

Database	Original	NN-Hill-Forward		NN-Hill-Backward		NN-Hill-Random	
	Atribs	**Atribs**	Avals	**Atribs**	Avals	**Atribs**	Avals
Iris	4	*2*	9	*2*	9	*2,0*	8,6
Breast	10	*3*	34	8	27	5,3	32,2
Glass	10	*3*	34	7	34	5,2	43,5
Wine	13	*2*	36	8	63	6,2	39,8
Vehicle	18	*6*	105	16	51	10,3	79,9
Segment	19	*4*	85	14	99	8,2	134,3
Waveform	21	*9*	165	17	95	14,7	121,7
WDBC	30	*4*	140	18	312	13,4	172,6
WPBC	33	*1*	65	26	236	16,0	176,7
Ionosphere	34	*3*	130	24	319	13,8	208,0
Sonar	60	*4*	290	39	1089	28,6	451,6
Média	**22,9**	**3,7**	**99,4**	**16,3**	**212,2**	**11,2**	**133,5**
(4-15)	9,3	2,5	28,3	6,3	33,3	4,7	31,0
(16-34)	25,8	4,5	115,0	19,2	185,3	12,7	148,9
(60)	60,0	4,0	290,0	39,0	1089,0	28,6	451,6

A tendência da NN-Hill-Forward de avaliar poucos estados sugere uma deficiência do método *Hill-Climbing* em escapar de máximos locais. Esta deficiência impede que, quando a NN-Hill-Forward é utilizada, atributos inter-relacionados, isto é, que combinados produzem melhores resultados do que separadamente, sejam detectados e selecionados. Assim, a busca acaba por avaliar poucos subconjuntos, pois logo fica presa em um máximo local.

Uma prova experimental desta deficiência que ocorre com a NN-Hill-Forward pode ser notada na base de dados Sonar, onde apenas quatro atributos são selecionados de um total de 60, levando a uma precisão de classificação bastante reduzida em relação ao original (caiu de 80% para 61%). Intuitivamente conclui-se que a busca logo é interrompida por chegar a um máximo local, onde nenhum subconjunto com um atributo a mais do que aqueles quatro selecionados otimiza a precisão de classificação.

Entre a NN-Hill-Backward e a NN-Hill-Random, é possível observar uma maior consistência na NN-Hill-Backward, que obtém bons resultados em 7 das 11 bases de dados utilizadas. Já a NN-Hill-Random melhorou ou manteve a precisão em menos da metade das bases de dados utilizadas nos experimento (5 das 11). Por outro lado, o número de atributos selecionados é invariavelmente menor na NN-Hill-Random frente à NN-Hill-Backward. Quanto aos estados explorados a NN-Hill-Random se mostrou mais favorável, sobretudo em bases de dados com dimensionalidade média ou alta.

A otimização da precisão de classificação nestes experimentos só foi significativa na base de dados WPBC. Por outro lado, apenas na base de dados Wine houve degradação significativa na precisão de classificação em todos os experimentos, mesmo havendo redução dos atributos em todos os experimentos realizados.

Conclui-se, portanto, que a NN-Hill-Backward é a variante que se mostrou mais interessante quando se opta por utilizar o método de busca *Hill-Climbing* para a seleção de atributos, já que leva a subconjuntos com maior precisão de classificação que, em geral, otimizam a precisão de classificação. A NN-Hill-Random, no entanto, é preferível quando uma maior redução do número de atributos é necessária.

3.4.2.2 Random Bit Climber

Nas Tabelas 3.4 e 3.5 são mostrados os resultados dos experimentos de avaliação do método de busca RBC e do algoritmo de aprendizado NN utilizando três diferentes estados iniciais, todos os atributos, nenhum atributo e um subconjunto de atributos randomicamente selecionados, sendo eles referenciados, respectivamente, como NN-RBC-TA, NN-RBC-NA e NN-RBC-SAR.

Dentre as três combinações com os diferentes estados iniciais, os melhores resultados foram obtidos com a NN-RBC-SAR (melhorou ou manteve a precisão em 5 das 11 bases de dados), com uma média de classificação superior tanto em relação à NN-RBC-TA, que melhorou ou manteve a precisão em 3 das 11 bases de dados, por uma pequena margem, quanto em relação à NN-RBC-NA, que melhorou ou manteve a precisão em 3 das 11 bases de dados, por uma margem mais significativa.

A média geral de precisão foi otimizada em relação ao conjunto original de atributos tanto na NN-RBC-SAR quanto na NN-RBC-TA. Não houve experimentos com degradação significativa da precisão, com exceção daqueles usando a base de dados Wine, onde as combinações NN-RBC-NA e NN-RBC-TA foram incapazes de reduzir o conjunto original de atributos sem degradar a precisão.

Tabela 3.4. Desempenho do *Random Bit Climber* com o NN.

Database	Todos Atributos	NN-RBC-NA	NN-RBC-TA	NN-RBC-SAR
Iris	94,67 ± 6,94	**94,67** ± 6,46	**94,67** ± 6,46	**94,67** ± 6,46
Breast	96,20 ± 3,68	94,44 ± 2,55	95,03 ± 2,38	95,15 ± 2,29
Glass	91,59 ± 8,23	89,53 ± 11,06	90,19 ± 7,87	**92,52** ± 6,78
Wine	97,75 ± 4,99	91,57 ± 8,80	92,81 ± 5,52	95,06 ± 5,97
Vehicle	69,74 ± 10,40	69,10 ± 9,62	69,43 8,81	67,87 ± 9,28
Segment	97,02 ± 1,46	95,96 ± 1,02	96,33 ± 1,01	96,07 ± 1,02
Waveform	77,15 ± 2,09	75,73 ± 2,04	76,92 ± 2,08	75,77 ± 2,09
WDBC	95,44 ± 3,71	94,53 ± 3,94	94,98 ± 4,43	**95,72** ± 3,95
WPBC	63,92 ± 14,72	**69,69** ± 11,61	**74,02** ± 12,14	**73,81** ± 12,09
Ionosphere	88,64 ± 5,37	**90,63** ± 7,26	**91,25** ± 6,60	**91,25** ± 6,47
Sonar	80,77 ± 8,10	75,00 ± 14,12	78,85 ± 12,52	77,31 ± 12,91
Média	**86,63**	**85,53**	**86,77**	**86,84**
(4-15)	95,05	92,55	93,18	94,35
(16-34)	81,99	82,61	83,82	83,42
(60)	80,77	75,00	78,85	77,31

Considerando o número de atributos selecionados, pode ser observado na Tabela 3.5 que o NN-RBC-NA obteve subconjuntos menores na maioria dos casos, da mesma forma que ocorreu com o NN-Hill-Forward em relação às outras variantes que utilizam o *Hill-Climbing* (variante que, da mesma forma que o NN-RBC-NA, utiliza como estado inicial um conjunto vazio de atributos).

Tabela 3.5. Atributos selecionados e número de avaliações no *Random Bit Climber* com o NN.

Database	Original	NN-RBC-NA		NN-RBC-TA		NN-RBC-SAR	
	Atribs	**Atribs**	Avals	**Atribs**	Avals	**Atribs**	Avals
Iris	4	*2,0*	12,0	*2,0*	12,0	*2,0*	12,4
Breast	10	6,1	32,0	*6,0*	31,0	6,4	26,0
Glass	10	6,5	20,0	6,0	44,0	*4,2*	37,0
Wine	13	*6,0*	39,0	7,2	58,5	*6,0*	28,6
Vehicle	18	*10,0*	61,2	14,7	43,2	10,7	54,0
Segment	19	6,9	30,0	8,0	20,0	*6,0*	34,3
Waveform	21	14,3	75,6	17,0	58,8	*13,4*	56,7
WDBC	30	*8,1*	96,0	14,4	108,0	11,4	114,0
WPBC	33	*5,3*	92,4	21,6	105,6	14,2	112,2
Ionosphere	34	*9,0*	108,8	15,1	122,4	11,2	136,0
Sonar	60	*11,5*	162,0	30,2	246,0	22,9	204,0
Média	**22,9**	**7,8**	**66,3**	**12,9**	**77,2**	**9,9**	**74,1**
(4-15)	9,3	5,2	25,8	5,3	36,4	4,7	26,0
(16-34)	25,8	8,9	77,3	15,1	76,3	11,2	84,5
(60)	60,0	11,5	162,0	30,2	246,0	22,9	204,0

A quantidade de estados avaliados não apresentou grandes diferenças entre as três combinações; já com relação ao tempo de processamento o NN-RBC-NA foi em média mais veloz, devido à utilização de menos atributos durante a avaliação da qualidade dos subconjuntos.

Comparando o RBC ao *Hill-Climbing*, nota-se a superioridade do RBC que, além de obter uma média de precisão melhor, obteve os resultados avaliando menos subconjuntos que o *Hill-Climbing*, o que caracteriza uma busca mais eficaz.

A maior precisão de classificação pode ser explicada devido à menor tendência do RBC a ficar preso em máximos locais, como foi verificado com o *Hill-Climbing*. Esta característica é evidenciada notando que a NN-RBC-NA seleciona um número maior de atributos que a NN-Hill-Forward; já a NN-RBC-TA remove um número maior de atributos que a Hill-Backward-NN, resultando em um menor subconjunto de atributos. Esses resultados sugerem que o RBC não tende a encontrar máximos locais tão prematuramente quanto o *Hill-Climbing.*

A NN-RBC-SAR mostrou-se bastante eficaz mesmo quando comparada com métodos mais elaborados como o *Beam Search* ou Algoritmos Genéticos, sendo um boa opção de seleção de atributos, sobretudo quando é necessário minimizar o número de estados avaliados (por exemplo, quando o algoritmo de aprendizado é computacionalmente custoso).

3.4.2.3 Beam Search

Os resultados obtidos com o método de busca *Beam Search* utilizando o algoritmo de aprendizado NN em conjunto com as três estratégias propostas (*Forward*, *Backward* e *Random*) estão nas Tabelas 3.6 e 3.7, referenciadas como as variantes NN-Beam-Forward, NN-Beam-Backward e NN-Beam-Random. O tamanho do beam utilizado foi 25 e o número de expansões sem melhoria (K) foi 50.

Como pode ser visto na Tabela 3.6, as três estratégias de busca utilizadas em combinação com o método *Beam Search* apresentaram resultados equilibrados, quando considerada a média de precisão final e superaram a média de precisão do conjunto original de atributos. O NN-Beam-Backward obteve o melhor resultado quando considerada apenas a média de precisão (melhorou ou manteve a precisão em 7 das 11

bases de dados), sendo também o melhor método quanto à precisão dentre todos os *wrappers* utilizando o método de aprendizado NN avaliados nesta dissertação. O NN-Beam-Forward melhorou ou manteve a precisão em 7 das 11 bases de dados, mas na média foi similar ao NN-Beam-Random, que melhorou ou manteve a precisão em 6 das 11 bases de dados.

Tabela 3.6. Desempenho do *Beam Search* com o NN.

Database	Todos Atributos	NN-Beam-Forward	NN-Beam-Backward	NN-Beam-Random
Iris	94,67 ± 6,94	**94,67** ± 6,46	**97,33** ± 5,27	**96,80** ± 5,51
Breast	96,20 ± 3,68	94,74 ± 2,33	94,74 ± 2,33	94,97 ± 2,58
Glass	91,59 ± 8,23	**92,52** ± 5,83	**92,52** ± 5,83	**92,52** ± 5,83
Wine	97,75 ± 4,99	93,26 ± 10,73	93,26 ± 7,87	94,94 ± 7,13
Vehicle	69,74 ± 10,40	**70,45** ± 10,38	68,79 ± 8,79	68,79 ± 8,79
Segment	97,02 ± 1,46	**97,18** ± 1,02	**97,18** ± 0,89	**97,21** ± 0,94
Waveform	77,15 ± 2,09	76,03 ± 1,77	76,03 ± 1,77	76,03 ± 1,77
WDBC	95,44 ± 3,71	**96,49** ± 2,30	**95,44** ± 3,31	**95,51** ± 4,13
WPBC	63,92 ± 14,72	**84,54** ± 14,85	**85,57** ± 10,67	**82,37** ± 11,30
Ionosphere	88,64 ± 5,37	**90,34** ± 6,57	**93,18** ± 5,08	**89,72** ± 7,16
Sonar	80,77 ± 8,10	78,85 ± 12,61	**83,65** ± 13,07	79,52 ± 12,34
Média	**86,63**	**88,10**	**88,88**	**88,03**
(4-15)	95,05	93,80	94,46	94,81
(16-34)	81,99	85,84	86,03	84,94
(60)	80,77	78,85	83,65	79,52

Os bons resultados com o *Beam Search, n*o entanto, são obtidos com um grande esforço computacional. O exemplo mais evidente deste esforço computacional pode ser visto quando utilizada a base de dados Sonar, onde foram avaliados mais de 6500 subconjuntos, um número alto e pouco interessante para um algoritmo de aprendizado com alto custo computacional (não o caso do NN).

Como pode ser visto na Tabela 3.7, as diferenças mais representativas entre as variantes do *Beam Search* aparecem em relação ao número de atributos selecionados, com a NN-Beam-Forward selecionando menos atributos, sobretudo quando a base de dados apresenta dimensionalidade média ou alta. Analisando estes dados, conclui-se que caso seja desejável um número mínimo de atributos selecionados a NN-Beam-Forward é recomendado, garantindo ainda uma boa precisão de classificação.

Tabela 3.7. Atributos selecionados e número de avaliações no *Beam Search* com o NN.

Database	Original	NN-Beam-Forward		NN-Beam-Backward		NN-Beam-Random	
	Atribs	**Atribs**	Avals	**Atribs**	Avals	**Atribs**	Avals
Iris	4	*2*	16	3	16	2,8	16,0
Breast	10	9	435	9	365	*7,2*	383,0
Glass	10	*5*	418	*5*	437	*5,0*	435,0
Wine	13	*6*	651	8	653	7,8	633,0
Vehicle	18	*11*	1533	13	881	13,0	1168,3
Segment	19	*9*	1229	*9*	1400	9,8	1601,3
Waveform	21	*15*	2022	*15*	1287	*15,0*	1184,7
WDBC	30	*11*	3544	16	2499	13,3	2527,7
WPBC	33	*8*	1923	12	3624	11,8	2839,7
Ionosphere	34	*11*	3196	14	4523	11,5	3419,5
Sonar	60	*24*	6564	35	7500	28,4	6614,4
Média	**22,9**	**10,1**	**1957,4**	**12,6**	**2107,7**	**11,4**	**1893,0**
(4-15)	9,3	5,5	380,0	6,3	367,8	5,7	366,8
(16-34)	25,8	10,8	2241,2	13,2	2369,0	12,4	2123,5
(60)	60,0	24,0	6564,0	35,0	7500,0	28,4	6614,4

A NN-Beam-Random é uma alternativa que avaliou um número de subconjuntos similar a NN-Beam-Forward (apenas ligeiramente inferior), mesmo iniciando a busca a partir de um estado com alguns atributos selecionados. A sua precisão média não apresentou diferenças significativas comparada às das duas outras variantes.

No geral o método *Beam Search* apresentou bons resultados, sendo recomendado para a otimização da precisão de classificação quando o número original de atributos não é superior a 50, que caracteriza um problema com $1{,}13 \times 10^{15}$ possíveis estados. Para números de atributos maiores que 50, o número de subconjuntos de atributos avaliados passa a ser muito grande, sendo mais recomendável um método de busca menos exaustivo tal como o RBC ou um Algoritmo Genético, a não ser que haja grande poder computacional disponível.

3.4.2.4 Las Vegas Wrapper

O Las Vegas Wrapper utilizando o NN como algoritmo de aprendizado, referenciado como NN-LasVegas, não apresenta variações quanto à estratégia de busca, já que a busca Las Vegas se caracteriza por ser randômica, não se encaixando nas estratégias *Forward* ou *Backward*. Assim, os dados obtidos só podem ser comparados frente aos outros métodos de busca. O número máximo de subconjuntos gerados sem haver melhoria foi definida como $60 \times$ atributos, variando, portanto, para cada base de dados.

Com o NN-LasVegas os resultados foram satisfatórios quando considerada apenas a precisão de classificação obtida (melhorou ou manteve a precisão em 6 das 11 bases de dados), que superou em média a precisão obtida com o conjunto original de atributos. Como discutido na Subseção 3.3.4, o Las Vegas é um algoritmo que utiliza força bruta, gerando subconjuntos de atributos randomicamente, sem qualquer heurística para otimizar o processo. Por isso, o número de avaliações é extremamente alto (como pode ser verificado na Tabela 3.8), mesmo quando o espaço de busca é pequeno. No caso de espaços de busca maiores, o número de avaliações muitas vezes supera o do método *Beam Search*, que mostrou melhor capacidade de otimizar a precisão de classificação.

Tabela 3.8. Desempenho do Las Vegas com o NN.

Database	Original		NN-LasVegas		
	Atribs	Precisão	Atribs	Precisão	Avals
Iris	4	94,67 ± 6,94	2,0	**94,67** ± 6,46	254,7
Breast	10	96,20 ± 3,68	5,0	95,00 ± 2,88	1067,9
Glass	10	91,59 ± 8,23	4,0	**93,74** ± 5,51	895,0
Wine	13	97,75 ± 4,99	7,1	94,61 ± 6,80	1185,5
Vehicle	18	69,74 ± 10,40	13,8	69,22 ± 8,74	2132,7
Segment	19	97,02 ± 1,46	10,6	**97,11** ± 1,03	2275,3
Waveform	21	77,15 ± 2,09	18,3	76,57 ± 1,85	1923,9
WDBC	30	95,44 ± 3,71	14,9	**96,00** ± 3,54	3012,9
WPBC	33	63,92 ± 14,72	7,6	**76,80** ± 12,31	3285,9
Ionosphere	34	88,64 ± 5,37	9,0	**89,49** ± 7,46	3728,7
Sonar	60	80,77 ± 8,10	26,1	77,40 ± 13,45	6936,8
Média	**44,2**	**86,63**	**10,8**	**87,33**	**2427,2**
(4-15)	9,3	95,05	4,5	94,51	850,8
(16-34)	25,8	81,99	12,4	84,20	2726,6
(60)	60,0	80,77	26,1	77,40	6936,8

A melhor característica do LasVegas para estas bases de dados é a sua capacidade de encontrar bons subconjuntos com poucos atributos, já que a média de atributos selecionados só é superior aos métodos que utilizam a estratégia *Forward*; no caso da NN-Beam-Forward, o LVW é inferior tanto em relação à precisão quanto em número de atributos ou número de avaliações realizadas.

Assim sendo, o NN-LasVegas não pode ser considerado um método muito atraente devido ao seu alto custo computacional que não reflete um desempenho otimizado, como acontece no caso do *Beam Search*.

3.4.2.5 Algoritmo Genético

Assim como o NN-LasVegas, o Algoritmo Genético (AG) utilizando o NN como algoritmo de aprendizado, referenciado como NN-AG, só pode ser comparado com os demais métodos de busca que utilizam o NN. A característica mais importante a respeito do AG é que ele utiliza um número fixo e não muito alto de avaliações (1000, com a população de 50 cromossomos e 20 gerações). Mesmo assim os resultados são superiores ao NN-LasVegas, que também melhorou ou manteve a precisão em 6 das 11 bases de dados, conforme mostram os resultados da Tabela 3.9.

Assim, o AG pode ser considerado um dos métodos de busca mais eficientes para ser empregado nos *wrappers*, principalmente quando as bases de dados possuem um maior número de atributos.

Tabela 3.9. Desempenho do Algoritmo Genético com o NN.

Database	Original		NN-AG	
	Atribs	Precisão	Atribs	Precisão
Iris	4	94,67 ± 6,94	2,0	**94,67** ± 6,46
Breast	10	96,20 ± 3,68	4,0	95,03 ± 2,75
Glass	10	91,59 ± 8,23	4,8	**92,71** ± 5,74
Wine	13	97,75 ± 4,99	6,9	93,93 ± 7,88
Vehicle	18	69,74 ± 10,40	12,9	68,82 ± 8,61
Segment	19	97,02 ± 1,46	9,1	**97,18** ± 1,08
Waveform	21	77,15 ± 2,09	16,5	76,58 ± 2,09
WDBC	30	95,44 ± 3,71	12,2	**96,04** ± 3,24
WPBC	33	63,92 ± 14,72	9,7	**78,66** ± 11,69
Ionosphere	34	88,64 ± 5,37	10,8	**90,23** ± 6,41
Sonar	60	80,77 ± 8,10	30,1	79,42 ± 12,69
Média	**44,2**	**86,63**	**10,8**	**87,57**
(4-15)	9,3	95,05	4,4	94,09
(16-34)	25,8	81,99	11,9	84,59
(60)	60,0	80,77	30,1	79,42

3.4.2.6 Considerações sobre os *Wrappers* NN

Em geral o aprendizado usando o NN foi beneficiado pelo processo de seleção de atributos, havendo significativa redução no número de atributos na maioria das bases de dados. Quase não aconteceram problemas relativos à degradação da precisão; alguns dos problemas foram conseqüências das características específicas das bases de dados, tais como as bases Wine e Breast-Cancer, que possuem poucos atributos, a maioria aparentemente relevante (nenhum método de busca foi capaz de encontrar um

subconjunto que otimizasse ou mantivesse a precisão de classificação nessas bases). Outros problemas de degradação na precisão de classificação ocorreram devido a peculiaridades dos métodos/estratégias de busca que, especificamente nas bases Sonar e Wine, resultaram em poucos atributos selecionados (não representativos), que não forneceram bons resultados com o conjunto de validação.

Em relação ao tempo de processamento o NN não apresentou grandes problemas, mesmo no caso de buscas que envolveram mais de 6000 avaliações, onde o processo durou apenas dois minutos no sistema utilizado.

3.4.3 A Família de *Wrappers* DistAl

Nesta subseção estão descritos os experimentos e discutidos os resultados dos *wrappers* que implementam o DistAl como algoritmo de aprendizado. O chamado Wrapper DistAl deu origem a diversas variantes devido aos diferentes métodos de busca e estratégias de direção empregadas. As próximas subseções são relativas às variantes criadas considerando cada um dos cinco métodos de busca investigados neste trabalho. A Figura 3.12 mostra o conjunto de variantes que foram avaliadas.

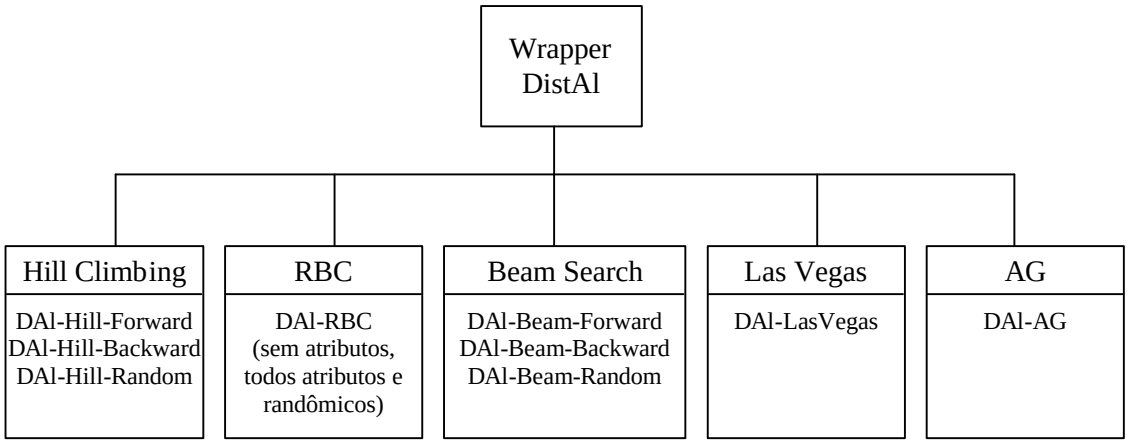


Figura 3.12. Conjunto de variantes dos *wrappers* integrados ao DistAl.

3.4.3.1 Hill-Climbing

Nas Tabelas 3.10 e 3.11 são mostrados os resultados dos experimentos de avaliação de cada uma das estratégias (*Forward*, *Backward* e *Random*) usando o DistAl como algoritmo de aprendizado e o *Hill-Climbing* como método de busca, referenciados como variantes DAl-Hill-Forward, DAl-Hill-Backward e DAl-Hill-Random.

Em geral, houve melhoria na classificação frente ao subconjunto original de atributos nas variantes DAl-Hill-Backward (melhorou ou manteve a precisão em 10 das 11 bases de dados) e DAl-Hill-Random (melhorou ou manteve a precisão em 7 das 11 bases de dados), como pode ser evidenciado na Tabela 3.10, sendo que com a DAl-Hill-Forward houve degradação da precisão de classificação em diversos casos (melhorou ou manteve a precisão em 6 das 11 bases de dados), o que reflete em uma média final bastante baixa.

Tabela 3.10. Desempenho do Hill-Climbing com o DistAl.

Database	Original	DAl-Hill-Forward	DAl-Hill-Backward	DAl-Hill-Random
Iris	92,86 ± 9,58	**95,71** ± 6,55	**95,71** ± 6,55	**95,71** ± 6,55
Breast	96,47 ± 2,56	**96,47** ± 2,56	**97,06** ± 2,94	**96,68** ± 2,46
Glass	74,00 ± 10,20	**95,00** ± 5,00	**84,00** ± 12,81	**87,40** ± 8,92
Wine	97,50 ± 5,00	91,25 ± 12,56	**98,75** ± 3,75	93,75 ± 6,97
Vehicle	54,76 ± 8,32	6,43 ± 14,13	**60,24** ± 7,61	**57,57** ± 6,40
Segment	88,93 ± 3,06	**93,89** ± 2,46	**92,67** ± 1,94	**92,88** ± 1,78
Waveform	85,13 ± 1,73	83,85 ± 1,25	**85,63** ± 1,55	83,18 ± 1,25
WDBC	96,07 ± 2,50	**96,07** ± 2,50	**96,43** ± 3,19	95,71 ± 3,41
WPBC	77,78 ± 14,91	58,89 ± 31,45	71,11 ± 15,87	77,00 ± 14,20
Ionosphere	88,82 ± 6,14	3,53 ± 10,59	**89,41** ± 5,76	**90,35** ± 6,28
Sonar	70,00 ± 12,65	**74,00** ± 8,00	**73,00** ± 7,81	**76,00** ± 11,44
Média	**83,85**	**72,28**	**85,82**	**86,02**
(4-15)	90,21	94,61	93,88	93,39
(16-34)	81,92	57,11	82,58	82,78
(60)	70,00	74,00	73,00	76,00

Desta forma, a busca *Hill-Climbing* com a estratégia *Forward* não se mostrou uma alternativa viável na integração com o DistAl, já que a preferência dessa busca/estratégia por subconjuntos mínimos freqüentemente acaba por degradar significativamente a precisão de classificação. Nas bases de dados Ionosphere e Vehicle houve grande degradação devido à seleção de apenas 2 atributos, que não se mostraram significativos para generalizar o conceito no conjunto de validação (o DistAl mostrou-se mais sensível que o NN em relação a seleção de atributos mal sucedida)

A mesma deficiência verificada a NN-Hill-Forward pode ser verificada aqui, que é a deficiência em escapar de máximos locais. A média de apenas 3,5 atributos selecionados, considerando todas as bases de dados reforça ainda mais esta deficiência do *Hill-Climbing* associado à estratégia *Forward*.

A DAl-Hill-Backward apresentou melhoria de precisão em praticamente todas as bases de dados, com destaque por ter sido um dos poucos métodos de busca (juntamente com o DAl-RBC-TA) a não escolher um subconjunto que reduzisse a precisão na base

de dados Wine, isso em razão da eliminação de apenas 3 atributos, como mostra a Tabela 3.11 (a deficiência de não escapar de máximos locais beneficiou o método). A média de atributos selecionados é extremamente alta, muito próxima da média dos conjuntos originais de atributos (em média foram removidos apenas 3,2 atributos de cada conjunto original de atributos).

Tabela 3.11. Atributos selecionados e número de avaliações no Hill-Climbing com o DistAl.

Database	Original	DAl-Hill-Forward		DAl-Hill-Backward		DAl-Hill-Random	
	Atribs	**Atribs**	Avals	**Atribs**	Avals	**Atribs**	Avals
Iris	4	*1*	7	3	7	1,4	8,6
Breast	10	*1*	19	6	40	3,7	34,0
Glass	10	*5*	45	7	34	5,2	28,4
Wine	13	*5*	63	10	46	7,3	47,6
Vehicle	18	*2*	51	16	51	9,9	74,9
Segment	19	*4*	85	11	135	9,0	107,7
Waveform	21	*10*	176	20	41	12,0	98,3
WDBC	30	*5*	165	28	87	14,0	135,1
WPBC	33	*1*	65	32	65	14,7	104,2
Ionosphere	34	*2*	99	28	217	16,2	132,1
Sonar	60	*3*	234	56	290	31,2	232,9
Média	**22,9**	**3,5**	**91,7**	**19,7**	**92,1**	**11,3**	**91,3**
(4-15)	9,3	3	33,5	6,5	31,8	4,4	29,7
(16-34)	27,4	4,4	118,0	23,8	109,0	13,2	115,5
(60)	60,0	3,0	234,0	56,0	290,0	31,2	232,9

Já a DAl-Hill-Random não otimizou tantas bases de dados, mas não degradou significativamente os resultados em nenhuma base de dados, sendo portanto uma opção interessante. O número de atributos selecionados foi muito menor que o selecionado pela DAl-Hill-Backward, em média a metade da quantidade de atributos do conjunto original. Esta característica torna a DAl-Hill-Random tão interessante quanto a DAl-Hill-Backward, sobretudo quando o objetivo é remover o maior número possível de atributos.

3.4.3.2 Random Bit Climber

Nas Tabelas 3.12 e 3.13 são mostrados os resultados dos experimentos de avaliação do método de busca RBC e do algoritmo de aprendizado DistAl, utilizando três diferentes estados iniciais, todos os atributos, nenhum atributo e um subconjunto de atributos randomicamente selecionados, sendo eles referenciados, respectivamente como variantes DAl-RBC-TA, DAl-RBC-NA e DAl-RBC-SAR.

Como pode ser visto na Tabela 3.12, os melhores resultados foram obtidos com a DAl-RBC-TA, que melhorou ou manteve a precisão em 10 das 11 bases de dados, e a DAl-RBC-SAR, que melhorou ou manteve a precisão em 6 das 11 bases de dados; já o DAl-RBC-NA melhorou ou manteve a precisão em 4 das 11 bases de dados.

Tabela 3.12. Desempenho do RBC com o DistAl.

Database	Original	DAl-RBC-NA	DAl-RBC-TA	DAl-RBC-SAR
Iris	92,86 ± 9,58	**95,71** ± 6,55	**95,71** ± 6,55	**95,71** ± 6,55
Breast	96,47 ± 2,56	96,24 ± 2,44	**97,18** ± 2,53	**97,03** ± 2,55
Glass	74,00 ± 10,20	**93,80** ± 5,60	**81,40** ± 11,05	**91,70** ± 6,98
Wine	97,50 ± 5,00	91,88 ± 7,51	**98,25** ± 4,25	93,00 ± 7,48
Vehicle	54,76 ± 8,32	23,62 ± 16,49	**59,52** ± 6,88	**59,05** ± 6,95
Segment	88,93 ± 3,06	**92,43** ± 1,96	**92,69** ± 2,05	**93,31** ± 1,96
Waveform	85,13 ± 1,73	71,52 ± 3,34	**85,63** ± 0,98	83,14 ± 1,47
WDBC	96,07 ± 2,50	95,61 ± 3,10	**96,11** ± 3,37	95,64 ± 3,22
WPBC	77,78 ± 14,91	77,33 ± 13,33	75,44 ± 13,95	76,56 ± 13,45
Ionosphere	88,82 ± 6,14	**89,71** ± 6,51	**89,94** ± 6,70	**90,76** ± 6,05
Sonar	70,00 ± 12,65	**74,70** ± 11,14	**72,60** ± 11,30	**75,00** ± 9,17
Média	**83,85**	**82,05**	**85,86**	**86,45**
(4-15)	90,21	94,41	93,14	94,36
(16-34)	81,92	75,03	83,22	83,08
(60)	70,00	74,70	72,60	75,00

Entre as três combinações existentes, a DAl-RBC-SAR obteve os melhores resultados, superando por uma pequena margem a DAl-RBC-TA, e por uma margem maior a DAl-RBC-NA que, por sua vez, não superou a média de precisão quando utilizado o conjunto original de atributos.

Tanto a DAl-RBC-SAR quanto a DAl-RBC-TA superaram em média a precisão obtida com o conjunto original de atributos, sendo, portanto, boas soluções para serem empregadas na busca por subconjuntos de atributos. Não houve degradação significativa da precisão nos resultados dos experimentos usando estas duas combinações. O mesmo não ocorreu com a DAl-RBC-NA , que em diversas bases de dados apresentou o mesmo problema que a DAl-Hill-Forward, selecionando poucos atributos que não são capazes de realizar uma boa classificação do conjunto de dados de validação.

Nas bases de dados Waveform e Vehicle, a DAl-RBC-NA degradou a precisão por selecionar poucos atributos. Na Waveform esta degradação não ocorreu em todos os experimentos (são 10 repetições), mas em alguns deles foi selecionado apenas um atributo que não foi capaz de garantir uma boa precisão no conjunto de dados de

validação. Devido a este comportamento indesejado, a DAl-RBC-NA não se mostrou uma solução interessante para a busca de subconjuntos de atributos.

Considerando o número de atributos selecionados, a DAl-RBC-SAR obteve subconjuntos menores em relação à DAl-RBC-TA na maioria dos casos por uma diferença significativa, como pode ser notado na Tabela 3.13. Já em relação à quantidade de estados avaliados quase não houve diferenças entre as três combinações.

Tabela 3.13. Atributos selecionados e número de avaliações no RBC com o DistAl.

Database	**Original**	**DAl-RBC-NA**		**DAl-RBC-TA**		**DAl-RBC-SAR**	
	Atribs	**Atribs**	Avals	**Atribs**	Avals	**Atribs**	Avals
Iris	4	*1*	8	*1*	8	1,8	7,6
Breast	10	5,4	25	6	31	*5,1*	33
Glass	10	2,2	28	5,6	22	2,5	27
Wine	13	*4,7*	32,5	9,7	40,3	6,9	31,2
Vehicle	18	*2,9*	41,4	12,2	46,8	8,8	48,6
Segment	19	7,4	57	10,2	62,7	*7,1*	72,2
Waveform	21	*11,8*	77	15,5	52,5	12,5	52,5
WDBC	30	*7,9*	75	23,1	102	13,9	81
WPBC	33	*3,7*	66	27	99	14,7	92,4
Ionosphere	34	*6,9*	102	27,1	102	15,6	88,4
Sonar	60	*6,9*	156	53,3	144	28,8	150
Média	**22,9**	**5,5**	**60,7**	**17,3**	**64,6**	**10,7**	**62,2**
(4-15)	9,3	3,3	23,4	5,6	25,3	4,1	24,7
(16-34)	27,4	7,5	75,4	20,6	83,6	12,8	77,3
(60)	60,0	6,9	156,0	53,3	144,0	28,8	150,0

Assim como foi observado no *wrapper* que utiliza o NN como método de aprendizado, o RBC mostrou-se mais interessante que o *Hill-Climbing* uma vez que, além de obter uma média de precisão melhor, obteve os resultados com menos atributos.

A combinação DAl-RBC-SAR mostrou-se a solução mais eficaz para a seleção de atributos, dentre aquelas que utilizam o RBC, sendo inclusive interessante frente ao *Beam Search* ou aos Algoritmos Genéticos, por avaliar poucos subconjuntos no processo de busca.

3.4.3.3 Beam Search

Os resultados obtidos com o método de busca *Beam Search* utilizando o algoritmo de aprendizado DistAl em conjunto com as três estratégias propostas (*Forward*, *Backward* e *Random*) estão nas Tabelas 3.14 e 3.15. Essas variantes são referenciados como DAl-

Beam-Forward, DAl-Beam-Backward e DAl-Beam-Random. Assim como nos experimentos realizados com o NN, o tamanho do beam utilizado foi 25 e o número de expansões sem melhoria (K) foi 50.

As três estratégias de busca utilizadas apresentaram um desempenho equilibrado com o método *Beam Search* quando considerada a média de precisão final e superaram a média de precisão do conjunto original de atributos. Assim como no *wrapper* que utiliza o NN como algoritmo de aprendizado, o *Beam Search* forneceu bons resultados e, da mesma forma, exigiu grande esforço computacional. O tempo de processamento do DistAl tornou a busca *Beam Search* bastante demorada em algumas bases de dados, já que esta avalia muitos subconjuntos e o DistAl é mais custoso computacionalmente em relação ao NN (levando até quatro horas no caso da base de dados Waveform, onde 1000 instâncias foram utilizadas para o processo de seleção).

Tabela 3.14. Desempenho do *Beam Search* com o DistAl.

Database	Original	DAl-Beam-Forward	DAl-Beam-Backward	DAl-Beam-Random
Iris	92,86 ± 9,58	**95,71** ± 6,55	**95,71** ± 6,55	**95,71** ± 6,55
Breast	96,47 ± 2,56	**96,76** ± 2,06	**96,76** ± 2,44	**96,76** ± 2,44
Glass	74,00 ± 10,20	**95,00** ± 5,00	**95,00** ± 5,00	**95,00** ± 5,00
Wine	97,50 ± 5,00	95,00 ± 8,29	95,00 ± 8,29	93,61 ± 9,24
Vehicle	54,76 ± 8,32	**62,38** ± 7,51	**62,38** ± 7,51	**62,38** ± 7,51
Segment	88,93 ± 3,06	**92,60** ± 1,57	**93,97** ± 1,94	**92,65** ± 1,75
Waveform	85,13 ± 1,73	84,13 ± 1,49	84,13 ± 1,49	84,13 ± 1,49
WDBC	96,07 ± 2,50	95,36 ± 4,53	95,00 ± 2,86	**96,75** ± 2,92
WPBC	77,78 ± 14,91	**85,56** ± 16,52	**82,22** ± 15,07	**79,00** ± 12,39
Ionosphere	88,82 ± 6,14	**91,18** ± 5,42	**91,76** ± 6,55	**92,06** ± 6,10
Sonar	70,00 ± 12,65	**78,00** ± 13,27	**72,00** ± 12,49	**75,70** ± 10,50
Média	**83,85**	**88,33**	**87,63**	**87,61**
(4-15)	90,21	95,62	95,62	95,27
(16-34)	81,92	85,20	84,91	84,49
(60)	70,00	78,00	72,00	75,70

A DAl-Beam-Forward apresentou a maior otimização, considerando que melhorou ou manteve a precisão em 8 das 11 bases de dados, e obteve em média o menor número de atributos selecionados, como mostra a Tabela 3.15. Assim, trata-se de uma solução bastante interessante mesmo frente ao Algoritmo Genético, o método de busca associado ao DistAl que obteve os melhores resultados em conjunto com o DistAl, considerando a precisão obtida e o número de subconjuntos avaliados (a variante DAl-AG é abordada na Subseção 3.4.3.5).

A DAl-Beam-Random também apresentou bons resultados (melhorou ou manteve a precisão em 9 das 11 bases de dados), selecionando poucos atributos e otimizando significativamente a precisão de classificação. Já a DAl-Beam-Backward selecionou um número maior de atributos e teve uma otimização na precisão similar à da DAl-Beam-Random (melhorou ou manteve a precisão em 8 das 11 bases de dados), sendo assim uma solução menos apropriada para a seleção de um subconjunto reduzido.

Tabela 3.15. Atributos selecionados e número de avaliações no *Beam Search* com o DistAl.

Database	**Original**	**DAl-Beam-Forward**		**DAl-Beam-Backward**		**DAl-Beam-Random**	
	Atribs	**Atribs**	Avals	**Atribs**	Avals	**Atribs**	Avals
Iris	4	*1*	16	*1*	16	*1,0*	16,0
Breast	10	6	791	5	541	5,0	547,6
Glass	10	*1*	380	*1*	457	*1,0*	426,7
Wine	13	*6*	686	*6*	1270	7,7	859,3
Vehicle	18	*9*	1855	*9*	1738	*9,0*	1236,8
Segment	19	*7*	2219	9	1139	7,7	1467,3
Waveform	21	*14*	2173	*14*	1946	*14,0*	2074,0
WDBC	30	*9*	1867	23	1791	14,3	2665,3
WPBC	33	*13*	4364	19	3591	16,2	2832,8
Ionosphere	34	*9*	2401	25	2848	18,0	3868,1
Sonar	60	*16*	5713	44	9645	28,1	6608,5
Média	**22,9**	**8,3**	**2042,3**	**14,2**	**2271,1**	**11,1**	**2054,8**
(4-15)	9,3	3,5	468,3	3,3	571	3,7	462,4
(16-34)	27,4	10,4	2604,8	18,0	2263,0	14,0	2581,5
(60)	60,0	16,0	5713,0	44,0	9645,0	28,1	6608,5

3.4.3.4 Las Vegas

O Las Vegas *Wrapper* integrado ao DistAL (DAl-LasVegas) apresentou bons resultados, que foram similares aos obtidos pela busca *Beam Search*, como pode ser observado na Tabela 3.16. Apenas a base de dados Wine não foi otimizada dentre aquelas utilizadas nos experimentos e descritas na Subseção 3.4.1.1; a variante DAl-LasVegas melhorou ou manteve a precisão em 10 das 11 bases de dados. Assim como nos experimentos realizados com o NN, o número máximo de subconjuntos gerados sem haver melhoria foi definido como $60 \times$ atributos.

A qualidade de encontrar subconjuntos com poucos atributos observada na integração com o NN foi verificada também com o DistAl. Uma característica indesejável que pode ser notada é que o LVW foi o método de busca que em média avaliou mais subconjuntos, em decorrência da sua busca não direcionada.

Se comparado com o método DAl-Beam-Forward, o DAl-LasVegas não apresenta quaisquer vantagens e por isso não se mostrou uma opção viável (considerando as bases de dados utilizadas), apesar da sua simplicidade de implementação.

Tabela 3.16. Desempenho do Las Vegas com o DistAl.

Database	Original		DAl-LasVegas		
	Atribs	Precisão	Atribs	Precisão	Avals
Íris	4	92,86 ± 9,58	1,0	**95,71** ± 6,55	250,2
Breast	10	96,47 ± 2,56	5,7	**96,62** ± 2,63	998,2
Glass	10	74,00 ± 10,20	1,0	**95,00** ± 5,00	672,8
Wine	13	97,50 ± 5,00	6,9	94,38 ± 7,94	1322,5
Vehicle	18	54,76 ± 8,32	7,7	**57,43** ± 9,46	1559,1
Segment	19	88,93 ± 3,06	6,2	**93,21** ± 1,63	2474,7
Waveform	21	85,13 ± 1,73	17,0	**85,35** ± 1,34	2206,0
WDBC	30	96,07 ± 2,50	12,6	**96,07** ± 3,67	3523,0
WPBC	33	77,78 ± 14,91	6,7	**81,00** ± 12,06	3202,1
Ionosphere	34	88,82 ± 6,14	18,8	**92,59** ± 6,15	3755,2
Sonar	60	70,00 ± 12,65	22,8	**75,60** ± 10,28	8188,3
Média	**22,9**	**83,85**	**9,7**	**87,54**	**2559,3**
(4-15)	9,3	90,21	3,7	95,43	810,9
(16-34)	27,4	87,35	12,3	89,64	3032,2
(60)	60	70,00	22,8	75,60	8188,3

3.4.3.5 Algoritmo Genético

A variante DAl-AG apresentou ótimos resultados, similares em precisão ao Beam-Forward, com o número de atributos reduzido à metade, mesmo com o número de avaliações de subconjuntos fixa em 1000 (20 gerações com 50 indivíduos cada), conforme os resultados apresentados na Tabela 3.17.

A única base de dados com degradação na precisão de classificação foi a Wine, que apresentou comportamento similar em outros métodos de busca. A DAl-AG melhorou ou manteve a precisão em 10 das 11 bases de dados.

Com estas características, o AG (integrado ao DistAl) pode ser considerado um método de busca bastante eficiente, pois produziu ótimos resultados sem necessitar avaliar um grande número de subconjuntos.

Tabela 3.17. Desempenho do Algoritmo Genético com o DistAl.

Database	Original		DAI-AG	
	Atribs	Precisão	Atribs	Precisão
Íris	4	92,86 ± 9,58	1,0	**95,71** ± 6,55
Breast	10	96,47 ± 2,56	5,0	**96,85** ± 2,46
Glass	10	74,00 ± 10,20	1,0	**95,00** ± 5,00
Wine	13	97,50 ± 5,00	6,8	93,38 ± 9,19
Vehicle	18	54,76 ± 8,32	9,5	**63,89** ± 6,70
Segment	19	88,93 ± 3,06	8,0	**93,57** ± 1,92
Waveform	21	85,13 ± 1,73	15,0	**85,33** ± 1,18
WDBC	30	96,07 ± 2,50	15,3	**96,54** ± 3,05
WPBC	33	77,78 ± 14,91	10,2	**80,78** ± 12,72
Ionosphere	34	88,82 ± 6,14	15,9	**93,29** ± 5,59
Sonar	60	70,00 ± 12,65	31,4	**76,00** ± 11,22
Média	**22,9**	**83,85**	**10,8**	**88,21**
(4-15)	9,3	90,21	3,5	95,24
(16-34)	27,4	87,35	12,9	89,90
(60)	60	70,00	31,4	76,00

3.4.3.6 Considerações sobre os *Wrappers* DistAl

Uma otimização similar à obtida com o algoritmo aprendizado NN também foi observada nos experimentos de seleção de atributos do tipo *wrapper* realizados com o algoritmo de aprendizado DistAl. No entanto, aconteceram problemas mais evidentes, principalmente quando poucos atributos foram selecionados por métodos que utilizaram a estratégia de seleção *Forward*.

Em algumas bases de dados (Waveform, Ionosphere, WPBC e Vehicle) a degradação na precisão de classificação foi significativa quando utilizados alguns dos métodos de busca avaliados neste trabalho, embora outros métodos de busca mais robustos tenham encontrado subconjuntos de atributos satisfatórios.

A seleção de atributos utilizando o DistAl no modelo *wrapper*, apesar de exigir maior poder computacional que aquele verificado no NN, monstrou-se viável nos experimentos realizados. O caso onde o processo de busca pelo melhor subconjunto necessitou maior tempo de execução foi com a base de dados Waveform em conjunto com o método de busca *Beam Search*, levando aproximadamente 4 horas.

3.5 Considerações sobre os *Wrappers*

Os métodos de seleção de atributos do tipo *wrapper* mostraram-se bastante adequados para a redução do número de atributos. Em diversos casos com o seu uso pôde ser observado um aumento na precisão de classificação, o que aumenta ainda mais a relevância dos métodos de seleção de atributos do tipo *wrapper*, cujo enfoque principal é voltado para a precisão de classificação.

Dentre os cinco métodos de busca avaliados, destacam-se o RBC, o *Beam Search* e o AG, que possuem características distintas, se adequando a diferentes situações. O RBC teve o melhor desempenho com relação ao número de estados avaliados para chegar ao subconjunto de atributos final, mantendo ainda a precisão de classificação (muitas vezes melhorando a precisão). Este método é, pois, adequado quando o custo computacional do método de seleção de atributos deve ser minimizado.

O *Beam Search* tem como principal característica a otimização máxima da precisão de classificação, avaliando um grande número de estados para chegar ao melhor subconjunto de atributos final. Assim, sua utilização é adequada em situações onde a melhoria da precisão de classificação é fundamental e não há restrições quanto ao custo computacional necessário para a execução do método de seleção de atributos.

O AG utilizado nos experimentos, por sua vez, mostrou-se uma opção intermediária entre os dois métodos anteriores, uma vez que não avalia tantos estados quanto o *Beam Search* mas, ainda assim, encontra subconjuntos que garantem aumento na precisão de classificação em diversas situações. Com a adequação de parâmetros tais como tamanho da população e número de gerações (específicos a cada domínio) é possível otimizar ainda mais os resultados obtidos.

Em relação às estratégias de direcionamento de busca empregadas nos métodos *Hill-Climbing* e *Beam Search*, conclui-se que a *Forward* é mais eficiente em encontrar subconjuntos de atributos com menor dimensionalidade mas, em geral, não é capaz de identificar subconjuntos que tenham a mesma precisão de classificação que a conseguida utilizando a estratégia *Backward*. Considerando as variantes avaliadas, as NN-Beam-Forward e DAl-Beam-Forward são aquelas com as melhores características, uma vez que selecionam subconjuntos com poucos atributos e, ainda assim, mantém uma boa precisão de classificação.

A estratégia *Random* não apresentou uma tendência que permita caracterizá-la

como uma estratégia favorável em relação à *Forward* ou à *Backward*. Em algumas situações houve comportamento similar à *Backward*, com identificação de subconjuntos capazes de otimizar boa classificação, em outras similar à *Forward*, com a avaliação de poucos subconjuntos.

Analisando a diferença de desempenho entre as famílias *wrapper* NN e DistAl, pode se dizer que tanto a redução da dimensionalidade quanto a otimização na precisão de classificação são possíveis utilizando ambos os algoritmos de aprendizado. No entanto, há uma grande diferença quanto ao custo computacional no uso dos dois algoritmos. A Tabela 3.18 mostra o tempo de processamento dos dois algoritmos de aprendizado articulados a cada um dos métodos de busca (*Beam Search* e AG). Para o RBC e o *Beam Search* o resultado mostrado corresponde à média entre os valores obtidos usando as suas três variantes.

Tabela 3.18. Tempo de processamento na seleção de atributos com o NN e DistAl (em segundos).

Database	Instâncias	NN			DistAl		
		RBC	Beam	AG	RBC	Beam	AG
Iris	75	0	0	1	0	0	9
Breast	341	1	7	18	19	124	204
Glass	107	0	2	4	4	12	44
Wine	89	0	3	4	2	20	21
Vehicle	423	4	67	63	194	2442	1538
Segment	1000	3	70	60	724	7755	4117
Waveform	1000	2	32	19	2546	13607	6090
WDBC	284	9	121	44	67	537	268
WPBC	97	3	28	9	13	177	48
Ionosphere	175	4	74	21	35	367	124
Sonar	104	10	142	21	38	634	94

Os dados obtidos com o AG permitem avaliar puramente o desempenho dos algoritmos de aprendizado, já que o número de subconjuntos avaliados é fixo em 1000. Enquanto o NN tem baixo custo computacional, podendo ser empregado mesmo em grandes bases de dados, o DistAl pode ser inadequado em algumas situações.

O NN implementado apresentou sensível aumento no custo computacional tanto com um maior número de atributos quanto de instâncias, mas em nenhum caso chegou a levar mais do que alguns minutos. Já o DistAl mostrou-se sensível principalmente ao número de instâncias de treinamento, tendo custo computacional bastante alto quando muitas instâncias foram utilizadas no processo de seleção de atributos.

4 CAPÍTULO

O MODELO FILTRO DE SELEÇÃO DE ATRIBUTOS

Este capítulo investiga três famílias de métodos de seleção de atributos caracterizados como filtros. São investigados os métodos Relief, Focus e LVF, juntamente com suas diversas variantes.

4.1 A Família Relief

Nesta seção são apresentadas e discutidas as principais características do algoritmo original Relief e de seis de suas variantes, a saber: Relief-A, Relief-B, Relief-C, Relief-D, Relief-E e Relief-F. Após a apresentação dos algoritmos, são apresentados os resultados obtidos com a seleção de atributos realizada pela Relief-F, a variante considerada como a mais eficiente desta família.

4.1.1 O Algoritmo Relief

O Relief foi proposto em [Kira & Rendell 1992], como um algoritmo apropriado para estimar a relevância de atributos discretos e contínuos em dados que caracterizam apenas duas classes.

Medidas tais como ganho de informação [Quinlan 1993], índice gini [Breiman et al. 1984] e medida_j [Smyth & Goodman 1992] assumem que os atributos são independentes e, portanto, não são convenientes para serem usadas em domínios onde existem dependências entre atributos. O Relief foi proposto com o objetivo de avaliar a qualidade dos atributos, analisando a capacidade deles distinguirem as instâncias entre suas vizinhas mais próximas.

Para avaliar a capacidade de discriminação dos atributos, os autores propõem que

seja usado um subconjunto do conjunto de instâncias de treinamento, a fim de reduzir a carga computacional, muito embora possa ser utilizado todo o conjunto de treinamento.

Usando distância euclidiana, para cada instância desse subconjunto, o Relief busca seus dois vizinhos mais próximos, com as seguintes características:

- O mais próximo com a mesma classe da instância – chamado de acerto mais próximo (*NearHit*).
- O mais próximo com uma classe diferente da instância – chamado de erro mais próximo (*NearMiss*).

O algoritmo utiliza a técnica de ponderação e atribui pesos proporcionais às relevâncias que os atributos têm, em discriminar as instâncias. Obviamente a capacidade de discriminação do atributo está relacionada à sua contribuição em discriminar instâncias que pertencem a classes diferentes e em não discriminar instâncias que pertencem à mesma classe. O Relief estima que o peso de um atributo A, notado por W[A], é uma aproximação da diferença de probabilidades dada pela Equação 4.1.

$$W[A] = p(\text{valor diferente de A} \mid \text{instância mais próxima de classe diferente}) - p(\text{valor diferente de A} \mid \text{instância mais próxima de mesma classe}) \quad \textbf{[4.1]}$$

A motivação para a definição de peso de atributo como na Equação 4.1 se deve à intuição que um ‘bom atributo’ deve diferenciar instâncias de classes diferentes e deve ter o mesmo valor para instâncias de mesma classe. O Relief inicializa os pesos associados a cada atributo com zero e os altera de acordo com as seguintes regras:

1. Quanto maior a contribuição de um atributo para discriminar uma instância de seu *NearMiss*, maior deve ser sua relevância (ou seja, esse atributo deve ser ‘promovido’) e, conseqüentemente, maior deve ser seu peso.
2. Quanto maior a contribuição de um atributo para discriminar uma instância de seu *NearHit*, menor deve ser sua relevância (ou seja, esse atributo deve ser ‘penalizado’) e, conseqüentemente, menor deve ser seu peso.

O aumento e a redução do peso de um atributo, portanto, é calculado por meio da diferença dos valores encontrados entre uma determinada instância e seu *NearHit* e

NearMiss, respectivamente. O peso de um atributo A para cada instância selecionada é dado pela Equação 4.2 que é uma reescrita da Equação 4.1.

$$W[A] = \text{diferença}(A, \text{Instância}, \text{NearMiss}) - \text{diferença}(A, \text{Instância}, \text{NearHit}) \quad \textbf{[4.2]}$$

sendo que a função 'diferença(A,i1,i2)' retorna a diferença entre as instâncias i1 e i2 relativa ao atributo A. Dependendo do tipo do atributo A, a função diferença é calculada como:

A é discreto

$$\text{diferença}(A,i_1,i_2) = \begin{cases} 0 \text{ se } A(i_1) = A(i_2) \\ 1 \text{ se } A(i_1) \neq A(i_2) \end{cases}$$

onde A(i) retorna o valor do atributo A na instância i.

A é contínuo

$$\text{diferença}(A,i_1,i_2) = \text{normalizado}(| i_1 - i_2 |_A)$$

onde "normalizado( )"indica que os valores dos atributos são previamente nomalizados, de forma a manter o valor da função diferença(A,i_1,i_2) dentro do intervalo [0,1].

O peso final de cada atributo é calculado por meio do somatório dos pesos W[A] para cada instância selecionada. Assim, recomenda-se uma nova normalização (já que serão somados diversos valores no intervalo [0,1]), de forma a manter o peso final dos atributos no intervalo [-1,1]. O algoritmo proposto em [Kira & Rendell 1992] reescrito em pseudocódigo é mostrado na Figura 4.1, onde o peso final é também normalizado.

A mesma função diferença(A,i_1,i_2) é utilizada para o cálculo do *NearHit* e *NearMiss*. A distância entre duas instâncias é obtida como o somatório das distâncias entre todos os atributos presentes (distância euclidiana).

Após a execução do algoritmo, com o vetor de pesos obtido, são selecionados os atributos que superam um determinado *threshold.* O valor mais conservador que pode ser estabelecido é zero, onde se garante que todos os atributos minimamente relevante são selecionados. No entanto, outros valores de *threshold* podem levar a melhores resultados. Na Subseção 4.1.5 são avaliados seis diferentes valores de *threshold*, o que permite avaliar o impacto deste parâmetro na seleção de atributos realizada pelo Relief.

```
relief(I, Q, W)

{entrada: I = {i1,i2,...,iM} , | I | = M onde cada instância i ∈ I é descrita por N atributos e uma
          classe associada;
          Q: quantidade de instâncias de I a serem usadas pelo Relief (Q ≤ M).
 saída:  vetor de pesos W, de dimensão N, onde cada posição 1 ≤ i ≤ N  corresponde
          ao peso do atributo correspondente.}

   Para j = 1 até N faça          /* define os pesos iniciais dos atributos como 0 */
      W[j] = 0;
   Fim Para;

   Para j = 1 até Q faça
        R = EscolheInstância(I);  /* seleção randômica de uma instância R em I, sem
                                     reposição */
        Hi = NearHit(R,I);        /* encontra o NearHit de R em I */
        Mi = NearMiss(R,I);       /* encontra o NearMiss de R em I */
        Para A = 1 até N faça     /* atualiza os pesos de cada atributo - a função
                                     diferença(A,i1,i2)retorna diferença entre as
                                     instâncias i1 e i2 para o atributo A */
          W[A] = W[A] – diferença(A,R,Hi)/Q + diferença(A,R,Mi)/Q;
        Fim Para;
   Fim Para;
Retorna (W);
```

Figura 4.1. Pseudocódigo do algoritmo Relief.

4.1.2 Um Exemplo de Uso do Relief

Esta subseção descreve em detalhes um exemplo do uso do Relief para a determinação do vetor de pesos associado aos atributos que caracterizam um conjunto de dados artificial.

Considere um conjunto de dados com seis instâncias, cada uma delas descritas por quatro atributos contínuos a_1, a_2, a_3 e a_4 e uma classe associada, como mostra a Tabela 4.1.

Tabela 4.1. Base de dados artificial.

	a_1	a_2	a_3	a_4	Classe
i1	0,1	0,6	0,8	0,1	A
i2	0,2	0,9	0,9	0,2	A
i3	0,3	0,2	0,8	0,2	A
i4	0,5	0,6	0,7	0,8	B
i5	0,5	0,4	0,7	0,9	B
i6	0,3	0,5	0,6	0,9	B

O conjunto de dados foi construído da seguinte forma: o atributo a_4 é o atributo que melhor discrimina entre as classes, os atributos a_1 e a_3 possuem menor poder de

discriminação entre as classes e o atributo a_2 apresenta valores aleatórios, que não ajudam na discriminação entre as classes.

O Relief foi executado selecionando-se três instâncias (Q=3). Como mostra a Figura 4.2, as instâncias selecionadas foram i1, i2 e i4. São apresentados os *NearHit* e *NearMiss* para estas instâncias (os cálculos referentes às distâncias entre as instâncias não são mostrados), bem como as atualizações do vetor de pesos.

O vetor W final (não normalizado) evidencia o atributo a_4 como o de maior relevância, os atributos a_1 e a_3 com uma relevância moderada e uma relevância negativa para o atributo a_2. A seleção do subconjunto de atributos final depende do *threshold* que determina a partir de qual valor os atributos passam a ser relevantes. Considerando o valor de *threshold* como 0, são selecionados os atributos a_1, a_3 e a_4, que apresentam valores positivos. Neste exemplo os valores não foram normalizados de forma a manter os pesos finais entre [-1, 1].

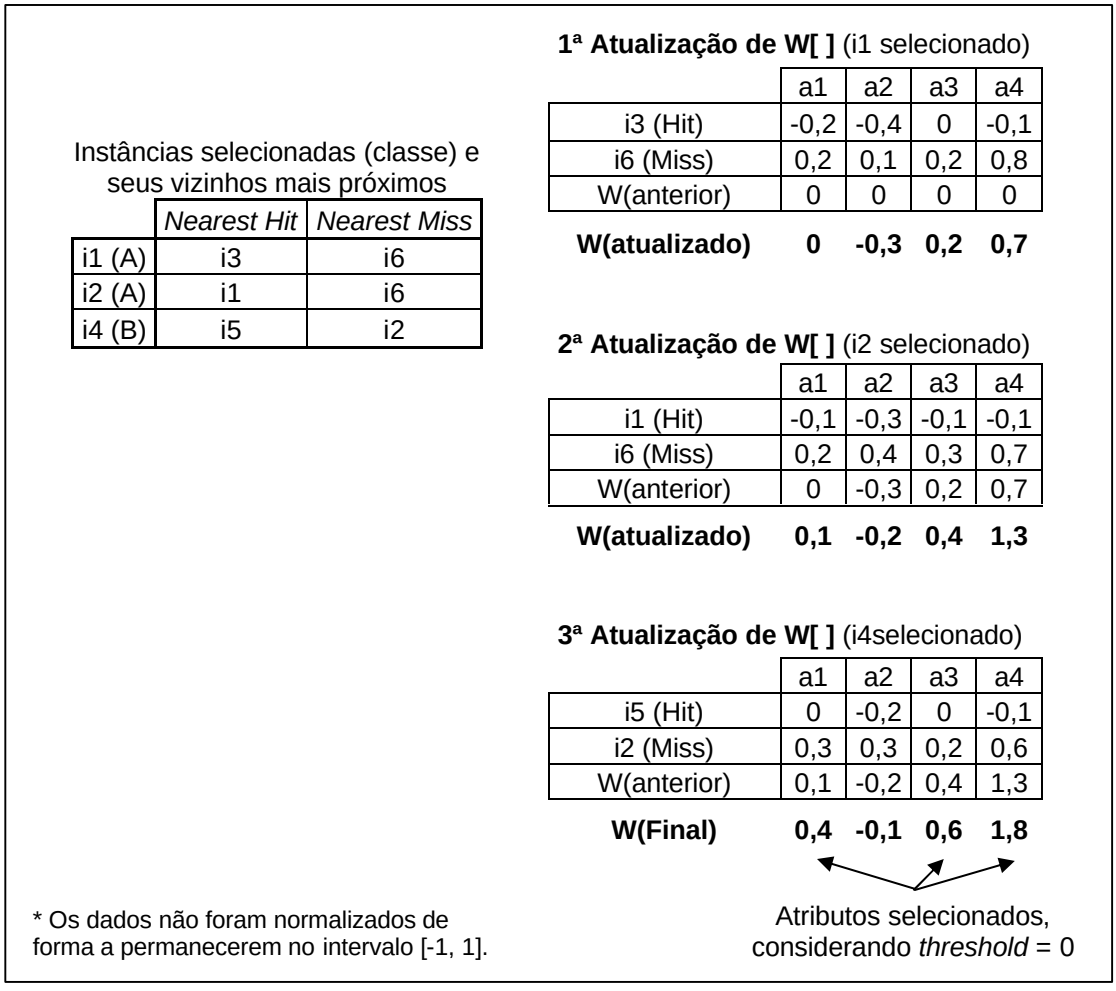


Figura 4.2. Execução do Relief usando um conjunto de dados artificial.

O Relief é bastante eficiente para a identificação de atributos irrelevantes, embora não seja adequado para eliminar atributos redundantes, já que no vetor de pesos construído dois ou mais atributos redundantes provavelmente apresentam graus de relevância semelhantes.

Na sua forma original o Relief apresenta algumas restrições. Ele não é adequado para o tratamento de dados com muito ruído e também é incapaz de manipular bases de dados incompletas. O tratamento de domínios com mais de duas classes também não pode ser feito diretamente pelo Relief original, como proposto em [Kira & Rendell 1992]. Estas deficiências foram contornadas nas propostas descritas em [Kononenko 1994], que contém uma série de extensões ao algoritmo básico do Relief, de forma a torná-lo mais robusto e eficiente, e que são referenciadas neste trabalho como a família Relief de algoritmos.

4.1.3 As Variantes Relief-A, Relief-B, Relief-C, Relief-D, Relief-E e Relief-F

As seis variantes descritas a seguir foram todas propostas em uma única referência [Kononenko 1994] e lidam especificamente com um determinado aspecto do Relief original, solucionando algumas de suas deficiências apontadas anteriormente.

(1) Relief-A

É a variante mais simples do Relief. Apesar de ainda lidar apenas com problemas com duas classes, considera não apenas uma instância próxima da mesma/outra classe, mas k instâncias mais próximas, de forma a tornar o algoritmo menos sensível a conjuntos de treinamento com muito ruído.

Em experimentos descritos em [Kononenko 1994], foi mostrado que a utilização de uma maior quantidade de vizinhos resulta em uma maior eficiência no reconhecimento dos atributos relevantes mesmo em condições de aprendizado livres de ruído (os valores de k utilizados foram 1, 3, 5, 10, 20, 30 e 40, com crescente melhoria).

(2) Relief-B

As três variantes, Relief-B, Relief-C e Relief-D, foram propostas para o tratamento de uma base de dados incompleta. As alterações são referentes à função diferença, que,

na ausência do valor de um atributo em pelo menos uma instância quando no cálculo da diferença entre duas instâncias, propõe o cálculo alternativo:

$$\text{diferença}(A,i_1,i_2) = \left(1 - \frac{1}{\text{n\_val\_atr}}\right)$$

onde n_val_atr representa o número de diferentes valores que o atributo em questão assume nas instâncias.

A variante Relief-B integra também as características da Relief-A, onde o número de instâncias próximas utilizadas nos experimentos em [Kononenko 1994] foi 10 para esta variante bem como para a Relief-C e Relief-D, que também tratam bases de dados incompletas e da mesma forma integram as características da Relief-A.

(3) Relief-C

A variante Relief-C propõe uma abordagem diferente da Relief-B, já que não realiza qualquer cálculo da função diferença, ignorando tal cálculo, quando uma das instâncias possui um valor de atributo ausente. Assim, o valor de W[A] deve ser normalizado conforme a quantidade de cálculos ignorados da função diferença (caso contrário os atributos com muitos valores ausentes tendem a ter valores menores).

Segundo [Kononenko 1994] esta otimização teve resultados ligeiramente superiores aos obtidos com a Relief-B, sobretudo quando aplicado em dados com ruído.

(4) Relief-D

A variante Relief-D é a única variante que trata bases de dados incompletas a diferenciar a ausência de um ou dois atributos. O cálculo da diferença considera a probabilidade de duas instâncias possuírem valores diferentes para um determinado atributo. Nas equações, V representa um determinado valor de atributo.

- Se apenas uma instância possui o valor desconhecido (i1):

  $$\text{diferença}(a,i1,i2) = 1 - p(\text{valor}(a,i2) \mid \text{classe}(i1))$$

- Se ambas as instâncias são desconhecidas:

  $$\text{diferença}(a,i1,i2) = 1 - \sum_{V}^{\text{n\_val\_atr}} (p(V \mid \text{classe}(i1)) \times p(V \mid \text{classe}(i2)))$$

Em [Kononenko 1994] é verificado que em uma base livre de ruídos, a diferença entre as eficiências das três variantes que tratam bases de dados incompletas é mínima, havendo uma ligeira superioridade para a Relief-D. Já na presença de ruídos a Relief-D apresentou resultados mais satisfatórios por uma maior margem, sendo assim utilizado nas variantes seguintes.

(5) Relief-E

As duas últimas variantes propostas, a Relief-E e a Relief-F, tratam todos os problemas anteriores e, adicionalmente, implementam a capacidade de tratar problemas com múltiplas classes. Em [Kira & Rendell 1992] é sugerido que o tratamento de situações de aprendizado com múltiplas classes pode ser realizado por meio da divisão do problema em diversos problemas com duas classes. De fato é possível utilizar esta abordagem, apesar de não ser uma solução muito prática.

A fim de contornar esta limitação [Kononenko 1994] propôs a variante Relief-E, que realiza uma simples generalização do Relief original, considerando o *NearMiss* como a instância mais próxima pertencente a uma classe diferente. Desta forma, a Relief-E realiza seus cálculos priorizando a distinção da classe mais próxima de cada instância.

(6) Relief-F

A variante final proposta em [Kononenko 1994] é a Relief-F, que trata o problema de múltiplas classes de forma mais abrangente que a Relief-E. Nesta variante é encontrado um *NearMiss* para cada classe diferente, exigindo, portanto um maior processamento computacional. A média entre todos os *NearMiss* encontrados é utilizada na alteração dos pesos de relevância dos atributos.

A variante Relief-F trata-se da extensão final, recomendada em [Kononenko 1994] por conseguir melhores resultados que a Relief-E tanto em bases de dados com ruídos quanto sem ruídos.

4.1.4 Considerações sobre a Família Relief

É interessante notar que o Relief realiza apenas uma estimativa da relevância dos atributos (que podem ser ordenados conforme sua relevância), não realizando uma

busca que acrescenta ou remove atributos, como na maioria dos outros métodos de seleção de atributos propostos na literatura. Assim, o algoritmo não exige muito poder computacional e, normalmente, é recomendado em situações onde o volume de dados é muito grande, já que sua complexidade é linear.

A família de algoritmos Relief está representada na Figura 4.3, onde cada caixa representa uma variante do algoritmo original, que é o ponto de partida. As linhas tracejadas separam as variantes quanto às características adicionais (indicada na parte superior esquerda da caixa tracejada) que é acrescentada à variante anterior.

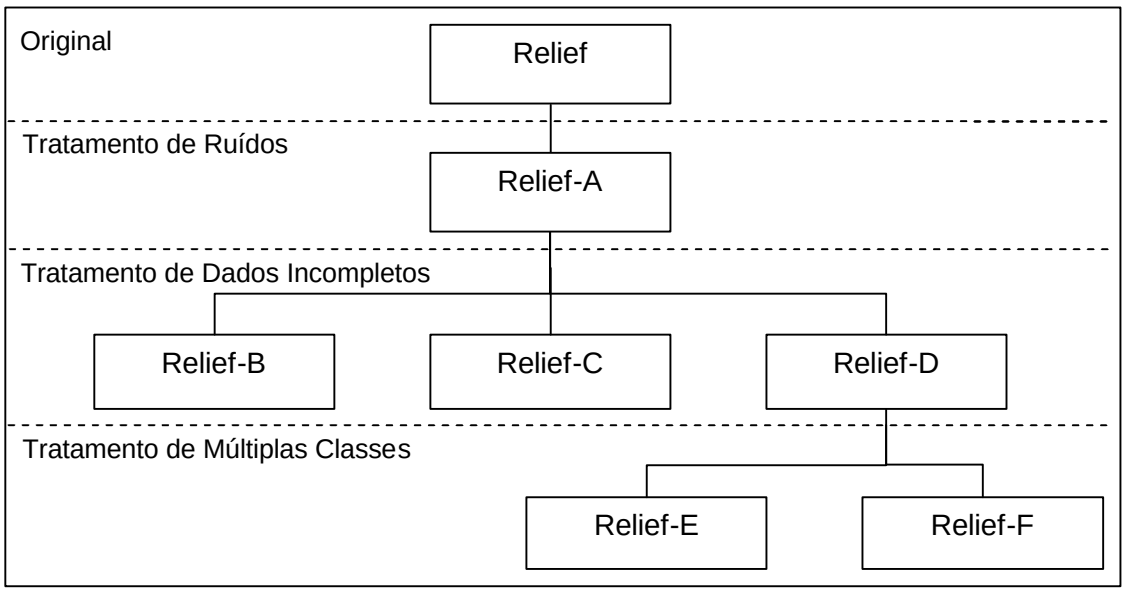


Figura 4.3. A família de algoritmos Relief e suas principais características.

4.1.5 Experimentos Realizados com o Relief

Nesta seção são apresentados os experimentos realizados com o algoritmo Relief-F, onde os principais objetivos foram (a) verificar a eficiência do algoritmo para identificar os melhores atributos e assim manter boa capacidade de classificação e (b) avaliar o impacto diferentes *thresholds* para a seleção dos atributos, após a ponderação realizada pelo algoritmo. As bases de dados utilizadas são as mesmas utilizadas nos *wrappers* e estão descritas na Subseção 3.4.1.1.

Em todos os filtros foi adotada a mesma estratégia de avaliação do desempenho do método de seleção de atributos usada na avaliação dos *wrappers.* Nesta estratégia, os dados disponíveis em cada base de dados foram separados em duas partes iguais, uma para ser utilizada pelo método de seleção de atributos e outra para a validação

(estimativa da precisão de classificação em dados não vistos, utilizando o subconjunto de atributos final). Para verificar a precisão de classificação obtida com os atributos selecionados foram utilizados dois algoritmos de AM, o NN e o DistAl, utilizando 10-validação cruzada sobre o conjunto de dados reservado para a validação.

Foram coletados dados referentes à utilização de seis *thresholds*, sendo o algoritmo Relief-F utilizado em todos os experimentos. Os parâmetros relativos ao número de instâncias utilizadas (Q) e ao número de vizinhos usados foi o máximo possível, correspondente ao número total de instâncias menos uma (em ambos os casos).

Na Tabela 4.2 é mostrado o desempenho do algoritmo NN após a seleção de atributos realizada pelo algoritmo Relief-F utilizando três diferentes *thresholds* que levaram a seleção de 75%, 50% e 25% dos atributos de cada base de dados. Nestes casos, o *threshold* não assume um valor fixo, sendo considerado o valor que permita a seleção de uma determinada porcentagem do total de atributos. Na Tabela 4.3 é mostrado o número de atributos selecionados em cada um destes casos.

Tabela 4.2. Desempenho com NN dos subconjuntos de atributos selecionados pelo Relief – 75%, 50% e 25% dos atributos.

Database	Original	75% Atributos	50% Atributos	25% Atributos
Iris	94,67 ± 6,94	**94,67** ± 6,94	**96,00** ± 6,34	93,33 ± 8,85
Breast	96,20 ± 3,68	94,44 ± 2,50	**96,20** ± 3,68	90,94 ± 3,78
Glass	91,59 ± 8,23	**93,46** ± 10,30	**93,46** ± 9,40	**96,26** ± 4,82
Wine	97,75 ± 4,99	**98,88** ± 3,96	94,38 ± 7,86	88,76 ± 11,71
Vehicle	69,74 ± 10,40	**71,63** ± 9,47	65,25 ± 8,47	60,05 ± 9,91
Segment	97,02 ± 1,46	**97,48** ± 1,25	**97,02** ± 1,37	91,15 ± 3,08
Waveform	77,15 ± 2,09	76,83 ± 2,00	**77,15** ± 1,92	72,40 ± 2,46
WDBC	95,44 ± 3,71	**96,14** ± 4,21	**97,89** ± 2,42	94,39 ± 3,38
WPBC	63,92 ± 14,72	**65,98** ± 10,40	**63,92** ± 12,35	61,86 ± 13,82
Ionosphere	88,64 ± 5,37	**91,48** ± 6,65	**89,77** ± 7,06	**89,77** ± 6,80
Sonar	80,77 ± 8,10	77,88 ± 9,18	77,88 ± 10,00	77,88 ± 14,93
Média	**86,63**	**87,17**	**86,27**	**83,34**
(4-15)	95,05	95,36	95,01	92,32
(16-34)	81,99	83,26	81,83	78,27
(60)	80,77	77,88	77,88	77,88

Na Tabela 4.4 é mostrado o desempenho do algoritmo NN após a seleção de atributos realizada pelo algoritmo Relief-F sendo utilizados outros três diferentes *thresholds*, considerados dinâmicos, pois o número de atributos selecionados não é previsível. Um dos *threshold* é definido a partir do atributo com maior relevância (referenciado como ‘maior’) um valor normalmente positivo e próximo de 1. Neste caso

o *threshold* foi definido como (maior/3), onde o número de atributos selecionados varia conforme a relevância dos atributos. Outro *threshold* usado foi em relação à media geral da relevância dos atributos, onde os atributos só foram selecionados quando sua relevância superou a média. O terceiro *threshold* dinâmico é 0, onde são selecionados todos os atributos minimamente relevantes. Na Tabela 4.5 é mostrado o número de atributos selecionados em cada um destes casos.

Tabela 4.3. Número de atributos selecionados pelo Relief – 75%, 50% e 25% dos atributos.

Database	Original	75% Atributos		50% Atributos		25% Atributos	
Iris	4	3	75,00%	2	50,00%	1	25,00%
Breast	10	7	70,00%	5	50,00%	2	20,00%
Glass	10	7	70,00%	5	50,00%	2	20,00%
Wine	13	9	69,20%	6	46,20%	3	23,10%
Vehicle	18	13	72,20%	9	50,00%	4	22,20%
Segment	19	14	73,70%	9	47,40%	4	21,10%
Waveform	21	15	71,40%	10	47,60%	5	23,80%
WDBC	30	22	73,30%	15	50,00%	7	23,30%
WPBC	33	24	72,70%	16	48,50%	8	24,20%
Ionosphere	34	25	73,50%	17	50,00%	8	23,50%
Sonar	60	45	75,00%	30	50,00%	15	25,00%
Média	**22,91**	**16,73**	**72,36%**	**11,27**	**49,06%**	**5,36**	**22,84%**
(4-15)	9,25	6,50	71,05%	4,50	49,05%	2,00	22,03%
(16-34)	25,83	18,83	72,80%	12,67	48,92%	6,00	23,02%
(60)	60,00	45,00	75,00%	30,00	50,00%	15,00	25,00%

Tabela 4.4. Desempenho com NN dos subconjuntos de atributos selecionados pelo Relief – Peso > (Maior/3), Peso > (Média), Peso > 0.

Database	Original	Peso >(maior/3)	Peso > (Média)	Peso > 0
Iris	94,67 ± 6,94	**94,67** ± 6,94	**94,67** ± 6,94	**94,67** ± 6,94
Breast	96,20 ± 3,68	94,44 ± 2,50	95,91 ± 2,43	**96,20** ± 3,68
Glass	91,59 ± 8,23	**98,13** ± 3,83	**96,26** ± 4,82	**91,59** ± 8,23
Wine	97,75 ± 4,99	95,51 ± 5,74	89,89 ± 11,07	**97,75** ± 4,99
Vehicle	69,74 ± 10,40	67,61 ± 9,11	64,30 ± 10,75	**69,74** ± 10,40
Segment	97,02 ± 1,46	96,56 ± 1,26	96,56 ± 1,26	**97,48** ± 1,14
Waveform	77,15 ± 2,09	76,93 ± 1,73	**77,25** ± 1,70	**77,45** ± 1,62
WDBC	95,44 ± 3,71	**96,84** ± 3,02	**96,84** ± 3,02	**95,44** ± 3,71
WPBC	63,92 ± 14,72	**65,98** ± 11,76	**65,98** ± 15,85	**65,98** ± 9,55
Ionosphere	88,64 ± 5,37	**90,91** ± 5,47	**92,05** ± 6,17	**88,64** ± 5,37
Sonar	80,77 ± 8,10	77,88 ± 11,22	77,88 ± 11,22	**80,77** ± 11,42
Média	**86,63**	**86,86**	**86,14**	**86,88**
(4-15)	95,05	95,69	94,18	95,05
(16-34)	81,99	82,47	82,16	82,46
(60)	80,77	77,88	77,88	80,77

Tabela 4.5. Número de atributos selecionados pelo Relief – Peso > (Maior/3), Peso > (Média), Peso > 0.

Database	Original	Peso >(maior/3)		Peso > (Média)		Peso > 0	
Iris	4	3	75,00%	3	75,00%	4	100,00%
Breast	10	1	10,00%	2	20,00%	10	100,00%
Glass	10	7	70,00%	5	50,00%	10	100,00%
Wine	13	5	38,50%	4	30,80%	13	100,00%
Vehicle	18	11	61,10%	6	33,30%	17	94,40%
Segment	19	10	52,60%	10	52,60%	15	78,90%
Waveform	21	14	66,70%	11	52,40%	20	95,20%
WDBC	30	13	43,30%	13	43,30%	30	100,00%
WPBC	33	10	30,30%	14	42,40%	25	75,80%
Ionosphere	34	27	79,40%	21	61,80%	33	97,10%
Sonar	60	26	43,30%	26	43,30%	57	95,00%
Média	**22,91**	**11,55**	**51,84%**	**10,45**	**45,90%**	**21,27**	**94,22%**
(4-15)	9,25	4,00	48,38%	3,50	43,95%	9,25	100,00%
(16-34)	25,83	14,17	55,57%	12,50	47,63%	23,33	90,23%
(60)	60,00	26,00	43,30%	26,00	43,30%	57,00	95,00%

Na Tabela 4.6 é mostrado o desempenho do algoritmo DistAl após a seleção de atributos realizada pelo Relief-F sendo utilizados três diferentes *thresholds* que levaram a seleção de 75%, 50% e 25% dos atributos de cada base de dados. Já na Tabela 4.7 é mostrado o desempenho do DistAl utilizando os três *thresholds* dinâmicos.

Tabela 4.6. Desempenho com DistAl dos subconjuntos de atributos selecionados pelo Relief – 75%, 50% e 25% dos atributos..

Database	Original	75% Atributos	50% Atributos	25% Atributos
Iris	92,86 ± 9,58	91,43 ± 9,48	**94,29** ± 7,00	**95,71** ± 6,55
Breast	96,47 ± 2,56	**97,06** ± 2,28	**96,76** ± 2,77	0,00 ± 0,00
Glass	74,00 ± 10,20	**79,00** ± 13,00	**85,00** ± 9,22	**87,00** ± 9,00
Wine	97,50 ± 5,00	**98,75** ± 3,75	93,75 ± 6,25	88,75 ± 10,38
Vehicle	54,76 ± 8,32	**54,76** ± 5,53	**54,76** ± 9,76	6,43 ± 14,13
Segment	88,93 ± 3,06	**91,15** ± 1,91	**89,31** ± 1,87	80,08 ± 2,06
Waveform	85,13 ± 1,73	**85,58** ± 1,28	83,93 ± 1,45	79,65 ± 2,11
WDBC	96,07 ± 2,50	95,71 ± 3,50	**97,14** ± 3,11	**96,07** ± 2,50
WPBC	77,78 ± 14,91	74,44 ± 14,10	73,33 ± 14,23	**80,00** ± 10,89
Ionosphere	88,82 ± 6,14	**91,76** ± 7,53	**92,94** ± 6,86	**90,59** ± 4,71
Sonar	70,00 ± 12,65	**79,00** ± 11,36	**79,00** ± 8,31	68,00 ± 8,72
Média	**83,85**	**85,33**	**85,47**	**70,21**
(4-15)	90,21	91,56	92,45	67,87
(16-34)	81,92	82,23	81,90	72,14
(60)	70,00	79,00	79,00	68,00

Tabela 4.7. Desempenho com DistAl dos subconjuntos de atributos selecionados pelo Relief – Peso > (Maior/3), Peso > (Média), Peso > 0.

Database	Original	Peso >(maior/3)	Peso > (Média)	Peso > 0
Iris	92,86 ± 9,58	91,43 ± 9,48	91,43 ± 9,48	**92,86** ± 9,58
Breast	96,47 ± 2,56	**97,06** ± 2,28	**96,76** ± 2,77	**96,47** ± 2,56
Glass	74,00 ± 10,20	**95,00** ± 5,00	**87,00** ± 9,00	**74,00** ± 10,20
Wine	97,50 ± 5,00	95,00 ± 8,29	88,75 ± 14,20	**97,50** ± 5,00
Vehicle	54,76 ± 8,32	53,81 ± 6,83	51,43 ± 13,27	**54,76** ± 8,32
Segment	88,93 ± 3,06	**89,54** ± 2,37	**89,54** ± 2,37	**91,53** ± 2,00
Waveform	85,13 ± 1,73	**86,10** ± 1,45	84,78 ± 1,49	**86,03** ± 1,25
WDBC	96,07 ± 2,50	**97,50** ± 2,79	**97,50** ± 2,79	**96,07** ± 2,50
WPBC	77,78 ± 14,91	75,56 ± 13,88	76,67 ± 16,81	74,44 ± 14,95
Ionosphere	88,82 ± 6,14	**89,41** ± 6,86	**92,35** ± 4,59	**88,82** ± 6,14
Sonar	70,00 ± 12,65	**75,00** ± 12,85	**75,00** ± 12,85	**71,00** ± 12,21
Média	**83,85**	**85,95**	**84,66**	**83,95**
(4-15)	90,21	94,62	90,99	90,21
(16-34)	81,92	81,99	82,05	81,94
(60)	70,00	75,00	75,00	71,00

Os dados obtidos permitem analisar que os diferentes *thresholds* utilizados levam a resultados diversos, em sua maioria positivos. Apenas duas das opções de *threshold* não se mostraram adequadas. Uma destas opções é quando ele é definido como 0, resultando em subconjuntos quase sempre do mesmo tamanho que o original, sendo as únicas exceções as bases de dados WPBC e Segment, que tiveram aproximadamente um quarto dos atributos removidos, sem haver muita degradação no desempenho.

Já o *threshold* que seleciona 25% dos atributos teve como ponto negativo a seleção subconjuntos quase sempre insuficientes para garantir uma boa precisão de classificação. Após a seleção de 25% dos atributos, poucas foram as situações onde os classificadores ainda foram capazes de manter uma capacidade de classificação similar àquela encontrada com o conjunto original de atributos.

Os demais *thresholds* apresentaram resultados satisfatórios, em geral sendo capazes de otimizar ou ao menos manter uma boa capacidade de classificação tanto com o NN quanto com o DistAl. Os *thresholds* 50% dos atributos, peso>média e peso>(maior/3) apresentaram resultados semelhantes tanto na redução do subconjunto original quanto na precisão de classificação final. Por isso, eles se mostraram os mais adequados para a utilização com o algoritmo Relief-F.

4.2 A Família Focus

Nesta seção são apresentadas e discutidas as principais características do algoritmo Focus e de suas variantes, a saber, Focus-2 e C-Focus. É também apresentada uma proposta desse trabalho, um método de busca alternativo para ser utilizado com o Focus, chamado de FocusD e sua variante C-FocusD. É proposto ainda um critério para a parametrização do algoritmo C-Focus. Conclui-se com a apresentação dos resultados obtidos com os experimentos realizados com o C-Focus e C-FocusD.

4.2.1 O Algoritmo Focus

O algoritmo de seleção de atributos Focus foi proposto em [Almuallim & Dietterich 1991] e é caracterizado como um método filtro, já que realiza a seleção dos atributos sem envolver um algoritmo de aprendizado para avaliar a qualidade dos subconjuntos de atributos gerados.

O processo de seleção implementado pelo Focus busca o menor subconjunto de atributos que não implica inconsistência no conjunto de treinamento. Para tanto, realiza uma busca completa no espaço definido por todos os possíveis subconjuntos de atributos (lembrando que para um conjunto de instâncias descrito por n atributos, a dimensão deste espaço de busca é 2^n). Para os autores, conjunto consistente é aquele no qual não podem ser encontradas duas instâncias com a mesma descrição (ou seja, mesmos valores associados aos atributos que as descrevem), pertencentes a classes diferentes. Essa definição de certa forma restringe o uso da proposta Focus original a domínios apenas com atributos com valores nominais ou discretos.

O processo de busca adotado pelo Focus pode ser caracterizado como uma busca em largura que avalia primeiramente todos os subconjuntos do conjunto original de atributos que tenham apenas um atributo para, a seguir, avaliar os subconjuntos com dois atributos, depois com três e assim sucessivamente. A busca termina quando um subconjunto consistente é encontrado; esse fato caracteriza o processo de busca utilizado pelo Focus como completo, mas não exaustivo (dado que o algoritmo não necessariamente 'visita' todos os possíveis estados do espaço de busca).

O *bias* implementado pelo algoritmo prioriza hipóteses com o menor número de atributos possível, desde que o conjunto descrito por eles, entretanto, não perca a consistência. Esse *bias* é chamado de Min-Features. A definição do *bias* Min-Features como proposta em [Almuallim & Dietterich 1991] pressupõe as definições apresentadas

a seguir. A Figura 4.4 mostra o pseudo-código do algoritmo Focus.

Definição 6. Para cada n = 1, seja $\{x_1, x_2, \ldots, x_n\}$ um conjunto de n atributos booleanos e seja X_n o conjunto $\{0,1\}^n$ de todas as atribuições a esses atributos – o conjunto de instâncias. Um conceito binário c é um subconjunto de X_n.

Definição 7. Uma função binária f representa o conceito c se f(x) = 1 para todo $x \in c$ e f(x) = 0 caso contrário. Como se sabe, funções binárias podem ser representadas como fórmulas booleanas. Um atributo x_i, $1 = i = n$ é dito ser relevante ao conceito c se x_i aparece em toda fórmula booleana que representa c e irrelevante caso contrário.

Definição 8. O *bias* Min-Features é definido em [Almuallim & Dietterich 1991] como: dado um conjunto de treinamento S que representa uma certa função binária f definida sobre X_n, seja V o conjunto de todas as funções binárias consistentes com S (V é algumas vezes chamado de espaço de versões [Mitchell 1982]). Seja H o subconjunto de V cujos elementos têm o menor número de atributos relevantes. O *bias* seleciona os atributos que definem H.

```
focus(I, W);

{entrada: conjunto C = {i1,i2,...,in} com M instâncias, cada uma delas descrita por N
          atributos;
 saída:  vetor de pesos W, de dimensão N, onde cada posição 1 ≤ i ≤ N tem atribuído o
         valor 1 para atributos relevantes e 0 para atributos irrelevantes}

i ← 1;                          /* i define o número de atributos selecionados nos
                                   subconjuntos a serem gerados */
executar ← true;                /* executar define se o loop continua sendo executado */
  Enquanto ( (i < N) & executar ) faça
      Para todo subconjunto de atributos A de tamanho i faça
       /* a função verInconsist(A,C) verifica se existem quaisquer duas instâncias de
       classes diferentes em C que possuem os mesmos valores de atributos,
       considerando o subconjunto de atributos A */
       Se (não(verInconsist (A,C))) então
          W = A;                    /* se subconjunto A é consistente, W recebe A */
          executar = false;         /* a execução do algoritmo é então interrompida */
       Fim Se;
    Fim Para todo;
    i = i + 1;
    Se i = N então W = A;       /* caso não seja encontrado nenhum subconjunto de atributos
                                   A consistente, retorna o conjunto com N atributos */
  Fim Enquanto;
Retorna (W);
```

Figura 4.4. Pseudo-código do algoritmo Focus.

Apesar de bastante eficiente na remoção de atributos tanto irrelevantes quanto redundantes, o Focus original possui dois grandes problemas que o impedem de ser utilizado em muitas situações, a saber, quando o número de atributos é muito grande e quando estão presentes atributos reais ou inteiros.

O primeiro problema apontado, relacionado à quantidade de atributos, não possui uma solução capaz de resolvê-lo completamente. A resolução não é simples, pois o espaço de busca cresce exponencialmente. Por exemplo, para um conjunto de dados com 32 atributos, existem mais de 4 bilhões de subconjuntos candidatos (2^{32}).

Uma opção é descrita na Subseção 4.2.2, referente ao Focus-2, um algoritmo proposto em [Almuallim & Dietterich 1992]. Outra opção é uma proposta deste trabalho, que procura explorar de forma mais inteligente o espaço de busca dos subconjuntos de atributos. Esta proposta é chamada de FocusD e é abordada na Subseção 4.2.3.

Para o tratamento de dados reais ou inteiros, uma versão adaptada do algoritmo original, chamada C-Focus, é a opção mais eficiente. Esta proposta é descrita na Subseção 4.2.5.

4.2.2 Focus-2

Uma solução para tratar o problema do grande espaço de busca foi proposta no trabalho seguinte dos autores do Focus original, em [Almuallim & Dietterich 1992]. Neste trabalho é descrito o Focus-2, que é capaz de focalizar a verificação de subconjuntos de atributos que tenham chance real de serem consistentes, o que, na prática, permite a avaliação de um número menor de subconjuntos. O processo tende a ser mais rápido, embora continue a crescer exponencialmente e tenha um processamento extra para detectar os subconjuntos promissores.

Para ignorar subconjuntos que não são capazes de apresentar uma hipótese consistente, o Focus-2 armazena os conflitos (relativos a atributos que são necessários para distinguir duas instâncias) existentes entre todas as instâncias de classes diferentes para todos os atributos. São gerados diversos vetores que indicam as posições onde os atributos são diferentes entre duas instâncias. Para uma hipótese ser consistente, ela deve necessariamente conter os atributos com valores diferentes entre todas as instâncias de classes diferentes.

Por exemplo, em um conjunto de treinamento com 6 instâncias, 6 atributos booleanos e uma classe associada (3 instâncias de classe A e 3 de classe B), são gerados os vetores de conflitos mostrados na Figura 4.5, onde os 1s representam os conflitos detectados.

Instâncias	Conflitos:	
(1) 010100, A	c_1 – 001100 (1)&(4)	c_6 – 010111 (2)&(6)
(2) 110010, A	c_2 – 111101 (1)&(5)	c_7 – 110111 (3)&(4)
(3) 101111, A	c_3 – 110001 (1)&(6)	c_8 – 000110 (3)&(5)
(4) 011000, B	c_4 – 101010 (2)&(4)	c_9 – 001010 (3)&(6)
(5) 101001, B	c_5 – 011011 (2)&(5)	
(6) 100101, B		

Figura 4.5. Exemplo de instâncias e conflitos gerados com o Focus-2.

No exemplo pode ser percebido que um dos subconjuntos possíveis é aquele que engloba o primeiro, terceiro e quarto atributos (considerando o conflito c_1 como base e observando que apenas o conflito c_3 não pode ser satisfeito pelo terceiro e quarto atributos presentes em c_1). Os conflitos auxiliam na busca da seguinte forma: para o conflito c_1, por exemplo, pode se concluir que qualquer subconjunto de atributos consistente deve possuir o terceiro ou o quarto atributo, já que existem duas instâncias de classes diferentes que só podem ser diferenciadas por um destes atributos.

Antes do processo de busca, uma série de condições como esta são construídas, de forma a evitar verificações de inconsistência desnecessárias. Apenas com a análise do conflito c_1 seriam evitadas as verificações de subconjuntos de atributos que não incluíssem o terceiro ou quarto atributos, por exemplo.

Em [Almuallim & Dietterich 1992] alguns experimentos foram conduzidos, mostrando que o Focus-2 é de fato mais rápido que o Focus, apresentando a mesma capacidade em encontrar subconjuntos inconsistentes.

4.2.3 FocusD

O espaço de busca possui 2^n possíveis subconjuntos a serem verificados em relação à sua consistência. A estratégia de busca utilizada pelo Focus inicia-se pelos subconjuntos com um atributo e avança com subconjuntos com um número crescente de atributos. Caso o subconjunto consistente possua uma grande quantidade de atributos, sobretudo

quando excede metade do total de atributos, o processo de busca irá explorar uma grande parte do espaço de busca, uma situação muitas vezes impraticável, quando o número de atributos original é muito alto.

Outra característica da busca em largura realizada pelo Focus, é que no espaço de busca explorado, os subconjuntos com um número intermediário de atributos possuem muito mais subconjuntos do que aqueles com poucos atributos ou muitos atributos. Para visualizar esta característica é interessante calcular o número de subconjuntos possíveis com um determinado número de atributos.

Isso pode ser calculado através de análise combinatória. O número de subconjuntos é obtido através de uma combinação simples, $C_{n,p,}$ onde n é a quantidade total de atributos que definem cada instância e p é o número de atributos que serão selecionados.

$$C_{n,p} = \frac{n!}{p!(n-p)!}$$

Desta forma, pode-se facilmente calcular o número de subconjuntos que podem ser construídos com um determinado número de atributos. Na Figura 4.6 este cenário é mostrado em um gráfico, quando o número de atributos é igual a 30.

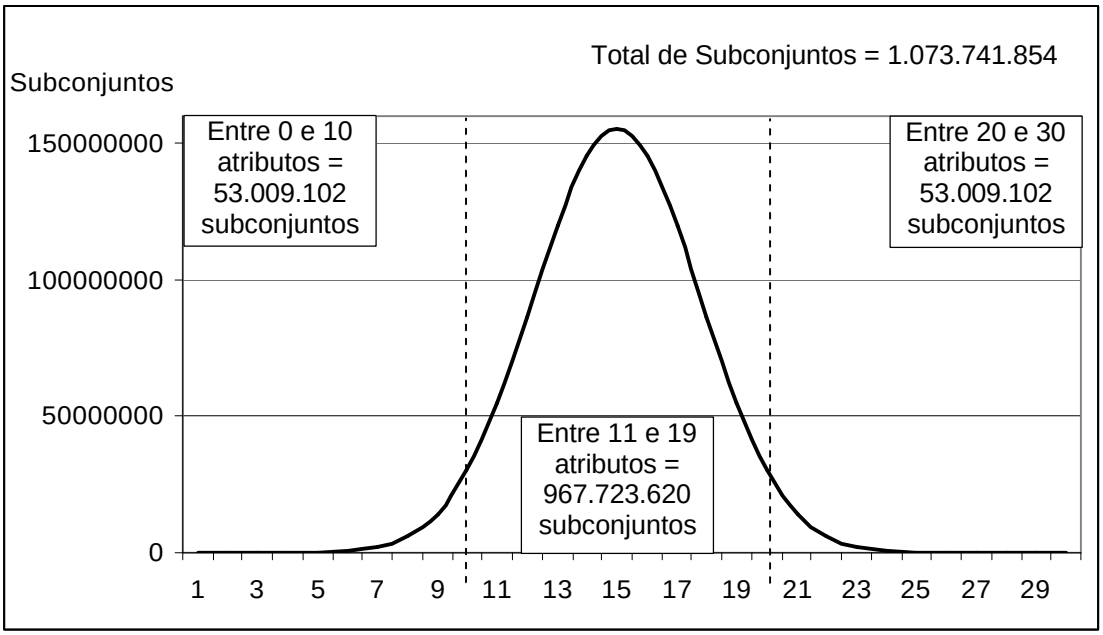


Figura 4.6. Subconjuntos de atributos em relação ao número de atributos.

Nota-se que com um número não muito alto de atributos (30) a região intermediária, aqui considerada a área entre um terço e dois terços do total, concentra

mais de 90% de todos os subconjuntos que podem ser explorados nesta situação. Quanto maior o número de atributos maior é esta concentração.

Observando esta característica, é proposta neste trabalho uma heurística otimizada para a exploração deste espaço de busca, que procura explorar mais eficientemente este espaço, evitando ao máximo, quando possível, a região intermediária. A heurística proposta é bastante eficiente quando o número de atributos necessários para garantir a consistência dos dados é superior à metade dos atributos originais.

Esta heurística utiliza dois métodos de escolha da região a ser explorada, um para a região intermediária (chamado de MRI) e outro para a região externa (chamado de MRE). O MRI sempre divide a busca em duas frentes, uma referente aos subconjuntos com um número de atributos igual a 1/3 entre o valor mínimo e o valor máximo e outra referente a 2/3 do mesmo intervalo. Caso um subconjunto consistente for encontrado com subconjuntos de 1/3, o método de busca utilizado passa a ser o MRE entre o valor mínimo e 1/3. Caso não for encontrado nenhum subconjunto com subconjuntos de 1/3 e nenhum de 2/3, o MRE é utilizado entre 2/3 e o valor máximo. Já se for encontrado um subconjunto de 2/3, o MRI é aplicado entre 1/3 e 2/3 (já que é subconjuntos menores podem existir). Desta forma, o MRI evita a maior concentração dos subconjuntos com um número de atributos próximos à metade do total.

A busca inicia-se com o MRI e procede conforme abordado. O MRE é utilizado nas situações indicadas e, após ser utilizado pela primeira vez, é o único método a ser utilizado. A diferença em relação ao MRI é que a exploração é feita diretamente nos subconjuntos com um número de atributos igual a 1/2 entre o valor mínimo e o valor máximo. Caso um subconjunto consistente for encontrado, o MRE é utilizado novamente entre o valor mínimo e 1/2, caso contrário o MRE é utilizado entre 1/2 e o valor máximo.

O processo só termina quando os valores mínimo e máximo são iguais, o que indica que não há mais possíveis subconjuntos consistentes com um número menor de atributos. Na Figura 4.7 é mostrada a dinâmica de uma busca realizada pelo Focus original e na Figura 4.8 a busca realizada pelo FocusD. Na Figura 4.9 é mostrado o pseudo-código deste método.

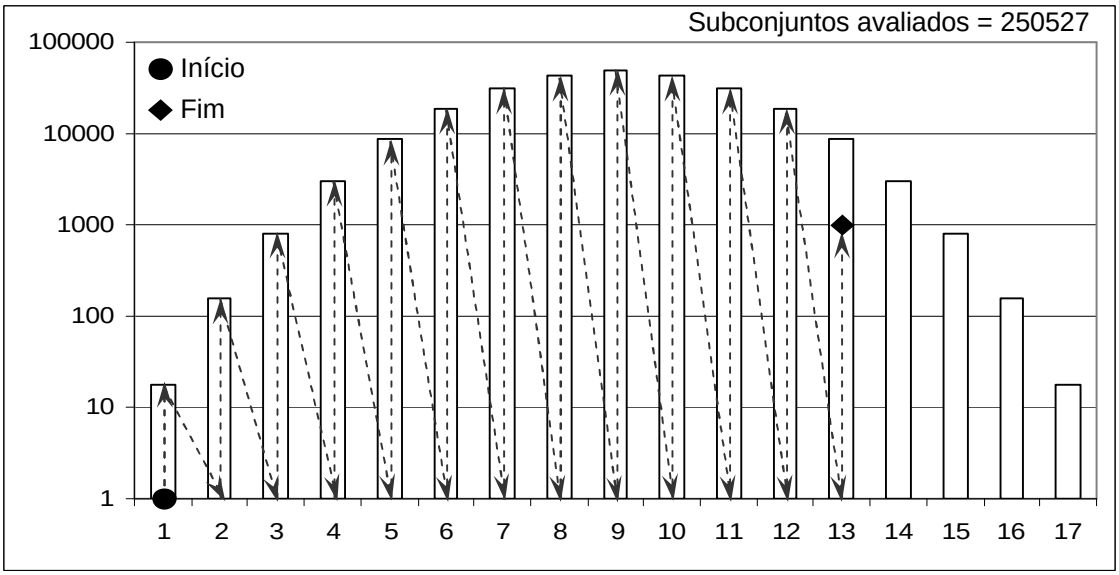


Figura 4.7. Dinâmica de uma busca realizada pelo Focus original.

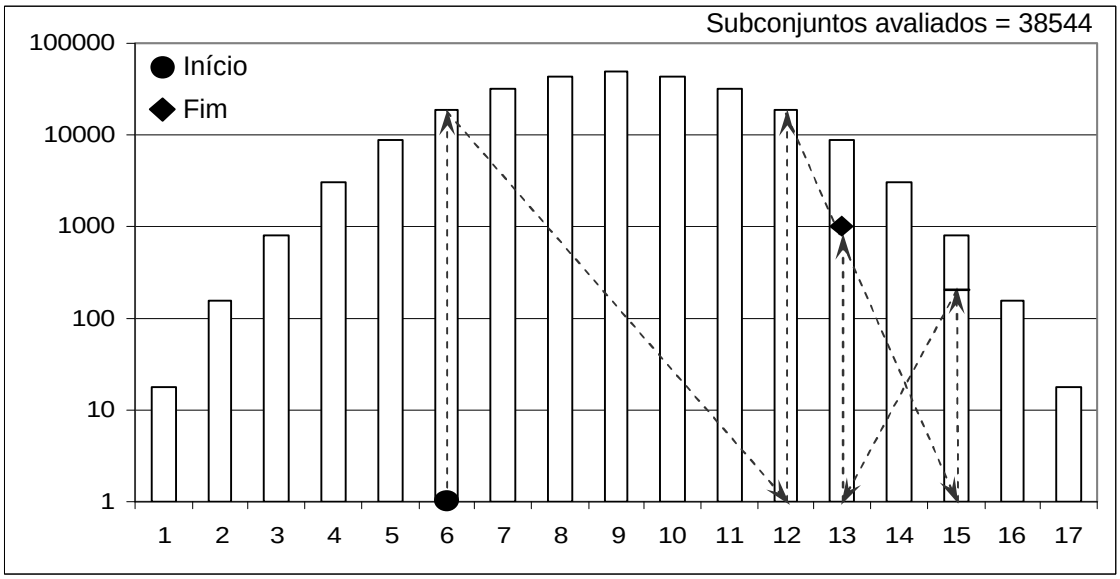


Figura 4.8. Dinâmica de uma busca realizada pelo FocusD.

```
MRI (0, num_atributos);   /* Algoritmo principal executa o MRI entre 0 e
                             num_atributos */
MRI (min, max)                              /* Método da Região Intermediária */
     explora1 = (max-min)/3 + min;          /* explora1 recebe 1/3 do intervalo */
     explora2 = (max-min)/3*2 + min;        /* explora2 recebe 2/3 do intervalo  */

     se ( checar(explora1) ) então          /* verifica se explora1 é consistente */
          se ( (explora1-min)>1) então MRE ( min, explora1-1 );
          /* se sim, e (explora1-min)>1, explorar a região até 1/3 com o MRE */
     fim se
     senão
          se ( checar(explora2) ) então     /* verifica se explora2 é consistente */
               se ( (explora2 - explora1)>1) então MRI (explora1+1, explora2-1);
               /* senão, se explora2 é consistente, e (explora2 - explora1)>1,
               explorar a região entre 1/3 e 2/3 com o MRI */
          fim se
     fim senão
     senão
          se ( (max - explora2)>1) então MRE (explora2+1, max);
           /* caso explora1 e explora2 não apresentem inconsistência, e
           (max - explora2)>1, explorar região acima de 2/3 com MRE */
     fim senão
Fim.

MRE (min, max)                              /* Método da Região Externa */
     explora1 = (max-min)/2) + min;         /* explora1 recebe 1/2 do intervalo */

     se ( checar(explora1) ) então          /* verifica se explora1 é consistente */
          se ( explora1-min>0 ) então
               MRE ( min, explora1-1 ); /* se sim, e explora1-min>0, explorar a
                                           região abaixo de 1/2 com o MRE */
          fim se
     fim se
     senão
          se ( max-explora1 >0 ) então
               MRE ( explora1+1, max );/* senão, e max-explora1 >0, explorar a
                                          região acima de 1/2 com o MRE */
          fim se
     fim senão
Fim.
```

Figura 4.9. Pseudo-código da busca realizada pelo FocusD.

4.2.4 Greedy Focus

Outra opção para a aplicação do *bias* Min-Features minimizando a carga de processamento, é a utilização de heurísticas que buscam não o subconjunto ótimo de atributos, mas um que seja bom o suficiente, em um tempo menor de busca. No mesmo trabalho [Almuallim & Dietterich 1992] foram propostos três algoritmos que implementam o *bias* Min-Features utilizando heurísticas, o MIG (*Mutual-Information-Greedy*), o SG (*Simple-Greedy*) e o WG (*Weighted-Greedy*).

Todos eles selecionam progressivamente um atributo que parece ser o mais

promissor (segundo uma determinada heurística), adicionando-o a um subconjunto parcial. A seleção dos atributos é executada em um pequeno número de iterações, número normalmente próximo ao total de atributos relevantes no conjunto de treinamento.

4.2.5 C-Focus

Como já comentado, o Focus e as demais propostas citadas do *bias* Min-Features podem tratar apenas problemas com atributos booleanos ou nominais, não sendo adequados a atributos contínuos. Esta é a grande limitação deste algoritmo

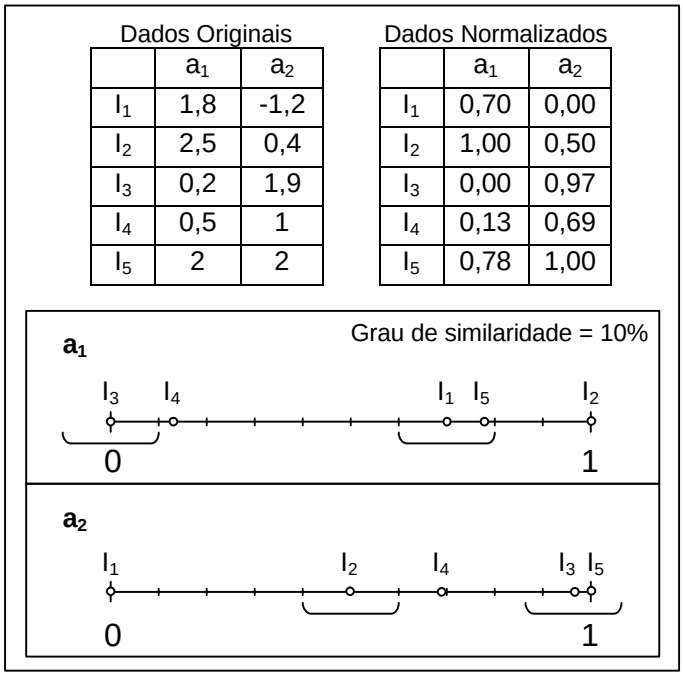
Para resolver a restrição de seleção apenas entre atributos nominais, duas diferentes abordagens podem ser adotadas. A abordagem mais simples é realizar a discretização dos atributos reais (ou inteiros) como um passo anterior ao processo de seleção de atributos. Desta forma um conjunto de treinamento intermediário é gerado, contendo atributos discretizados em intervalos.

Outra abordagem proposta em [Arauzo et al. 2003] é chamada de C-Focus. A diferença para o Focus original está na verificação de consistência, que sofre uma alteração que proporciona a verificação de dados reais e inteiros. Neste algoritmo, não apenas os valores iguais são responsáveis por evidenciar inconsistência entre duas instâncias, mas também atributos com valores próximos entre si (conceito definido como 'bastante próximo') A proximidade entre os valores de atributo que determina se estes são bastante próximos ou não é um valor percentual pré-definido pelo usuário, chamado de 'grau de similaridade'. Os dados originais são normalizados no intervalo [0,1] para a aplicação do conceito de bastante próximo em relação a um grau de similaridade.

Na Figura 4.10 está representado um conjunto de dados com 2 atributos e cinco instâncias (Dados Originais), que são primeiramente normalizados (Dados Normalizados). Foi considerado o grau de similaridade de 10% onde, no caso do atributo a_1, as instâncias I_1 e I_5 (valores de atributo normalizados 0,7 e 0,78 respectivamente) possuem valores bastante próximos, enquanto que as instâncias I_3 e I_4 não (valores de atributo normalizados 0 e 0,13 respectivamente). Para o atributo a_2, as instâncias com valores bastante próximos são a I_3 e I_5, com valores normalizados

0,97 e 1 respectivamente. As demais instâncias não possuem valores bastante próximos.

No exemplo mostrado nesta figura os dois atributos são necessários para manter a consistência dos dados (considerando 10% como grau de similaridade), já que separadamente ambos os atributos apresentam instâncias com valores similares, o que indica inconsistência. Caso as instâncias que apresentassem inconsistência fossem as mesmas para os dois atributos, este conjunto de dados seria inconsistente.



	a_1	a_2
I_1	1,8	-1,2
I_2	2,5	0,4
I_3	0,2	1,9
I_4	0,5	1
I_5	2	2

	a_1	a_2
I_1	0,70	0,00
I_2	1,00	0,50
I_3	0,00	0,97
I_4	0,13	0,69
I_5	0,78	1,00

Figura 4.10. O conceito de ‘bastante próximo’, utilizado pelo C-Focus.

Desta forma, quando duas instâncias de classes diferentes possuem os atributos selecionados (definido pelo subconjunto de atributos) bastante próximos, a seleção não é considerada consistente. O maior problema desta abordagem, é que o grau de similaridade a ser definido pelo usuário pode trazer resultados bastante diversos. Caso a variação proposta seja muito pequena, conjuntos com poucos atributos podem ser selecionados mais facilmente, pois serão poucos os casos de similaridade. Já com valores altos de variação, os valores serão muitas vezes similares, podendo resultar numa seleção falha (nenhum subconjunto de atributos é selecionado, por haver muita inconsistência).

Para encontrar um bom valor para o grau similaridade entre valores de atributos, é proposta nesse trabalho uma estratégia bastante prática e de simples implementação.

É considerado que o grau de similaridade escolhido deve antes de tudo permitir que o conjunto com todos os atributos originais seja consistente. Para isso, antes de iniciar o algoritmo C-Focus propriamente dito, são experimentados diferentes graus de similaridade para ser aplicado no conjunto de atributos original, iniciando-se em 50%. Este valor inicial é baseado nos experimentos realizados com as bases de dados descritas na Subseção 3.4.1.1, sendo considerado um valor suficientemente alto para seu propósito. O grau de similaridade então é decrementado de 1% em 1%, até que um determinado grau de similaridade faça com que o conjunto original de atributos seja analisado como inconsistente. O grau de similaridade anterior é então o grau máximo de similaridade que garante que ao menos o conjunto original de atributos é consistente. Este valor é chamado de grau de similaridade máximo (GSM).

Opcionalmente pode ser utilizado um decremento ainda menor, 0,01% por exemplo, embora em relação aos resultados obtidos com o decremento de 1% as diferenças tenham sido insignificantes, considerando os experimentos realizados nas bases de dados descritas na Subseção 3.4.1.1.

Como será mostrado na Subseção 4.2.6, referente aos experimentos realizados com o C-Focus, o grau de similaridade encontrado nas diferentes bases de dados ficou entre 3% e 27%, conforme as características dos dados. Foram experimentados também outros graus de similaridade, referentes a 90% e 80% do GSM.

4.2.6 Experimentos Realizados com o C-Focus e C-FocusD

Os experimentos realizados com o C-Focus e posteriormente com o C-FocusD foram todos realizados com as bases de dados descritas na Seção 3.4.1.1, embora não tenham sido incluídas nenhuma das bases de dados com mais de 34 atributos, já que a busca exaustiva realizada pelo C-Focus se torna proibitiva nestes casos. Assim, a base de dados Sonar não foi incluída nos experimentos realizados com este método de seleção de atributos.

Por se tratarem de bases de dados com dados contínuos, o C-Focus foi utilizado em conjunto com a estratégia proposta na Seção 4.2.5 para encontrar o grau de similaridade máximo (GSM) para cada uma das diferentes bases de dados. Optou-se também por

estudar o impacto de reduzir o GSM a 90% e 80% do valor encontrado, permitindo uma busca por subconjuntos menores, já que as condições de similaridade encontradas são menos freqüentes.

Na Tabela 4.8 são apresentados os GSM para todas as bases de dados utilizadas bem como os tamanhos dos subconjuntos encontrados utilizando diferentes graus de similaridade (100%, 90% e 80% do GSM). É perceptível uma sensível redução no tamanho dos subconjuntos encontrados, embora em alguns casos a alteração do grau de similaridade não tenha resultado em alteração no tamanho dos subconjuntos.

Tabela 4.8. Atributos selecionados pelo C-Focus com diferentes graus de similaridade.

Database	Total de Atributos	Grau de Similaridade Máximo	100% do GSM		90% do GSM		80% do GSM	
			Atributos Selecionados		Atributos Selecionados		Atributos Selecionados	
Íris	4	7,0%	4	100,00%	4	100,00%	3	75,00%
Breast	10	11,0%	3	30,00%	3	30,00%	3	30,00%
Glass	10	9,0%	5	50,00%	5	50,00%	4	40,00%
Wine	13	27,0%	10	76,92%	8	61,54%	6	46,15%
Vehicle	18	7,0%	7	38,89%	7	38,89%	7	38,89%
Segment	19	3,0%	5	26,32%	5	26,32%	5	26,32%
Waveform	21	14,0%	15	71,43%	13	61,90%	12	57,14%
WDBC	30	11,0%	9	30,00%	7	23,33%	6	20,00%
WPBC	33	18,0%	10	30,30%	8	24,24%	7	21,21%
Ionosphere	34	13,0%	7	20,59%	7	20,59%	6	17,65%
Média	**19,2**	**12,00%**	**7,5**	**47,44%**	**6,7**	**43,68%**	**5,9**	**37,24%**
(4-16)	9,25	13,50%	5,50	64,23%	5,00	60,38%	4,00	47,79%
(17-34)	25,83	11,00%	8,83	36,25%	7,83	32,55%	7,17	30,20%

Para avaliar a qualidade dos subconjuntos selecionados pelo C-Focus e comparar o desempenho dos diferentes graus de similaridade utilizados, nas Tabelas 4.9 e 4.10 são apresentados os desempenhos de classificação dos classificadores NN e DistAl (respectivamente), para o conjunto original de atributos e para os subconjuntos selecionados com o C-Focus. A mesma estratégia de avaliação de classificação adotada na avaliação do Relief-F e descrita na Subseção 4.1.5 foi utilizada para a obtenção dos resultados relatados nestas tabelas.

Tabela 4.9. Precisão com o NN dos subconjuntos de atributos selecionados pelo C-Focus.

Database	Precisão Original	Precisão (100% GSM)	Precisão (90% GSM)	Precisão (80% GSM)
Iris	94,67 ± 6,94	**94,67** ± 6,94	**94,67** ± 6,94	**94,67** ± 6,94
Breast	96,20 ± 3,68	95,61 ± 2,85	95,61 ± 2,85	95,61 ± 2,85
Glass	91,59 ± 8,23	88,79 ± 8,53	88,79 ± 8,53	**92,52** ± 9,66
Wine	97,75 ± 4,99	**97,75** ± 4,68	92,13 ± 11,77	91,01 ± 10,21
Vehicle	69,74 ± 10,40	61,70 ± 9,86	61,70 ± 9,86	61,70 ± 9,86
Segment	97,02 ± 1,46	91,53 ± 2,29	91,53 ± 2,29	91,53 ± 2,29
Waveform	77,15 ± 2,09	74,28 ± 2,17	73,35 ± 2,23	71,85 ± 2,19
WDBC	95,44 ± 3,71	**95,44** ± 6,00	**95,79** ± 5,49	**95,79** ± 2,24
WPBC	63,92 ± 14,72	**70,10** ± 14,98	**69,07** ± 13,19	**64,95** ± 15,24
Ionosphere	88,64 ± 5,37	**92,05** ± 5,99	**92,05** ± 5,99	**93,18** ± 6,30
Média	**87,21**	**86,19**	**85,47**	**85,28**
(4-16)	95,05	94,20	92,80	93,45
(17-34)	81,98	80,85	80,58	79,83

Tabela 4.10. Precisão com o DistAl dos subconjuntos de atributos selecionados pelo C-Focus.

Database	Precisão Original	Precisão (100% GSM)	Precisão (90% GSM)	Precisão (80% GSM)
Iris	92,86 ± 9,58	**92,86** ± 9,58	**92,86** ± 9,58	**94,29** ± 7,00
Breast	96,47 ± 2,56	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00
Glass	74,00 ± 10,20	70,00 ± 17,32	70,00 ± 17,32	**82,00** ± 9,80
Wine	97,50 ± 5,00	92,50 ± 6,12	91,25 ± 12,56	90,00 ± 12,25
Vehicle	54,76 ± 8,32	52,86 ± 3,81	52,86 ± 3,81	52,86 ± 3,81
Segment	88,93 ± 3,06	80,46 ± 2,67	80,46 ± 2,67	80,46 ± 2,67
Waveform	85,13 ± 1,73	83,05 ± 1,84	81,28 ± 2,28	80,23 ± 2,17
WDBV	96,07 ± 2,50	95,71 ± 3,85	95,71 ± 4,46	95,36 ± 3,21
WPBC	77,78 ± 14,91	76,67 ± 18,89	**77,78** ± 11,11	74,44 ± 11,17
Ionosphere	88,82 ± 6,14	**92,94** ± 6,34	**92,94** ± 6,34	**88,82** ± 8,50
Média	**85,23**	**73,70**	**73,51**	**73,85**
(4-16)	90,21	63,84	63,53	66,57
(17-34)	81,92	80,28	80,17	78,69

Os resultados obtidos sugerem que as variações do grau de similaridade utilizados não têm grande impacto quanto à precisão final obtida, embora utilizando o 100% do GSM tenha sido ligeiramente superior na maioria dos casos. No geral a precisão de classificação obtida com os subconjuntos selecionados não foi degradada em relação à precisão original, sobretudo com o NN.

Em algumas bases de dados, no entanto, a medida de consistência utilizada pelo C-Focus mostrou-se inadequada, já que a precisão de classificação de classificação foi degradada para ambos os classificadores empregados na avaliação dos subconjuntos. No

entanto, é importante avaliar que o número de atributos dos subconjuntos encontrados pelo C-Focus foi em geral pequeno, se comparado com os resultados obtidos com o Relief-F. A base de dados Breast apresentou um erro onde os atributos selecionados pelo C-Focus não permitiram a indução de um conceito com o DistAl, resultando na impossibilidade de classificação das instâncias.

Na Tabela 4.11 são mostrados os principais dados necessários para fazer uma análise comparativa entre a eficiência do C-FocusD frente ao C-Focus original. O espaço de busca de cada uma das bases de dados e o número de subconjuntos avaliados por cada um dos métodos é apresentado, permitindo uma comparação direta de quanto do espaço de busca é explorado por cada um deles. Para uma comparação entre as diferentes bases de dados, também é mostrada o percentual do espaço de busca explorado pelos dois métodos. A tabela apresenta ainda o tempo de execução dos algoritmos em segundos, onde fica claro que o processo de seleção do subconjunto inconsistente se torna mais lento quando o espaço de busca se torna muito grande.

Tabela 4.11. Comparação do número de subconjuntos explorados pelos métodos C-Focus e C-FocusD (GSM = 100%).

Database	Espaço de Busca	Atributos Selec.	C-Focus			C-FocusD		
			Tempo (s)	Subcjts. Avaliados	Espaço de Busca Exp.	Tempo (s)	Subcjts. Avaliados	Espaço de Busca Exp.
Iris	16	**100,00%**	0	16	100%	0	11	**68,75%**
Breast	1024	30,00%	0	113	11,04%	0	173	16,89%
Glass	1024	**50,00%**	0	479	46,78%	0	468	**45,70%**
Wine	8192	**76,92%**	0	7919	96,67%	0	4115	**50,23%**
Vehicle	262144	38,89%	6	48302	18,43%	8	55112	21,02%
Segment	524288	26,32%	1	5414	1,03%	1	5489	1,05%
Waveform	2,10E+06	**71,43%**	1356	2015251	96,09%	580	349115	**16,65%**
WDBC	1,07E+09	30,00%	2245	22905446	2,13%	2678	29911765	2,79%
WPBC	8,59E+09	30,30%	487	59136719	0,69%	461	54256735	**0,63%**
Ionosphere	1,72E+10	20,59%	198	3837810	0,02%	254	4841476	0,03%
Média	**2,68E+09**	**47,44%**	**429**	**8795746,9**	**37,29%**	**398**	**8942445,9**	**22,37%**

Com os resultados obtidos fica claro que o C-FocusD é eficiente quando o subconjunto final possui mais da metade dos atributos originais. Em todos os casos onde isso acontece o C-Focus original explora uma maior parte do espaço de busca, o mesmo acontecendo também na base de dados WPBC, onde houve uma pequena diferença na exploração do espaço de busca, favorável ao C-FocusD.

A média do percentual do espaço de busca explorado indica a vantagem para o C-

FocusD, apesar da média de subconjuntos avaliados do C-Focus original ser ligeiramente inferior. Esta diferença ocorre devido à exploração mais eficiente do C-Focus nas bases de dados que possuem um maior espaço de busca. Mesmo assim, o C-FocusD pode ser considerado mais eficiente pois em nenhum caso ultrapassou 70% de exploração do espaço de busca e nos casos onde a eficiência foi pior que no C-Focus original seu desempenho não divergiu de forma muito significativa.

4.3 A Família LVF

Esta seção estuda o algoritmo LVF e suas variantes, a saber: C-LVF, uma adaptação para dados contínuos, e os LVI/C-LVI, correspondentes a uma versão incremental do algoritmo LVF, desenvolvida para ser utilizada em bases de dados com grande número de instâncias.

4.3.1 LVF

O LVF[5] [Liu & Setiono 1996a] é um método de seleção de atributos do tipo filtro que, assim como o algoritmo Focus, verifica a inconsistência dos dados para selecionar os subconjuntos de atributos. No entanto, estes dois algoritmos são significativamente diferentes, tanto em relação ao método de busca utilizado quanto no processo de verificação da inconsistência dos subconjuntos.

O processo de busca é realizado com o algoritmo probabilístico conhecido como Las Vegas [Brassard & Bratley 1996]. Como visto na Subseção 3.3.4, este algoritmo utiliza a seleção randômica de atributos durante a busca, sem utilizar qualquer heurística de busca. Desta forma, esse algoritmo tem uma vantagem sobre aqueles que utilizam uma heurística em relação à seleção de atributos fortemente correlacionados. Apesar de utilizar um algoritmo probabilístico de busca, os autores afirmam que o algoritmo possui bom desempenho computacional, fato que foi verificado nos experimentos realizados e descritos ao longo desta seção. A seguir são apresentadas as principais características do LVF e do C-LVF, uma versão modificada do mesmo, capaz de tratar dados contínuos. Posteriormente são descritos e apresentados os resultados obtidos com o C-LVF.

[5] LVF é a forma abreviada de Las Vegas Filter, método de seleção de atributos classificado como filtro.

4.3.1.1 O Algoritmo LVF

O funcionamento do método de busca é de fato bastante eficiente, apesar de não utilizar heurísticas de busca. A cada iteração do *loop* principal é gerado um subconjunto de atributos S, cujos atributos são selecionados de forma randômica. Caso o número de atributos selecionados em S, denominado C, seja menor que o número de atributos do melhor subconjunto encontrado (C_{melhor}), ou seja, $C < C_{melhor}$, o algoritmo verifica se este subconjunto é consistente em relação a um critério estabelecido previamente (explicado posteriormente). Caso este subconjunto seja consistente, os valores de C_{melhor} e S_{melhor} são atualizados para corresponder ao melhor subconjunto encontrado até então e o sistema imprime o subconjunto em tempo de execução. No caso de $C = C_{melhor}$, o algoritmo também imprime este subconjunto, que possui características tão satisfatórias quanto o anteriormente encontrado. Este *loop* é executado MAX_TRIES vezes, sendo este valor definido inicialmente pelo usuário. Em [Liu & Setiono 1996a] o valor de MAX_TRIES é definido como $77 \times N$, sendo N o número de atributos da base de dados em questão. Este mesmo valor para MAX_TRIES foi utilizado nos experimentos realizados posteriormente nesta seção.

O critério de inconsistência é onde há diferenças em relação ao critério de seleção em relação ao Focus. A verificação de inconsistência de um determinado subconjunto retorna um valor real que é comparado com um parâmetro γ e caso seja inferior a este, é considerado consistente. A taxa de consistência é calculada considerando os três seguintes pontos:

1. Duas instâncias são inconsistentes se possuem valores similares em todos os atributos e classes diferentes;

2. Para as instâncias com mesmos valores, a contagem de inconsistência é o número total de instâncias similares menos as instâncias similares em maior número de uma determinada classe. Por exemplo, se existem n instâncias similares, sendo n_1 instâncias da classe c_1, n_2 da c_2 e n_3 da c_3, então $n = n_1 + n_2 + n_3$. No caso de n_3 ser o maior valor, a contagem de inconsistência seria $(n - n_3)$;

3. A taxa de inconsistência é a soma das contagens de inconsistência dividido pelo total de instâncias.

O pseudo-código do LVF é mostrado na Figura 4.11, adaptado de [Liu & Setiono 1996a].

```
LVF (D, N, MAX_TRIES, S);
{entrada: conjunto D = {i1,i2,...,in} com M instâncias, descritas por N atributos;
          MAX_TRIES, o valor de subconjuntos gerados;
          γ, taxa de inconsistência permitida.
 saída: subconjunto com K atributos que satisfazem o critério de inconsistência. }

Cmelhor = N;
Para i = 1 até MAX_TRIES faça     /* realizar o processo MAX_TRIES vezes */
    S = SubconjuntoRandomico( ); /* S recebe um subconjunto gerado randomicamente */
    C = numeroAtributos(S);      /* quantidade de atributos do subconjunto S */
    Se (C < Cmelhor) então       /* se a quantidade de atributos C for menor que a do
                                    melhor subconjunto Cmelhor */
        Se (verInconsist (S,D) <= γ) então   /* verifica se o subconjunto S é consistente
                                                no conjunto de dados D */
            Smelhor = S;  Cmelhor = C;   /* se sim, atualiza os valores */
            imprimeMelhor(S);            /* e mostra o subconjunto S */
        Fim Se;
    Fim Se;
    Senão
        Se (C = Cmelhor ) e (verInconsist (S,D) <= γ) então imprimeMelhor(S);
        /* no caso de ter a mesma quantidade de atributos e ser consistente,
        apenas imprime S */
    Fim Senão
Fim Para;
Retorna (S);
```

Figura 4.11. Pseudo-código do algoritmo LVF.

Como notado em [Liu & Setiono 1996a], o LVF é adequado apenas para atributos discretos. Os autores propõem que para empregar o LVF em atributos contínuos, eles podem ser discretizados ou tratados como atributos discretos (apenas em alguns casos). Trata-se do mesmo problema observado com o algoritmo Focus original, que foi adaptado para o C-Focus, uma versão adaptada aos atributos contínuos.

Utilizando a mesma abordagem empregada no C-Focus, é proposto neste trabalho o algoritmo C-LVF, uma variação do LVF capaz de tratar atributos contínuos sem qualquer processo anterior.

4.3.2 C-LVF

A alteração realizada no LVF para permitir atributos contínuos é realizada na verificação de similaridade entre duas instâncias, ou seja, no primeiro e segundo ponto

do ponto do cálculo da taxa de consistência. O primeiro ponto do LVF original diz que "duas instâncias são inconsistentes se possuem valores similares em todos os atributos e classes diferentes".

No C-LVF, duas instâncias são inconsistentes se possuem valores bastante próximos em todos os atributos selecionados e classes diferentes, onde bastante próximos denota uma variação percentual entre os valores (com a base de dados normalizada), conceito expresso na Figura 4.10.

O segundo ponto do cálculo da taxa de consistência, referente à contagem da inconsistência, também é alterado no C-LVF, já que "instâncias com mesmos valores" passam a ser caracterizadas como "instâncias com valores bastante próximos", considerando a mesma definição apresentada anteriormente.

4.3.3 Resultados obtidos com o C-LVF

Os experimentos realizados com o C-LVF, assim como nos outros métodos do tipo filtro, tiveram como finalidade verificar se os subconjuntos de atributos selecionados utilizando uma medida particular (de consistência no caso do LVF/C-LVF) são capazes de garantir uma boa precisão de classificação utilizando métodos de AM como o NN e o DistAl aqui utilizados.

Foram utilizadas todas as bases de dados descritas na Subseção 3.4.1.1, sem qualquer restrição, já que o C-LVF é capaz de tratar atributos contínuos e o método de busca probabilístico Las Vegas não exige grande poder computacional, mesmo quando o espaço de busca em questão é muito grande. Por se tratar de uma busca probabilística, cada execução do C-LVF pode retornar diferentes resultados e, assim sendo, o algoritmo C-LVF foi executado dez vezes para cada base de dados, sendo reportados os valores médios encontrados.

Na Tabela 4.12 pode ser observado o tamanho médio dos subconjuntos encontrados e também o número total de atributos original. O tempo de execução (em segundos) é também informado, havendo uma forte relação da duração do processo com a quantidade de instâncias de treinamento disponíveis, além da quantidade de atributos.

Tabela 4.12. Atributos selecionados pelo C-LVF e o tempo de processamento influenciado pelo número instâncias da base de dados.

Database	Total de Atributos	Atributos Selecionados	Tempo (s)	Instâncias
Iris	4	4	0	75
Breast	10	5,3	1	107
Glass	10	3	4	341
Wine	13	12,4	1	89
Vehicle	18	9,7	18	423
Segment	19	6,4	189	1000
Waveform	21	19,1	109	1000
WDBC	30	18,1	10	284
WPBC	33	16,2	2	97
Ionosphere	34	11,2	7	175
Sonar	60	22,6	8	104
Média	**22,91**	**11,64**	**31,73**	**335,91**
(4-16)	9,25	6,18	1,50	153,00
(17-34)	25,83	13,45	55,83	496,50
(60)	60,00	22,60	8,00	104,00

O número de atributos nos subconjuntos selecionados é grande em diversos casos e reduzido em outros, resultando uma média próxima da metade do total. Esta constatação não é muito interessante se for considerado o desempenho de classificação em geral inferior ao original (mesmo com metade dos atributos, em média), conforme mostra as Tabelas 4.13 e 4.14.

Tabela 4.13. Precisão com o NN dos subconjuntos de atributos selecionados pelo C-LVF.

Database	NN Original	NN C-LVF
Iris	94,67 ± 6,94	**94,67** ± 6,94
Breast	96,20 ± 8,23	92,40 ± 3,18
Glass	91,59 ± 3,68	90,65 ± 9,79
Wine	97,75 ± 4,99	96,63 ± 5,60
Vehicle	69,74 ± 10,40	63,59 ± 9,88
Segment	97,02 ± 1,46	94,96 ± 1,09
Waveform	77,15 ± 2,09	74,18 ± 1,16
WDBV	95,44 ± 3,71	94,04 ± 4,93
WPBC	63,92 ± 14,72	62,89 ± 15,08
Ionosphere	88,64 ± 5,37	**90,91** ± 8,89
Sonar	80,77 ± 8,10	77,88 ± 9,18
Média	**86,63**	**84,80**
(4-16)	95,05	93,59
(17-34)	81,99	80,10
(60)	80,77	77,88

Na Tabela 4.13 são mostrados os resultados da precisão de classificação do NN utilizando o conjunto original de atributos e os subconjuntos selecionados pelo C-LVF. Já na Tabela 4.14 os resultados da precisão de classificação são relativos ao DistAl. É bastante evidente que, em geral, o C-LVF não é capaz de manter a precisão de classificação obtida com todos os atributos, mesmo não selecionando subconjuntos muito pequenos.

Tabela 4.14. Precisão com o DistAl dos subconjuntos de atributos selecionados pelo C-LVF.

Database	DistAl Original	DistAl C-LVF
Iris	92,86 ± 9,58	**92,86** ± 9,58
Breast	96,47 ± 2,56	0,00 ± 0,00
Glass	74,00 ± 10,20	**77,00** ± 16,16
Wine	97,50 ± 5,00	92,50 ± 8,29
Vehicle	54,76 ± 8,32	51,67 ± 5,74
Segment	88,93 ± 3,06	85,27 ± 2,56
Waveform	85,13 ± 1,73	83,80 ± 1,55
WDBC	96,07 ± 2,50	95,00 ± 4,29
WPBC	77,78 ± 14,91	**83,33** ± 10,24
Ionosphere	88,82 ± 6,14	**92,94** ± 5,76
Sonar	70,00 ± 12,65	**74,00** ± 8,00
Média	**83,85**	**75,31**
(4-16)	90,21	65,59
(17-34)	81,92	82,00
(60)	70,00	74,00

Nos resultados obtidos com o DistAl é importante ressaltar que com a base de dados Breast ele teve sérios problemas para a classificação das instâncias após a seleção de atributos realizada pelo C-LVF. A seleção de atributos não foi satisfatória neste caso, mesmo o problema sendo nitidamente do classificador (que apresentou problemas para induzir o conceito, mas apenas com esta base de dados).

4.3.4 LVI

Os mesmos autores do algoritmo LVF também desenvolveram sua versão incremental, chamada de LVI[6] [Liu & Setiono 1998]. O critério de avaliação dos subconjuntos é o mesmo utilizado no LVF, assim como o método de busca, que também é o Las Vegas. A única diferença é a abordagem incremental, que se caracteriza por não utilizar todas

[6] LVI é a forma abreviada de Las Vegas Incremental Filter, método de seleção de atributos incremental.

as instâncias durante o processo inicial de busca, podendo utilizar-se incrementalmente delas. Mesmo assim, utilizando uma estratégia de verificação de consistência, o método garante que o subconjunto final seja consistente entre todas as instâncias de treinamento, gerando resultados semelhantes aos encontrados com o LVF.

Este método é particularmente interessante quando a quantidade de instâncias disponíveis para o treinamento é extremamente grande, onde o método LVF encontra problemas relativos ao tempo de processamento, como já foi verificado nos experimentos realizados e mostrados na Subseção 4.3.3, onde com 1000 instâncias de treinamento e menos de 20 atributos o tempo de processamento foi bastante superior ao obtido em bases de dados com 60 atributos (ver Tabela 4.12).

A motivação dos autores foi tratar bases de dados com mais de 5 mil instâncias de treinamento, cuja aplicação do algoritmo LVF levou algumas horas de processamento conforme relatado em [Liu & Setiono 1998]. Assim, o LVI não é um algoritmo particularmente adequado para bases de dados com poucas instâncias, sendo o LVF normalmente uma opção mais vantajosa nestes casos.

4.3.4.1 O Algoritmo LVI e o C-LVI

O algoritmo do LVI não só é derivado do LVF, como também utiliza este algoritmo como parte de seu processo de seleção de atributos. Além do próprio algoritmo LVF, é incluída uma função chamada de checkIncon( ) com as mesmas características da função inconCheck( ) presente no LVF e uma outra característica adicional. Todas as instâncias que são inconsistentes na verificação são armazenadas em um vetor, chamado de inconData.

O LVI inicialmente seleciona um determinado percentual (definido pelo usuário) de instancias do conjunto de treinamento original D, armazenando-as em D_0. Em D_1 são armazenadas as instâncias remanescentes, presentes em D, mas não em D_0. É necessário também definir inicialmente uma taxa de inconsistência γ, usualmente definida como 0. Então é aplicado o algoritmo LVF em D_0, que contem apenas uma fração das instâncias originais. O subconjunto de atributos encontrado é retornado em ‘subcjt’. Este subconjunto é então utilizado para verificar se o restante das instâncias, presentes em D_1, também é consistente, utilizando o método checkIncon( ), que retorna possíveis instâncias inconsistentes no vetor ‘inconData’. Caso obedeça ao critério de consistência,

ou seja, seja inferior à $\gamma - \beta$, este subconjunto de atributos 'subcjt' é retornado como resultado final do algoritmo. Caso contrário, o algoritmo adiciona à D_0 as instâncias presentes em 'inconData' e as remove de D_1.

Da mesma forma que o LVF, o LVI original não é capaz de tratar dados com valores contínuos, já que este é derivado do algoritmo LVF. Em nossos experimentos utilizamos o algoritmo C-LVF, proposto na seção anterior, ao invés do LVF, tornando o algoritmo LVI apto a tratar dados contínuos. Esta implementação foi batizada de C-LVI, um algoritmo de seleção de atributos probabilístico incremental capaz de tratar atributos contínuos.

```
LVI (D, N, MAX_TRIES, p, subcjt)
{entrada: conjunto D = {i1,i2,...,in} com M instâncias, descritas por N atributos;
          MAX_TRIES, o valor de subconjuntos gerados;
          γ, taxa de inconsistência permitida;
          p, a porcentagem inicial dos dados de D utilizados na seleção de atributos.
 saída: subconjunto com K atributos que satisfazem o critério de inconsistência. }

D0 = copiaDataset(p,D);          /* copia p% dos dados de D, de forma randômica */
D1 = D - D0                      /* D1 armazena o restante dos dados */
Executar = true;                 /* variável booleana que controla a execução do loop */

Enquanto ( Executar e  (D0 ≠ D) ) faça
    β = LVF(D0, γ, subcjt);      /* o algoritmo LVF é executado sobre D0, retornando o
                                    subconjunto subcjt; g é a taxa de inconsistência */
    Se ( checkIncon(subcjt, D1, inconData) <= γ - β ) então
        Executar = false;        /* se o a consistência encontrada em D1 utilizando subcjt
                                    for inferior à g - b (aquela encontrada em D0), parar
                                    execução para retornar subcjt */
    Fim Se;
    Senão                            /* caso a consistência em D1 for superior */
        Adiciona(inconData, D0);     /* adiciona as instancias que apresentaram
                                        inconsistência em D0 */
        Remove(inconData, D1);       /* e às remove de D0 */
    Fim Senão
Fim Para;
Retorna (subcjt);                /* caso executar seja alterado para false, retornar subcjt */
```

Figura 4.12. Pseudo-código do algoritmo LVI.

4.3.5 Resultados obtidos com o C-LVI

Os experimentos realizados com o C-LVI apresentam dois aspectos adicionais além da verificação do tamanho do subconjunto selecionado e da precisão de classificação utilizando algoritmos de AM (estes dois comuns aos demais experimentos realizados nesta dissertação).

O primeiro deles é verificar se o caráter incremental deste algoritmo é capaz de auxiliar na redução da carga de processamento mantendo a mesma capacidade de seleção do C-LVF.

O segundo aspecto é relativo à porcentagem de instâncias necessárias para a realização da busca pelos subconjuntos consistentes. Seguindo os parâmetros adotados em [Liu & Setiono 1998], os experimentos foram iniciados com D_0 igual a 20% das instâncias disponíveis para treinamento havendo o incremento das instâncias que apresentavam inconsistência em D_1 durante a execução do método. Ao final do processo, o total de instâncias presentes em D_0 foi verificado, a fim de avaliar se os 20% iniciais são uma boa escolha.

Os experimentos seguiram os mesmos moldes do C-LVF, descritos em 4.3.3. Por se tratar de um algoritmo incremental probabilístico, todos os experimentos foram realizados dez vezes para cada base de dados.

Na Tabela 4.15 estão presentes os resultados sobre todas as bases de dados descritas em 3.4.1.1, com dados sobre a quantidade de atributos selecionados, o tempo de execução do processo (em segundos) e também a porcentagem final de instâncias utilizadas durante a busca.

Em relação ao tamanho dos subconjuntos selecionados, observa-se grande semelhança com os resultados obtidos com o C-LVF, um fato esperado, já que os dois algoritmos utilizam a mesma função de avaliação (consistência). Em relação ao tempo de execução, não há uma melhoria significativa, havendo diversos casos onde o C-LVI levou ainda mais tempo para encontrar um bom subconjunto consistente.

Tabela 4.15. Atributos selecionados pelo C-LVI, o tempo de processamento influenciado pelo número instâncias da base de dados e a porcentagem utilizada para análise.

Database	Total de Atributos	Atributos Selecionados	Tempo (s)	Instâncias	Instâncias Utilizadas
Iris	4	4	0	75	53,33%
Breast	10	5,3	1	107	30,75%
Glass	10	3	3	341	30,29%
Wine	13	12,2	2	89	80,90%
Vehicle	18	7,8	30	423	37,59%
Segment	19	5,8	136	1000	23,69%
Waveform	21	16,9	144	1000	32,22%
WDBC	30	16,7	10	284	35,99%
WPBC	33	16,7	5	97	84,85%
Ionosphere	34	11,4	19	175	68,97%
Sonar	60	20,2	15	104	70,00%
Média	**22,91**	**10,91**	**33,18**	**335,91**	**49,87%**
(4-16)	9,25	6,13	1,50	153,00	48,82%
(17-34)	25,83	12,55	57,33	496,50	47,22%
(60)	60,00	20,20	15,00	104,00	70,00%

O desempenho computacional pouco satisfatório do C-LVI pode ser analisado como um problema resultante do porcentual inicial utilizado. Caso o valor seja muito baixo o algoritmo precisa executar o C-LVF internamente muitas vezes, pois a verificação de consistência em D_1 falha insistentemente. Nas bases de dados onde o tamanho final de D_0 não cresceu muito, o desempenho do C-LVI foi mais satisfatório, mas de fato este algoritmo é adequado apenas quando estão disponíveis grandes bases de dados.

Nas Tabelas 4.16 e 4.17 são apresentados os resultados da precisão de classificação encontrados para os subconjuntos selecionados com o C-LVI para o NN e para o DistAl, respectivamente. Como pode ser notado, a medida de consistência utilizada pelo C-LVI não é capaz de otimizar ou manter a precisão de classificação em muitas bases de dados, apresentando resultados similares àqueles encontrados com o C-LVF e apresentados nas Tabelas 4.13 e 4.14.

Tabela 4.16. Precisão com o NN dos subconjuntos de atributos selecionados pelo C-LVI.

Database	NN Original	NN C-LVI
Iris	94,67 ± 6,94	**94,67** ± 6,94
Breast	96,20 ± 8,23	93,86 ± 3,18
Glass	91,59 ± 3,68	**92,52** ± 9,44
Wine	97,75 ± 4,99	95,51 ± 7,91
Vehicle	69,74 ± 10,40	53,19 ± 6,71
Segment	97,02 ± 1,46	96,64 ± 0,97
Waveform	77,15 ± 2,09	74,15 ± 1,68
WDBC	95,44 ± 3,71	**96,14** ± 3,48
WPBC	63,92 ± 14,72	**64,95** ± 8,45
Ionosphere	88,64 ± 5,37	**90,91** ± 8,44
Sonar	80,77 ± 8,10	72,12 ± 7,08
Média	**86,63**	**84,06**
(4-16)	95,05	94,14
(17-34)	81,99	79,33
(60)	80,77	72,12

Tabela 4.17. Precisão com o DistAl dos subconjuntos de atributos selecionados pelo C-LVI.

Database	DistAl Original	DistAl C-LVI
Iris	92,86 ± 9,58	**92,86** ± 9,58
Breast	96,47 ± 2,56	0,00 ± 0,00
Glass	74,00 ± 10,20	**75,00** ± 13,60
Wine	97,50 ± 5,00	96,25 ± 5,73
Vehicle	54,76 ± 8,32	49,52 ± 4,23
Segment	88,93 ± 3,06	88,32 ± 2,92
Waveform	85,13 ± 1,73	83,08 ± 1,18
WDBC	96,07 ± 2,50	**95,71** ± 3,85
WPBC	77,78 ± 14,91	**76,67** ± 14,44
Ionosphere	88,82 ± 6,14	88,24 ± 5,26
Sonar	70,00 ± 12,65	**76,00** ± 8,00
Média	**83,85**	**74,70**
(4-16)	90,21	66,03
(17-34)	81,92	80,26
(60)	70,00	76,00

4.4 Conclusões sobre os Filtros

Os métodos de seleção do tipo filtro apesar de terem em comum com os *wrappers* a finalidade de reduzir a dimensionalidade das bases de dados, não são comprometidos com a precisão de classificação final obtida com um algoritmo de aprendizado. Por utilizar medidas outras que não a taxa de erro do classificador (como nos *wrappers*), os filtros estão sujeitos a degradar a precisão de classificação de um método de aprendizado, fato este que pôde ser verificado nos experimentos descritos neste

capítulo.

Sobretudo nos filtros que utilizam como função de avaliação a medida de consistência os resultados mostraram que a seleção de atributos não foi capaz de, na média, otimizar a precisão de classificação dos algoritmos de aprendizado. Analisando que estes métodos tentam aproximar o conceito de relevância forte, teoricamente selecionando poucos atributos fracamente relevantes, é possível concluir que, nas bases de dados onde houve degradação da precisão, mais atributos com relevância fraca seriam fundamentais para garantir uma boa precisão de classificação.

No geral, entretanto, a precisão não foi significativamente reduzida. Um único caso sério relacionado com o DistAl e a base de dados Breast foi verificado, onde o algoritmo de aprendizado não conseguiu induzir o conceito após a seleção de atributos, resultando em 0% de precisão de classificação. Esta incompatibilidade foi verificada com os três algoritmos que utilizam uma medida de consistência como função de avaliação.

Já com o Relief-F o desempenho em relação à precisão foi melhor, havendo na média uma melhoria em relação à utilização do conjunto original de atributos, desde que usados os parâmetros adequados. Quando foram selecionados subconjuntos de atributos com aproximadamente metade do conjunto original os resultados foram bons, em geral.

Considerando apenas a capacidade de redução da dimensionalidade os filtros cumpriram bem a tarefa. Desta forma, eles são ainda uma alternativa viável para a seleção de atributos, considerando que o custo computacional necessário nestes métodos é bastante inferior ao necessário pelos métodos do tipo *wrapper*, com exceção do método C-Focus, que utiliza a busca exaustiva, tornando-o bastante custoso computacionalmente. É importante considerar também que os métodos filtro possuem maior generalidade, não se adaptando apenas a um método de AM.

Em comparação com os *wrappers*, os filtros não conseguiram igualar a melhoria da precisão de classificação nos métodos de aprendizado específicos. Mesmo com o Relief-F, que obteve os melhores resultados entre os filtros, a precisão de classificação obtida não foi tão boa quanto nos *wrappers*. Este algoritmo, no entanto, possui baixíssimo custo computacional, sendo ideal para a redução de grandes bases de dados, com alta dimensionalidade. Uma boa alternativa é utilizar o algoritmo Relief-F como passo anterior da aplicação de um *wrapper*, a fim de reduzir o custo computacional necessário na utilização do algoritmo de AM durante a busca pelo melhor subconjunto.

5 CAPÍTULO

Conclusões

Este trabalho de pesquisa teve como principal objetivo o estudo dos métodos de seleção de atributos caracterizados como *wrapper* e filtro, de forma a permitir a avaliação de ambos quanto à capacidade de redução da dimensionalidade, otimização da precisão de classificação em métodos de AM e custo computacional. O trabalho também investiga a sensibilidade de alguns dos métodos ao processo de busca utilizado.

Após uma visão geral sobre os principais conceitos que delineiam o problema da seleção de atributos, envolvendo o conceito de atributo relevante, a organização dos métodos de seleção de atributos e a caracterização dos métodos *wrapper* e filtro, os experimentos realizados proporcionaram uma visão prática da aplicação de tais métodos. A investigação dos *wrappers* envolveu a implementação de cinco diferentes métodos de busca, onde o desempenho de cada um deles pode ser avaliado quanto à eficiência em encontrar bons subconjuntos de atributos e em relação ao esforço necessário para conseguir os resultados finais. A avaliação das diferentes estratégias de direcionamento da busca permitiu analisar seus impactos na busca, confrontando a dimensionalidade do subconjunto *versus* precisão de classificação.

Com os filtros implementados foi possível verificar o baixo custo computacional do método Relief, a complexidade da busca completa executada pelo Focus e também a eficiência do método de busca Las Vegas para encontrar boas soluções, embora não ótimas.

Como principais contribuições desta dissertação destacam-se as seguintes:

- Avaliação de cinco métodos de busca para utilização com *wrappers*, onde destacaram-se os métodos RBC, *Beam Search* e AGs;

- Avaliação do impacto de três diferentes estratégias de direcionamento de busca, *Forward Selection*, *Backward Elimination* e *Random*, sendo evidenciada a tendência da *Forward* em selecionar subconjuntos de atributos mínimos e da *Backward* em selecionar subconjuntos com maior precisão de classificação;
- Identificação do método de busca RBC como a melhor solução de baixo custo computacional para ser empregada em *wrappers* (dentre as avaliadas), garantindo ótimos resultados em relação à dimensionalidade e precisão de classificação;
- Avaliação de diferentes *thresholds* associados ao Relief-F, permitindo determinar os valores mais adequados a serem utilizados;
- Proposta e desenvolvimento da variante do Focus denominada FocusD, que otimiza a busca pelo menor subconjunto consistente, explorando o espaço de busca de forma a evitar regiões com maior densidade de subconjuntos;
- Proposta e desenvolvimento de uma metodologia para a definição do grau de similaridade utilizado pelo método C-Focus, permitindo encontrar valores adequados a cada base de dados;
- Adaptação dos algoritmos LVF e LVI a dados contínuos, utilizando o mesmo critério adotado pelo C-Focus.

Esta linha de pesquisa ainda pode ser explorada nas seguintes frentes:

- Estudo de demais filtros, que utilizem outras funções de avaliação;
- Integração entre os métodos *wrapper* e filtro;
- Uso de métodos de aprendizado Bayesianos em *wrappers*;
- Avaliação da eficiência de seleção de atributos dos algoritmos de AM que integram alguma estratégia de seleção de atributos (modelo integrado);
- Estudar uma forma não randômica de inversão dos bits no método de busca RBC.

REFERÊNCIAS BIBLIOGRÁFICAS•

[Aha & Bankert 1995]

AHA, D. W.; BANKERT, R.L. A Comparative Evaluation of Sequential Feature Selection Algorithms. In *Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics*, 1-7, 1995. 13, 40.

[Almuallim & Dietterich 1991]

ALMUALLIM, H.; DIETTERICH, T. G. Learning with Many Irrelevant Features. In *Proceedings of the 9th National Conference on Artificial Intelligence*, 547-552, 1991. 11, 13, 17, 22, 26, 34, 91, 92, 98.

[Almuallim & Dietterich 1992]

ALMUALLIM, H.; DIETTERICH, T. G. Efficient Algorithms for Identifying Relevant Features. In *Proceedings of the 9th Canadian Conference on Artificial Intelligence*, 35-45, 1992. 34, 93, 94, 95, 98.

[Amaldi & Kann 1998]

AMALDI, E.; KANN, V. On the Approximability of Minimizing Nonzero Variables or Unsatisfied Relations in Linear Systems. *Theoretical Computer Science*, 209:237-260, 1998. 39.

[Arauzo et al. 2003]

ARAUZO, A.; BENITEZ, J. M.; CASTRO, J. L. C-Focus: A Continuous Extension of Focus. In J. Benitez, O. Cordon, F. Hoffmann & R. Roy (eds.), *Advances in Soft Computing – Engineering, Design and Manufacturing*, 225-232, Springer, London, 2003. 34, 99.

[Beale & Jackson 1994]

BEALE, R.; JACKSON T. *Neural Computing – An Introduction.* Institute of Physics Publishing, Bristol, UK, 1994. 140.

[Beasley et al. 1993]

BEASLEY, D.; RALPH, R. M.; DAVID, R. B. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, 15(2):58-69, 1993. 131.

• Em todas as referências aqui apresentadas são indicadas todas as páginas nas quais as mesmas são citadas no texto (após o ano de publicação e separadas por vírgulas).

[Ben-Bassat 1982]

BEN-BASSAT, M. Pattern Recognition and Reduction of Dimensionality. In P. R. Krishnaiah & L. N. Kanal (eds.), *Handbook of Statistics II*, 2:773-791, 1982. 28, 30.

[Blake & Merz 1998]

BLAKE, C. L.; MERZ, C. J. *UCI Repository of Machine Learning Databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998. 55.

[Blum et al. 1999]

BLUM, A. I.; KALAI, A.; LANGFORD, J. Beating the Hold-out: Bounds for K-fold and Progressive Cross-Validation. In *Proceedings of the 12th International Conference on Computational Learning Theory*, 203-208, 1999. 8.

[Blum & Langley 1997]

BLUM, A. I.; LANGLEY, P. Selection of Relevant Features and Examples in Machine Learning. *Journal of Artificial Intelligence*, 97(1-2):245-271, 1997. 21, 23, 29, 31.

[Borrajo & Veloso 1997]

BORRAJO, D; VELOSO, M. Lazy Incremental Learning of Control Knowledge for Efficiently Obtaining Quality Plans. *AI Review Journal (Special Issue on Lazy Learning)*, 11(1-5):371-405, 1997. 3.

[Brassard & Bratley 1996]

BRASSARD, G.; BRATLEY, P. *Fundamentals of Algorithms*. Prentice Hall, New Jersey, 1996. 50, 105.

[Breiman et al. 1984]

BREIMAN, L.; FRIEDMAN, J. H.; OLSHEN, R. A.; STONE, C. J. *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA, 1984. 25, 78.

[Briscoe & Caelli 1996]

BRISCOE, G.; CAELLI, T. *A Compendium of Machine Learning, Volume 1: Symbolic Machine Learning*, Ablex, 1996. 4.

[Carbonell et al. 1991]

CARBONELL, J. G.; KNOBLOCK, C. A.; MINTON, S. Prodigy: An Integrated Architecture for Planning and Learning. In K. VanLehn (ed.), *Architecture for Intelligence*, 241-278, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991. 4.

[Cardie 1993]

CARDIE, C. Using Decision Trees to Improve Case-based Search. In *Proceedings of the 10th International Conference on Machine Learning*, 25-32, 1993. 35.

[Caruana & Freitag 1994]

CARUANA R.; FREITAG, D. Greedy Attribute Selection. In *Proceedings of the 11th International Conference on Machine Learning*, 28-36, 1994. 33.

[Caruana & de Sa 1998]

CARUANA, R.; DE SA, V. R. Using Feature Selection to Find Inputs that Work Better as Extra Outputs. In *Proceedings of the 8th International Conference on Artificial Neural Networks*, 299-304, 1998. 38.

[Caruana & de Sa 2003]

CARUANA, R.; DE SA, V. R. Benefiting from the Variables that Variable Selection Discards. *Journal of Machine Learning Research*, 3:1245-1264, 2003. 38.

[de Castro et al. 2004]

DE CASTRO, P. A. D.; SANTORO, D. M.; CAMARGO, H. A.; NICOLETTI, M.C. Improving a Pittsburgh Learnt Fuzzy Rule Base Using Feature Subset Selection. In *Proceedings of the 4th International Conference Hybrid Intelligent Systems Conference*, 180-185, Kitakyushu, Japan, 2004. 10.

[Chang & Lippmann 1991]

CHANG, E.; LIPPMANN, R. Using Genetic Algorithms to Improve Pattern Classification Performance. In R. P. Lippmann, J. E. Moody & D. S. Touretzky (eds.), *Advances in Neural Information Processing Systems*, 3:797-803, Morgan Kaufmann, San Mateo, CA, 1991. 51.

[Clark & Niblett 1989]

CLARK, P.; NIBLETT, T. The CN2 Induction Algorithm. *Machine Learning Journal*, 3(4):261-283, 1989. 11.

[Coley 1999]

COLEY, D. A. An *Introduction to Genetic Algorithms for Scientists and Engineers*. Singapore, World Scientific, 1999. 133.

[Cover & Hart 1967]

COVER, T. M.; HART, R. E. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13 (1):21-27, 1967. 40.

[Cunningham 2000]

CUNNINGHAM, P. Overfitting and Diversity in Classification Ensembles Based on Feature Selection. Technical Report, Department of Computer Science, Trinity College, Dublin, 2000. 52.

[Dasarathy 1990]

DASARATHY, B. V. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA. 140.

[Dash & Liu 1997]

DASH, M.; LIU, H. Feature Selection for Classification. *Intelligent Data Analysis*, 1(3):131-156, Elsevier Science, 1997. 10, 14, 23, 26, 27, 28, 29, 32, 37.

[Davis 1991]

DAVIS, L. Bit-climbing, Representational Bias, and Test Suite Design. In R. K. Belew, & L. B. Booker (eds.), In *Proceedings of the 4th International Conference on Genetic Algorithms*, 18-23,1991. 40, 45, 46.

[Dietterich 1990]

DIETTERICH, T. G. Machine learning. *Annual Review of Computer Science*, 4:255-306, 1990. 2.

[Dietterich 1997]

DIETTERICH, T. G. Machine Learning Research: Four Current Directions. *Artificial Intelligence Magazine*, 18(4):97-136, 1997. 11.

[Dietterich 2003]

DIETTERICH, T. G. Machine Learning. *Nature Encyclopedia of Cognitive Science*, London: Macmillan, 2003. 6.

[Devijver & Kittler 1982]

DVIJVER, P. A.; KITTLER, J. *Pattern Recognition: A Statistical Approach*. Prentice Hall, 1982. 24, 32.

[Doak 1992]

DOAK, J. An Evaluation of Feature Selection Methods and their Application to Computer Security. Technical Report CSE-92-18, Department of Computer Science, University of California, Davis, CA, 1992. 13, 32, 33.

[Duran & Odell 1974]

DURAN, B. S.; ODELL, P. L. *Cluster Analysis – A Survey*. New York: Springer-Verlag, Berlin, 1974. 36.

[Efron & Tibshirani 1993]

EFRON, B.; TIBSHIRANI, R. *An Introduction to the Bootstrap*. Chapman & Hall, 1993. 9.

[Frean 1990]

FREAN, M. The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation*, 2(2):198-209, 1990. 138.

[Foroutan & Sklansky 1987]

FOROUTAN, I.; SKLANSKY, J. Feature Selection for Automatic Classification of Non-Gaussian Data. *IEEE Transactions on Systems, Man and Cybernetics*, 17(2):187-198, 1987. 32.

[Gallant 1990]

GALLANT, S. Perceptron-Based Learning Algorithms. *IEEE Transactions on Neural Networks*, 1(2):179-191, 1990. 138.

[Gates 1972]

GATES, G. W. The Reduced Nearest Neighbor Rule. *IEEE Transactions on Information Theory*, 18(3):431-433, 1972. 140.

[Gennari et al. 1989]

GENNARI, J. H.; LANGLEY, P.; FISHER, D. Models of Incremental Concept Formation. *Artificial Intelligence*, 40:11-62, 1989. 18.

[Ginsberg 1993]

GINSBERG, M. *Essentials Of Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, 1993. 47.

[Goldberg 1989]

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley Publishing, 1989. 40, 130.

[Guerra-Salcedo et al. 1999]

GUERRA-SALCEDO, C.; CHEN, S.; WHITLEY, D.; SMITH, S. Fast and Accurate Feature Selection Using Hybrid Genetic Strategies. In *Proceedings of the Congress on Evolutionary Computation*, 1:177-184, 1999. 45, 51.

[Guyon & Elisseeff 2003]

GUYON, I.; ELISSEEFF, A. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157-1182, 2003. 12, 37, 41.

[Han et al. 1996]

HAN, J.; CHEN M.; YU P. Data Mining: An Overview from Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866-883, 1996. 13.

[Hartley 1998]

HARTLEY, J. *Learning and Studying. A Research Perspective*. London: Routledge, 1998. 2.

[Holland 1975]

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, University of Michigan Press, 1975. 130.

[Ichino & Sklansky 1984]

ICHINO, M.; SKLANSKY, J. Feature Selection for Linear Classifier. In *Proceedings of the 7th International Conference on Pattern Recognition*, 1:124-127, 1984. 32.

[Jain et al. 2000]

JAIN, A.; DUIN, P.; MAO J. Statistical Pattern Recognition: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4-37, 2000. 9.

[Ji & Ma 1999]

JI, C.; MA, S. Performance and Efficiency: Recent Advances in Supervised Learning. In *Proceedings of the IEEE*, 87:1519-1535, 1999. 11.

[John et al. 1994]

JOHN, G.; KOHAVI, R.; PFLEGER, K. Irrelevant Features and the Subset Selection Problem. In *Proceedings of the 11th International Conference on Machine Learning*, 121-129, 1994. 13, 14, 17, 20, 22, 30, 31.

[Kearns & Ron 1997]

KEARNS, M. J.; RON, D. Algorithmic Stability and Sanity-Check Bounds for Leave-One-Out Cross-Validation. In *Proceedings of the 10th Annual Conference on Computational Learning Theory*, 152-162, Association for Computing Machinery (ACM) Press, 1997. 8.

[Kira & Rendell 1992]

KIRA, K.; RENDELL, L. A Practical Approach to Feature Selection. In *Proceedings of the 9th International Conference on Machine Learning*, 249-256, Morgan Kaufmann, 1992. 11, 13, 14, 22, 34, 78, 80, 83, 85.

[Kittler 1978]

KITTLER, J. Feature Set Search Algorithms. In C. H. Chen (ed.), *Pattern Recognition and Signal Processing*, 41-60, 1978. 41.

[Kohavi 1995]

KOHAVI, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1137-1145, 1995. 7, 8, 9.

[Kohavi & John 1997]

KOHAVI, R.; JOHN, G. Wrappers for Feature Selection. *Artificial Intelligence: Special Issue on Relevance*, 97(1-2):273-324, 1997. 22, 25, 31, 44, 48, 52.

[Kohavi & Sommerfield 1995]

KOHAVI, R.; SOMMERFIELD, D. Feature Subset Selection Using the Wrapper Method: Overfitting and Dynamic Search Space Topology. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, 192-197, 1995. 11, 52

[Koller & Sahami 1996]

KOLLER, D.; SAHAMI, M. Toward Optimal Feature Selection. In *Proceedings of the 13th International Conference on Machine Learning*, 284-292, 1996. 14, 16, 35, 36.

[Kononenko 1994]

KONONENKO, I. Estimating Attributes: Analysis and Extension of Relief. In *Proceedings of European Conference on Machine Learning*, 171-182, 1994. 22, 34, 83, 84, 85.

[Lacerda & Carvalho 1999]

LACERDA, E. G. M.; CARVALHO, A. C. P. L. F. Introdução aos Algoritmos Genéticos. In *Jornada de Atualização em Informática, Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação*, 1999. 131, 133.

[Langley 1994]

LANGLEY, P. Selection of Relevant Features in Machine Learning. In *Proceedings of the American Association for Artificial Intelligence Fall Symposium on Relevance*, 140-144, 1994. 10.

[Langley & Simon 1995]

LANGLEY, P.; SIMON, H. A. Applications of Machine Learning and Rule Induction. *Communications of the Association for Computing Machinery (ACM)*, 38(11):55-64, 1995. 3.

[Liu & Motoda 1998]

LIU H.; MOTODA H. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Press, 1998. 10.

[Liu & Setiono 1996a]

LIU, H.; SETIONO, R. A Probabilistic Approach to Feature Selection – A Filter Solution. In *Proceedings of the 13th International Conference on Machine Learning*, 319-327, 1996. 31, 35, 105, 106, 107.

[Liu & Setiono 1996b]

LIU, H.; SETIONO, R. Feature Selection and Classification – A Probabilistic Wrapper Approach. In *Proceedings of 9th International Conference on Industrial and Engineering Applications of AI and ES*, 284-292, 1996. 33, 40, 50.

[Liu & Setiono 1998]

LIU, H.; SETIONO, R. Incremental Feature selection. *Applied Intelligence*, 9(3):217-230, 1998. 35, 110, 111, 113.

[Liu & Yu 2002]

LIU, H.; YU, L. Feature Selection for Data Mining. [http://www.public.asu.edu/~huanliu/sur-fs02.ps], 2002. 32, 33.

[Liu et al. 2002]

LIU, H.; MOTODA, H.; YU, L. Feature Selection with Selective Sampling. In *Proceedings of the 19th International Conference on Machine Learning*, 395-402, 2002. 12.

[Loughrey & Cunningham 2005]

LOUGHREY, J.; CUNNINGHAM, P. Overfitting in Wrapper-Based Feature Subset Selection: The Harder You Try The Worse It Gets. Technical Report TCD-CS-2005-17, Computer Science Department, Trinity College Dublin, 2005. 52.

[Mézard & Nadal 1989]

MÉZARD, M.; NADAL, J. Learning Feedforward Layered Networks: The Tiling Algorithm. *Journal of Physics A: Mathematical and General*, 22(12):2191-2203, 1989. 138.

[Mitchell 1982]

MITCHELL, T. M. Generalization as Search. *Artificial Intelligence*, 18:203-226, 1982. 92.

[Mitchell 1997]

MITCHELL, T. M. *Machine Learning*. McGraw-Hill Companies Inc., 1997. 3, 11, 14, 140.

[Michie et al. 1994]

MICHIE D.; SPIEGELHALTER D. J.; TAYLOR C. C. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994. 15.

[Modrzejewski 1993]

MODRZEJEWSKI, M. Feature Selection Using Rough Sets Theory. In P. B. Brazdil (ed.), In *Proceedings of the European Conference on Machine Learning*, 213-226, 1993. 35.

[Mucciardi & Gose 1971]

MUCCIARDI, A. N.; GOSE, E. E. A Comparison of Seven Techniques for Choosing Subsets of Pattern Recognition Properties. *IEEE Transactions on Computers*, C-20(9):1023-1031, 1971. 13, 35.

[Narendra & Fukunaga 1977]

NARENDRA, P. M.; FUKUNAGA, K. A Branch and Bound Algorithm for Feature Selection. *IEEE Transactions on Computers*, C-26(9):917-922, 1977. 14, 35, 36.

[Nicoletti 1994]

NICOLETTI, M. C. *Ampliando os Limites do Aprendizado de Máquina Através das Abordagens Construtiva e Relacional.* Tese (Doutorado). Instituto de Física de São Carlos – Universidade de São Paulo, 1994. 4, 5.

[Palma Neto & Nicoletti 2003]

PALMA NETO, L. G.; NICOLETTI, M.C. Aprendizado Neural Construtivo – O Algoritmo DistAl. RT-DC 002/03, São Carlos, 2003. 139.

[Pawlak 1984]

PAWLAK, Z. Rough Sets. *International Journal of Man-Machine Studies*, 21:127-134, 1984. 36.

[Portinale & Saitta 2002]

PORTINALE, L.; SAITTA, L. *Feature Selection*. Deliverable D14.1, Università Del Piemonte Orientale Spalgo Marengo, Italy, 2002. 14.

[Quinlan 1986]

QUINLAN, J. R. Induction of Decision Trees. *Machine Learning Journal*, 1(1):81-106, 1986. 25.

[Quinlan 1993]

QUINLAN, J. R. *C4.5: Program for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993. 11, 25, 78.

[Reich & Barai 1999]

REICH, Y.; BARAI, S. V. Evaluating Machine Learning Models for Engineering Problems. *Artificial Intelligence in Engineering*, 13(3):257-272, 1999. 6.

[Reunanen 2003]

REUNANEN, J. Overfitting in Making Comparisons Between Variable Selection Methods. *Journal of Machine Learning Research*, 3:1371-1382, 2003. 52, 53.

[Rissanen 1978]

RISSANEN, J. Modeling by the Shortest Data Description. *Automatica*, 14:465-471, 1978. 35.

[Saitta 2001]

SAITTA, L. Integrated Architectures for Machine Learning. In G. Paliouras, V. Karkaletsis e C. D. Spyropoulus (eds.), *Machine Learning and Its Applications 2001: Advanced Lectures*, 2049:218-229, Springer, 2001. 11.

[Santoro & Nicoletti 2005]

SANTORO, D. M.; NICOLETTI, M. C. Investigating a Wrapper Approach for Selecting Features Using Constructive Neural Networks. *International Conference on Information Technology: Coding & Computing*, April 4-6, Las Vegas, USA, 2005. 29.

[Sheinvald et al. 1990]

SHEINVALD, J.; DOM, B.; NIBLACK, W. A Modeling Approach to Feature Selection. In *Proceedings of the 10th International Conference in Pattern Recognition*, 1:535-539, 1990. 35.

[Siedlecki & Sklansky 1988]

SIEDLECKI, W.; SKLANSKY, J. On Automatic Feature Selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(2):197-220, 1988. 30.

[Smith 1999]

SMITH, M. K. Learning Theory. *The Encyclopedia of Informal Education*, [http://www.infed.org/biblio/b-learn.htm], 1999. 1.

[Smyth & Goodman 1992]

SMYTH, P.; GOODMAN, R. M. An Information Theoretic Approach to Rule Induction from Databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(4):301-316, 1992. 78.

[Talavera 1999]

TALAVERA L. Feature Selection as a Preprocessing Step for Hierarchical Clustering. In *Proceedings of the 16th International Conference on Machine Learning*, 389-397, 1999. 10, 14.

[Utgoff 1989]

UTGOFF, P. F. Incremental Induction of Decision Trees. *Machine Learning Journal*, 4(2):161-186, 1989. 5.

[Vafaie & De Jong 1993]

VAFAIE, H.; DE JONG, K. Robust Feature Selection Algorithms. In *Proceedings of the Fifth IEEE International Conference on Tools for Artificial Intelligence*, 356-363, 1993. 40.

[Vafaie & Imam 1994]

VAFAIE, H.; IMAM, I. F. Feature Selection Methods: Genetic Algorithms vs. Greedy-like Search. In *Proceedings of the International Conference on Fuzzy and Intelligent Control Systems*, 1994. 51.

[Walker 1967]

WALKER E. L. C*onditional and Instrumental Learning.* Wadsworth Publishing Company Inc., 1967. 2.

[Wettschereck et al. 1997]

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*, 11:273-314, 1997. 15.

[Wurst et al. 2002]

WURST, M.; NOVAK, J.; SCHNEIDER, M. Integrating Different Machine Learning Methods to Support Search in Cross-domain Information Sources – The Project Awake. In *Proceedings of the FGML Workshop*, 2002. 11.

[Yang et al. 1999]

YANG, J.; PAREKH, R.; HONAVAR, V. DistAl: An Inter Pattern Distance Based Constructive Learning Algorithm. *Intelligent Data Analysis 3*, 53-73, 1999. 40,138, 139.

[Yang & Honavar 1998]

YANG, J.; HONAVAR, V. Feature Subset Selection Using a Genetic Algorithm. *IEEE Intelligent Systems (Special Issue on Feature Transformation and Subset Selection)*, 13(2):44-49, 1998. 11, 26, 51, 52.

APÊNDICE A
CONSIDERAÇÕES SOBRE OS ALGORITMOS GENÉTICOS

A.1 Introdução

Os Algoritmos Genéticos (AGs) são métodos de busca e otimização inspirados no processo de evolução e na seleção natural proposta por Charles Darwin no século XIX. Os princípios básicos dos AGs foram propostos por John Holland em [Holland 1975], mas ganhou maior repercussão apenas a partir da década de 80 [Goldberg 1989].

O princípio da seleção natural, no qual se baseiam os AGs, diz que quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será a chance de sobreviver e gerar descendentes, enquanto que indivíduos menos adaptados tendem a desaparecer durante o processo evolutivo. Nos AGs, este processo de seleção é realizado em torno de possíveis soluções de um problema, que evoluem no intuito de encontrar a melhor solução possível.

Os AGs possuem algumas características que os diferem de outras técnicas de busca e otimização convencionais [Goldberg 1989]:

- Trabalham com uma codificação dos dados e não com os próprios dados;
- Utilizam informações obtidas pelo próprio método, não utilizando conhecimento auxiliar;
- Trabalham com diversas soluções potenciais e não apenas uma;
- Utilizam para guiar a busca regras de transição probabilísticas e não determinísticas.

Estas características fazem com que os AGs sejam métodos de busca robustos, sobretudo quando o espaço de busca é extenso, irregular ou desconhecido. Em situações

onde existem métodos de otimização ou busca específicos, os AGs normalmente são superados tanto na precisão da busca como na velocidade.

Uma importante consideração sobre AGs, apontada em [Beasley et al. 1993], é a de que apesar desses algoritmos não garantirem que a solução global ótima para um dado problema será encontrada, eles geralmente são eficientes para encontrar soluções aceitáveis, rapidamente.

Um AG depende basicamente de uma codificação específica para as possíveis soluções em cada problema, de uma função de avaliação, capaz de avaliar a qualidade de cada possível solução e também de métodos para garantir a evolução, conhecidos como operadores genéticos. Estes conceitos são apresentados nas duas subseções que seguem.

A.2 Terminologia e Conceitos Básicos

A terminologia utilizada com os AGs usa de inúmeros termos trazidos da própria Genética e da Biologia. Um cromossomo (ou hipótese) é uma possível solução do problema sendo otimizado, correspondendo a um ponto no espaço de busca. Os cromossomos são normalmente formados por uma série de parâmetros, em uma cadeia de símbolos (ou genes), que utilizam uma codificação/representação específica para cada problema (normalmente números binários ou reais). Um conjunto de cromossomos forma uma população, que representa um conjunto de pontos no espaço de busca.

A população inicial corresponde ao espaço de busca inicial, podendo ser gerada aleatoriamente ou então utilizando hipóteses já conhecidas como interessantes para o problema, em uma técnica conhecida como *seeding* [Lacerda & Carvalho 1999]. A partir da população inicial, os cromossomos evoluem através de sucessivas iterações, chamadas de gerações.

A formação de novas gerações envolve a criação de novos cromossomos (chamados de descendentes ou filhos), gerados a partir de cromossomos selecionados da geração corrente (chamados de pais), que posteriormente passam por uma fase de reprodução.

A seleção dos pais que irão gerar descendentes depende da avaliação que os cromossomos recebem utilizando-se uma função de aptidão. Uma função de aptidão avalia o quão bom um cromossomo é, aferindo a sua adequabilidade ao problema em

questão.

Na fase de reprodução, os pais selecionados podem ser combinados através de um operador de cruzamento e podem sofrer mutação, através da alteração dos genes dos cromossomos.

Portanto, uma nova geração é formada da seguinte forma:

- Seleciona-se alguns cromossomos da população corrente, de acordo com seus valores de aptidão (usualmente quanto melhores os cromossomos, maior a probabilidade de seleção);
- Combina-se alguns dos cromossomos selecionados utilizando um operador de cruzamento;
- Modifica-se alguns genes através de um operador de mutação.

Após algumas gerações criadas desta forma, espera-se que o AG leve a busca a convergir para a melhor solução possível, podendo ser ótima ou quase ótima, segundo a função de aptidão utilizada.

O AG só termina a sua execução, interrompendo a criação de novas gerações, quando um determinado critério de parada é satisfeito. As paradas podem ocorrer quando:

- Um determinado número de gerações é atingido;
- Um determinado tempo é atingido;
- O valor ótimo da função de aptidão é atingido, desde que seja possível conhecê-lo;
- Observa-se pouca melhora no melhor cromossomo durante um determinado número de gerações (convergência);
- Observa-se que grande parte da população possui a mesma aptidão.

A forma de representação das hipóteses é também fundamental para que um AG possa resolver um problema com sucesso. As duas formas mais difundidas de codificação para as hipóteses são a binária e a real. Em ambos os casos, diversos parâmetros (ou genes) são aninhados em uma *string*. No caso da representação binária

cada gene é representado por 0 ou 1, já na representação real cada gene é representado por números reais, ampliando a capacidade de representação.

A representação binária é normalmente utilizada para representar variáveis discretas, enquanto que a representação real é mais adequada para variáveis contínuas. No problema de seleção de atributos utilizando um *wrapper* (ou mesmo um filtro) a representação binária é utilizada. Portanto, um AG simples consiste de (baseado em [Coley 1999]):

- Algumas soluções em potencial do problema (hipóteses), com a devida representação (binária ou real);
- Uma maneira de calcular a qualidade de cada uma das soluções individuais da população (função de aptidão);
- Um método capaz de selecionar os melhores indivíduos de uma geração para compor a próxima geração (operador de seleção);
- Um método para compor partes entre as diversas soluções, de forma a criar novas soluções (operador de cruzamento);
- Uma forma de alterar os cromossomos para inserir variabilidade na população (operador de mutação);
- Uma condição para o término de sua execução (critério de parada).

Todo este procedimento pode ser abrangido em um algoritmo genético canônico, como pode ser observado na Figura A.1 [Lacerda & Carvalho 1999].

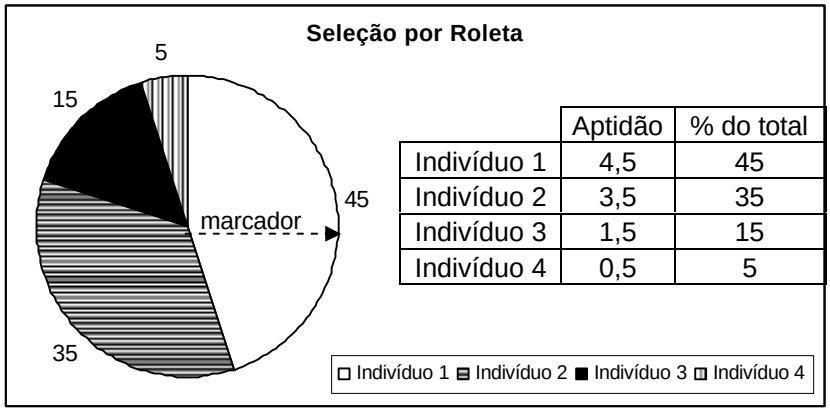
```
g = 0;              /* variável g representa a geração corrente */
inicializar P(g);   /* inicializa a população inicial P(0), gerando hipóteses randomicamente */
avaliar P(g);       /* avalia a qualidade de cada possível solução pela função de aptidão */

/* loop para a criação de novas gerações; critério de parada determina a interrupção */
enquanto (critério de parada não estiver satisfeito) faça
    g = g + 1;
    selecionar P(g) a partir de P(g – 1);   /* seleciona hipóteses da geração anterior */
    aplicar cruzamento sobre P(g);          /* cruzamento entre hipóteses selecionadas */
    aplicar mutação sobre P(g);             /* alteração de alguns genes */
    avaliar P(g);                           /* avaliação da nova geração de hipóteses */
fim enquanto;
```

Figura A.1. Algoritmo Genético canônico.

A.3 Operadores Genéticos

Em qualquer aplicação utilizando um AG, uma especial atenção deve ser dada aos operadores genéticos. A seleção se encarrega de selecionar os pais (geradores) da próxima geração, normalmente os melhores indivíduos, que posteriormente poderão sofrer alterações pelo cruzamento e mutação, gerando seus filhos. Para manter os melhores indivíduos na busca, a seleção é aplicada utilizando técnicas onde usualmente os indivíduos menos aptos são descartados (não obrigatoriamente), permanecendo apenas os mais adaptados e seus descendentes. As técnicas de seleção mais difundidas são:

- **Seleção por *Ranking***: os indivíduos são classificados em um *ranking* segundo suas aptidões; passam para a próxima geração N_{pop} indivíduos (onde N_{pop} representa o tamanho da população), sendo que os mais bem colocados no *ranking* tem maiores chances de serem selecionados e podem ser selecionados mais de uma vez;
- **Seleção por Torneio**: uma quantidade n de indivíduos (normalmente 2) são escolhidos aleatoriamente (com probabilidades iguais), sendo que a cada confronto é selecionado para a próxima geração apenas aquele com a maior aptidão. O processo é repetido N_{pop} vezes.
- **Seleção por Roleta**: os indivíduos possuem probabilidade de seleção proporcional a suas aptidões. Há analogia com uma roleta, pois considera-se que cada indivíduo possui uma fatia na roleta proporcional à sua aptidão, como mostra a Figura A.2. A seleção dos indivíduos ocorre com N_{pop} giros da roleta, onde indivíduos com maior aptidão possuem maior probabilidade de serem selecionados pelo 'marcador' a cada giro.



	Aptidão	% do total
Indivíduo 1	4,5	45
Indivíduo 2	3,5	35
Indivíduo 3	1,5	15
Indivíduo 4	0,5	5

Figura A.2. Representação gráfica de uma 'roleta' na Seleção por Roleta.

Após a seleção, normalmente são aplicados outros dois operadores genéticos. O cruzamento realiza uma troca de segmentos de dois cromossomos (pais) selecionados para a geração de novos cromossomos (filhos). Com o cruzamento, busca-se propagar as características dos indivíduos mais aptos, trocando informações entre eles. As principais técnicas de cruzamento para uma representação binária, adequada para a seleção de atributos, são (ver Figura A.3):

- **Cruzamento de um ponto ou padrão**: um ponto de cruzamento no vetor (cromossomo) é escolhido aleatoriamente, havendo uma troca de informação genética entre os pais a partir deste ponto. Um dos pais irá concatenar os seus genes (bits na representação binária) até o ponto de cruzamento com os genes do outro pai após o ponto de cruzamento, gerando um novo cromossomo; outro cromossomo irá concatenar as outras duas partes não utilizadas para a geração do primeiro cromossomo filho.
- **Cruzamento multiponto**: é uma expansão do cruzamento de um ponto; são selecionados n pontos de cruzamento, onde as informações genéticas são trocadas entre os pais a cada ponto.
- **Cruzamento uniforme**: não são selecionados pontos de cruzamento, mas sim posições de troca, através de uma máscara de bits. Em cada posição definida os genes são trocados entre os pais.

Após o cruzamento, alguns dos indivíduos sofrem a ação do último operador genético, a mutação. A mutação é essencial para introduzir e manter a diversidade genética da população. E, no caso da representação binária, é realizada através da alteração aleatória dos bits dos cromossomos. O processo é bastante simples quando é utilizada a representação binária, havendo apenas a escolha de algumas posições no cromossomo que têm seus valores alterados de 0 para 1 e de 1 para 0.

Uma técnica adicional que pode ser utilizada na construção de uma nova geração é o elitismo, onde um número pré-determinado dos melhores indivíduos é passado para a próxima geração da forma como são, sem ser aplicado cruzamento ou mutação. Desta forma garante-se que as melhores hipóteses encontradas durante a busca não serão descartadas.

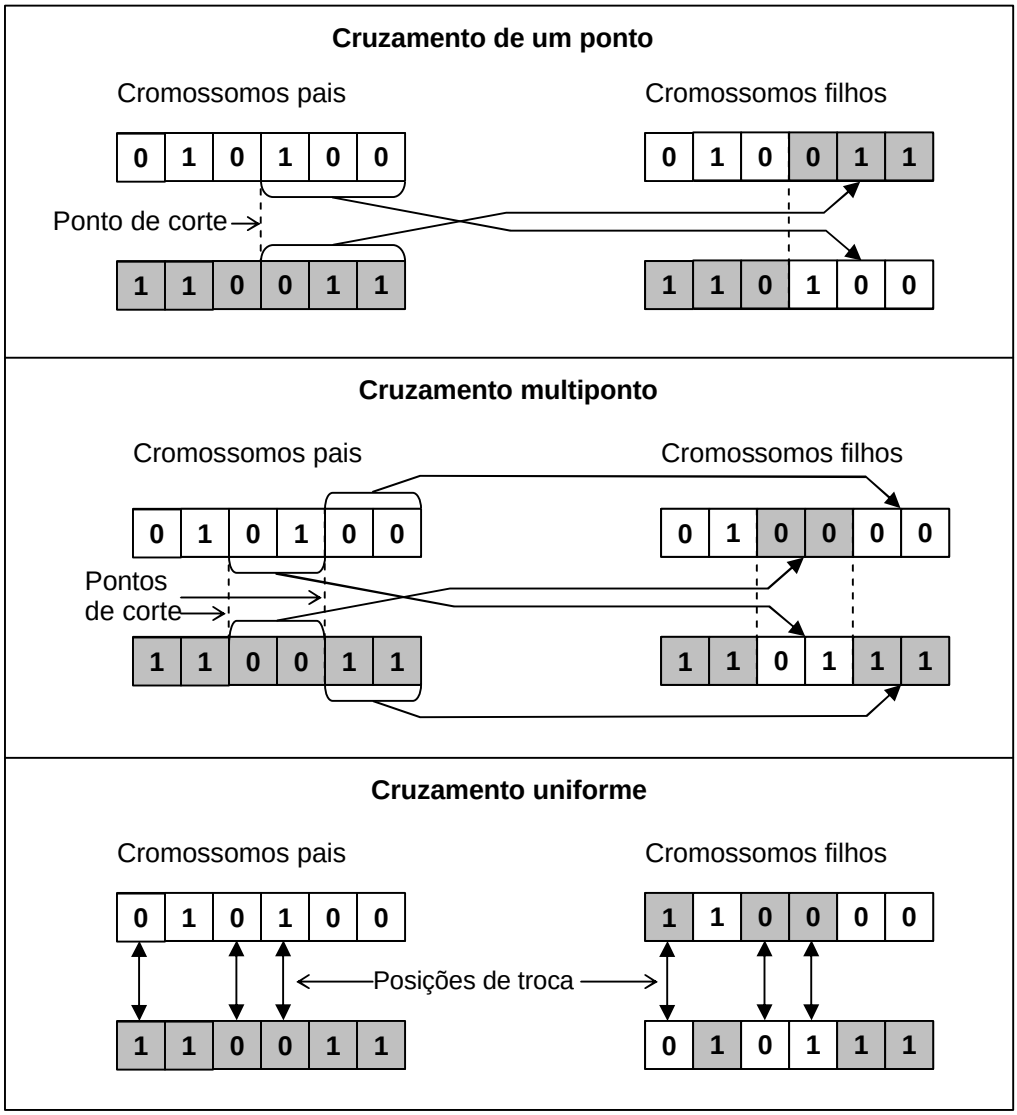


Figura A.3. Exemplos de cruzamento para representação binária.

É importante ressaltar que o cruzamento e a mutação não ocorrem sobre todos os indivíduos. O cruzamento ocorre com uma probabilidade de aplicação usualmente entre 60% e 100%. Probabilidades de cruzamento mais altas permitem uma exploração maior do espaço de busca, evitando a convergência para máximos locais. No entanto, probabilidades muito altas podem acabar resultando em explorações pouco promissoras, desperdiçando poder computacional.

Já a probabilidade de ocorrer mutação nos bits de um cromossomo é normalmente baixa (abaixo de 1%), para garantir uma variabilidade na população sem torná-la demasiadamente aleatória.

A.4 Outras Considerações sobre os AGs

Para que um algoritmo genético obtenha bons resultados é fundamental que os operadores genéticos sejam corretamente selecionados, a codificação adequada seja utilizada e que os parâmetros genéticos como o tamanho da população e as taxas de aplicação dos operadores genéticos sejam apropriadamente definidos.

O tamanho da população, por exemplo, deve ser definido criteriosamente, de forma a não ser muito pequeno, restringindo a cobertura do espaço de busca, e também não muito grande, exigindo mais processamento computacional.

Outros parâmetros com grande relevância em um AG são as taxas de aplicação dos operadores genéticos. As taxas de mutação e cruzamento devem ser escolhidas e avaliadas com bastante critério, para que a variabilidade das hipóteses seja adequada a cada problema.

APÊNDICE B
OS ALGORITMOS DE APRENDIZADO UTILIZADOS

B.1 Redes Neurais Construtivas e DistAl

Aprendizado neural construtivo é um modelo de aprendizado neural que não pressupõe a definição de uma topologia de rede fixa, antes do início do treinamento. Os algoritmos neurais construtivos são capazes de determinar dinamicamente a topologia da rede, introduzindo novos neurônios, camadas e conexões [Yang et al. 1999].

Em geral, a idéia básica que os algoritmos neurais construtivos implementam segue o mesmo princípio. Inicialmente treina-se um único neurônio e verifica-se se ele é capaz de classificar corretamente as instâncias de treinamento; caso este neurônio não seja apto a classificar corretamente as instâncias, novos neurônios são adicionados dinamicamente, até que a classificação de todo o conjunto de treinamento esteja correta.

Os algoritmos neurais construtivos possuem as seguintes vantagens sobre o aprendizado neural convencional [Yang et al. 1999]:

- Não necessitam a definição de uma topologia para a rede neural;
- Tem grandes chances de gerar redes ótimas (quase mínimas);
- Garantem classificações sem erros em qualquer conjunto de treinamento finito e não contraditório;
- Utilizam neurônicos básicos que são treinados pelo perceptron;
- Não exigem a definição de parâmetros.

Na literatura já foram propostos diversos algoritmos neurais construtivos, a saber, os algoritmos Tower [Gallant 1990], Pyramid [Gallant 1990], Tiling [Mézard & Nadal 1989], Upstart [Frean 1990], DistAl[Yang et al. 1999], entre outros.

O DistAl, um dos algoritmos utilizado neste trabalho como algoritmo de aprendizado na avaliação dos subconjuntos de atributos, é particularmente interessante por não utilizar um mecanismo iterativo para calcular os pesos e *threshold*s dos neurônios criados. Esta característica torna o DistAl mais ágil que os outros algoritmos neurais construtivos, algo muito atrativo para aplicações onde são utilizados grandes volumes de dados ou em sistemas de aprendizado híbridos onde o método de aprendizado faz parte de um processo maior [Yang et al. 1999], como no caso da seleção de atributos.

Outra característica que diferencia o DistAl dos outros algoritmos neurais construtivos é que ele utiliza neurônios esféricos na única camada intermediária que cria. Um neurônio esférico é uma variação dos TLUs (*Threshold Logic Units*); a cada um deles estão associados um vetor de pesos e dois valores de *threshold* (limites superior e inferior). A criação e definição dos limites dos neurônios intermediários depende de uma métrica de distância.

A idéia do DistAl é adicionar neurônios esféricos intermediários um a um, de forma que cada neurônio seja capaz de, usando alguma métrica de distância, identificar corretamente o maior grupo de instâncias de treinamento possível pertencente a uma determinada classe. As instâncias já corretamente classificadas por algum neurônio não são consideradas na adição de novos neurônios, de forma que o algoritmo termina quando todas as instâncias de treinamento são ‘cobertas’ pelos neurônios inseridos.

Ao final do processo de construção, a rede neural construída tem um total de neurônios esféricos intermediários sempre menor que o total de instâncias de treinamento, a não ser que o conjunto de treinamento possua irregularidades (seja contraditório). Como em todo algoritmo neural construtivo, a construção da rede acontece simultaneamente com o seu treinamento.

A implementação do DistAl possui alguns detalhes relevantes a serem ressaltados, como a utilização de diversas métricas para avaliar a distância entre as instâncias. A implementação disponibiliza 23 diferentes métricas, dentre elas a Euclidiana, a Manhattan e o valor máximo, sendo 12 delas variações normalizadas das três citadas (ver [Palma Neto & Nicoletti 2003] para maiores detalhes).

Por meio do uso de diferentes métricas são encontrados valores distintos quanto à precisão do aprendizado. Cada situação de aprendizado pode se mostrar mais adequada

à determinada métrica e, por isso, é interessante que todas as métricas sejam experimentadas para a seleção da mais adequada a um determinado domínio. Durante a seleção de atributos, optou-se por utilizar apenas uma métrica para todas as situações (a euclidiana normaliza), de forma a reduzir o processamento computacional necessário.

B.2 Nearest Neighbor

O algoritmo conhecido como *Nearest Neighbor* (NN) é um método de AM bastante simples, comumente identificado como um método *lazy* (preguiçoso) [Mitchell 1997]. É assim denominado por não realizar um processamento de aprendizado até que uma instância tenha que ser classificada. Por ser extremamente simples e eficiente, o NN é um algoritmo utilizado em diversas aplicações e bastante explorado na literatura (ver, por exemplo, [Beale & Jackson 1994] [Dasarathy 1990]). Seu funcionamento é brevemente descrito a seguir (ver [Mitchell 1997] para maiores detalhes).

Considere um espaço bidimensional definido por dois atributos, x_1 e x_2, onde estão representadas quatro instâncias de classe A e quatro instâncias de classe B. Considere, ainda, a presença de uma instância Z, cuja classe é desconhecida. Esta situação é mostrada na Figura B.1. O algoritmo NN irá classificar a instância Z com a classe da instância mais próxima de Z (utilizando-se distância euclidiana, por exemplo).

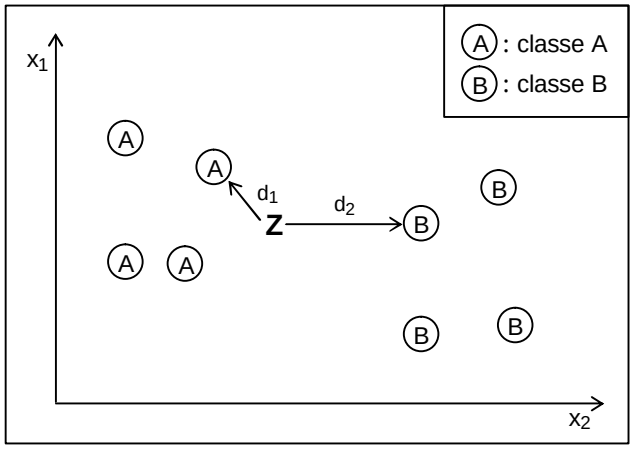


Figura B.1. A figura representa a instancia Z, a ser classificada pelo NN. A menor distância (d_1) para uma instância da classe A leva o algoritmo a atribuir a classe A para Z.

O NN atribui como classe da instância Z a mesma da instância com a menor distância (d_1), ou seja, a classe A. A Figura B.2 descreve a definição formal do NN, com base na descrição encontrada em [Gates 1972].

Considere:

- Espaço n-dimensional de atributos;
- M classes, numeradas 1, 2, ... , M;
- p instâncias de treinamento, cada uma representada como um par (x_i, θ_i), para $1 \leq i \leq p$, onde:

  a) x_i é uma instância de treinamento, representada por um vetor de pares atributo-valor:

  $$x_i = (x_{i_1}, x_{i_2}, ..., x_{i_n})$$

  b) $\theta_i \in \{1, 2, ... , M\}$ denota a classe correta da instância x_i.

Seja $T_{NN} = \{(x_1,\theta_1), (x_2,\theta_2), ..., (x_p,\theta_p)\}$ o conjunto de treinamento do NN.

Dada uma instância desconhecida x, o NN decide que x está na classe θ_j se:

$$d(x, x_j) \leq d(x, x_i) \quad 1 \leq i \leq p$$

onde d(x, y) é alguma métrica n-dimensional de distância.

Figura B.2. Definição formal do algoritmo NN.