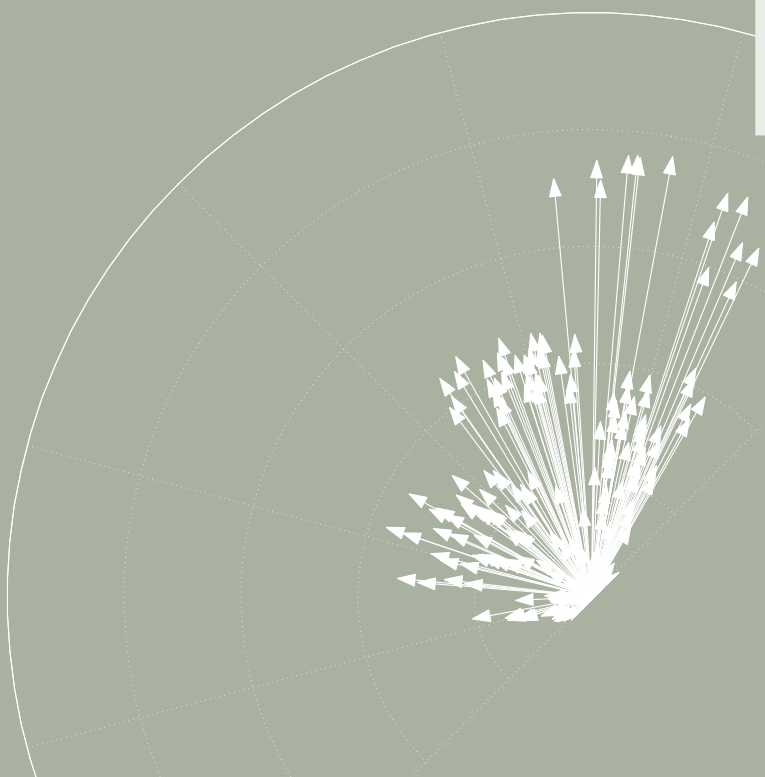
MANUAL DE COMPUTAÇÃO EVOLUTIVA E METAHEURÍSTICA

ANTÓNIO GASPAR-CUNHA
RICARDO TAKAHASHI
CARLOS HENGGELER ANTUNES
COORDENADORES



IMPRENSA DA UNIVERSIDADE DE COIMBRA
COIMBRA UNIVERSITY PRESS

EDITORA ufmg

CAPÍTULO 18

Algoritmos Genéticos em Problemas de Classificação

Márcio P. Basgalupp [*] *André Rossi* [**]

Ana C. Lorena [**] *André C. P. L. F. de Carvalho* [**]

[*] *Instituto de Ciência e Tecnologia*
Universidade Federal de São Paulo

[**] *Instituto de Ciências Matemáticas e de Computação*
Universidade de São Paulo - São Carlos

Muitas técnicas de Aprendizado de Máquina (AM) utilizam conhecimentos da Inteligência Artificial e da Estatística para construção de modelos capazes de adquirir conhecimento a partir de um conjunto de dados. Os exemplos desse conjunto são chamados de dados de treinamento e a partir desses faz-se a inferência indutiva, que pode gerar hipóteses verdadeiras ou não (Alpaydin, 2004; Monard e Baranauskas, 2003). Todo algoritmo de aprendizado, ou algoritmo de indução, ou simplesmente indutor, possui um viés (*bias*) indutivo, que é a preferência de uma hipótese sobre outra, que não são igualmente prováveis. Os exemplos de um conjunto de dados são formados por atributos e cada atributo especifica uma característica particular para esse conjunto. O aprendizado indutivo pode ser dividido em aprendizado supervisionado e não-supervisionado. Atualmente o aprendizado semi-supervisionado também tem atraído considerável atenção da comunidade de AM (Chapelle et al., 2006).

No aprendizado supervisionado, cada exemplo apresentado ao algoritmo de aprendizado possui um atributo especial que especifica o rótulo da classe real à qual este exemplo pertence. Se os rótulos das classes forem discretos, o problema é conhecido como classificação; se forem contínuos, como regressão ou aproximação de funções.

No aprendizado não-supervisionado ou agrupamento, o algoritmo de aprendizado não tem conhecimento dos rótulos das classes reais. Dessa maneira, o algoritmo agrupa os exemplos por meio de extração de padrões nos valores de seus atributos.

No aprendizado semi-supervisionado, exemplos em que os rótulos das classes são conhecidos e também exemplos em que os rótulos das classes não são conhecidos são apresentados ao algoritmo de aprendizado. O algoritmo utiliza, assim, tanto os exemplos rotulados como os não rotulados durante o aprendizado.

Neste capítulo serão discutidas algumas alternativas para o uso de computação evolutiva, especificamente algoritmos genéticos, no projeto e combinação de modelos de classificação. Segundo (Ye, 2003), o objetivo da classificação é construir um modelo conciso de distribuição do atributo classe (ou alvo) em função demais atributos, denominados atributos preditivos. O resultado desse modelo pode ser utilizado para atribuir valores a exemplos onde somente seus atributos preditivos são conhecidos.

Em um problema de classificação, os dados de entrada podem ser descritos pelo par ordenado (X, y), em que X é um vetor que representa os atributos preditivos, $X = (x_1, x_2, x_3, ..., x_n)$, e y é o rótulo da classe à qual esse exemplo pertence. Na Tabela 18.1 tem-se um conjunto de dados para classificação do estado de saúde de um paciente. Nessa tabela, cada linha representa um exemplo do conjunto de dados e cada coluna um atributo desse exemplo. O atributo *Diagnóstico* é especial, pois possui o rótulo da classe para cada exemplo, ou seja, doente ou saudável.

Tabela 18.1: Conjunto de dados para o diagnóstico da saúde de pacientes.

Exemplo	**Febre**	**Enjôo**	**Manchas**	**Dor**	**Diagnóstico**
T1	sim	sim	pequenas	sim	doente
T2	não	não	grandes	não	saudável
T3	sim	sim	pequenas	não	saudável
T4	sim	não	grandes	sim	doente
T5	sim	não	pequenas	sim	saudável
T6	não	não	grandes	sim	doente

Técnicas de AM têm sido utilizadas em problemas de classificação e cada uma utiliza um algoritmo de aprendizado para construir um modelo (classificador) que relaciona os atributos e os rótulos das classes. Um ponto importante dos algoritmos de aprendizado é construir modelos que possuam boa capacidade de generalização, ou seja, consigam predizer, com alta taxa de acerto, rótulos das classes para exemplos que não foram apresentados anteriormente (Tan et al., 2005).

Outra característica dos classificadores a ser observada diz respeito à interpretabilidade do conhecimento adquirido. Os classificadores do tipo caixa-preta são aqueles que possuem uma representação interna que, geralmente, não se consegue interpretar facilmente, ou seja, é difícil conhecer as características do problema que o levaram a uma determinada dedução. Um exemplo de classificador desse tipo são as Redes Neurais Artificiais (RNs). Os classificadores orientados a conhecimento, como as Árvores de Decisão (ADs) e sistemas *fuzzy*, criam estruturas simbólicas que normalmente são mais compreensíveis do que os classificadores do tipo caixa-preta (Monard e Baranauskas, 2003).

Na Figura 18.1 é ilustrado um diagrama do processo de indução de um classificador e posteriormente a sua utilização. Primeiro o conjunto de treinamento, no qual os rótulos das classes dos exemplos são conhecidos, é utilizado por um algoritmo de aprendizado para construir um modelo. Após a construção, esse classificador pode ser aplicado para predizer os rótulos das classes para exem-

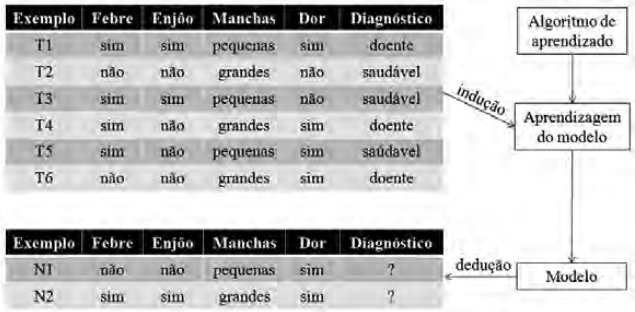


Figura 18.1: Diagrama do processo de indução de um classificador e sua utilização na dedução de novos exemplos.

plos do conjunto de teste.

No contexto de problemas de classificação, os algoritmos genéticos têm tido um papel importante em diversas aplicações. Dentre elas, destacam-se: (i) ajuste de parâmetros dos algoritmos de indução de classificadores; (ii) indução de árvores de decisão; e (iii) decomposição de problemas multiclasses.

O restante deste capítulo está organizado como segue. A Seção 1 discute como os algoritmos genéticos podem ser utilizados para ajustar parâmetros de algoritmos de indução de classificadores. Ao descrever o algoritmo LEGAL-Tree, a Seção 2 mostra como os algoritmos genéticos são facilmente aplicados na indução de modelos de classificação baseados em árvores de decisão. A solução para decomposição de problemas multiclasses por meio de algoritmos genéticos é descrita na Seção 3. Por fim, a Seção 4 apresenta as considerações finais.

1. Ajuste de Parâmetros de Classificadores por AGs

Grande parte dos algoritmos de AM possuem parâmetros cujos valores devem ser especificados pelo usuário. Esses valores para os parâmetros livres, em geral, influenciam diretamente no desempenho de modelos induzidos (Kohavi e John, 1995), o que pode ser entendido como uma deficiência das técnicas de AM. O objetivo do processo de ajuste desses parâmetros pode ser visto como encontrar os melhores valores dos parâmetros livres para um determinado conjunto de dados (Hutter e Hamadi, 2005). O ajuste de parâmetros, cujo intuito é obter melhor desempenho e maior robustez dessas técnicas, é uma tarefa subjetiva e pode consumir muito tempo.

Para definir um conjunto de valores, as atuais técnicas de ajuste normalmente consideram a interação entre o viés (*bias*) do algoritmo de indução (Mitchell, 1982) e o conjunto de treinamento disponível. Técnicas comuns, como a busca exaustiva, são intratáveis quando há mais de dois parâmetros a serem ajustados (Chapelle et al., 2002). Heurísticas podem ser aplicadas com relativo sucesso para uma variedade de conjuntos de dados. Porém, a melhor estratégia é obter valores dos parâmetros que funcionem bem para um conjunto de dados particular (Kohavi e John, 1995). A abordagem mais comum para definir os valores dos parâmetros livres é por tentativa e erro, a qual é altamente subjetiva. Além disso, a busca pelos melhores valores para os parâmetros, geralmente, envolve a otimização por um grande espaço de busca, o que torna esse problema muito custoso computacionalmente.

Por essa razão, técnicas alternativas de otimização têm sido propostas para ajustar de forma automática os parâmetros livres de algoritmos de AM. Algumas dessas técnicas, como os AGs, têm obtido resultados promissores.

Ajuste de Parâmetros de Máquinas de Vetores de Suporte

As Máquinas de Vetores de Suporte (SVMs, do inglês, Support Vector Machines) são baseadas na Teoria do Aprendizado Estatístico (Vapnik e Chervonenkis, 1971), mais especificamente, na Teoria de Minimização do Risco Estrutural (Vapnik, 1995). Os vetores de suporte utilizados pelas SVMs são exemplos que estão próximos da superfície de decisão e, portanto, são os mais difíceis de serem classificados (Haykin, 1999). São esses exemplos que influenciam diretamente na localização da superfície de decisão. Para o problema de classificação, o princípio das SVMs está em encontrar um hiperplano ótimo que separa satisfatoriamente os dados de entrada. O hiperplano ótimo é definido como aquele para o qual a margem de separação entre as classes é maximizada (Haykin, 1999).

O desempenho das SVMs é diretamente influenciado pelo parâmetro *custo* e pela escolha da função *kernel* e dos valores de seus parâmetros (Chapelle et al., 2002). O parâmetro *custo* controla o equilíbrio entre a complexidade do modelo e o número de exemplos não-separáveis (Haykin, 1999). Essa dependência dos valores dos parâmetros não é uma exclusividade das SVMs, e muitos algoritmos de aprendizado dependem de uma escolha adequada para encontrar um bom modelo. O conhecimento sobre o domínio pode auxiliar na escolha da função *kernel* apropriada, reduzindo o problema de seleção de modelo para o ajuste de parâmetros (Cristianini e Shawe-Taylor, 2000).

Para o ajuste de parâmetros de SVMs, métodos de otimização baseados no gradiente são muito utilizados. Porém, segundo (Imbault e Lebart, 2004), esses métodos não resolvem totalmente o problema, a menos que um ponto inicial seja conhecido. Nesse trabalho foi mostrado que o problema de ajuste de parâmetros apresenta mínimo local e foram comparados métodos clássicos de ajuste que utilizam busca local com AGs e *recozimento simulado*, que são métodos de otimização global. Um fator importante dos métodos de otimização global é que estes são desenvolvidos para evitar mínimos locais. Os resultados mostraram que os dois métodos citados obtiveram soluções próximas da ótima de forma mais robusta e eficiente.

Algoritmos Genéticos também foram utilizados em (Lorena e Carvalho, 2006) para ajustar os parâmetros de SVMs multiclasses com *kernel* gaussiano. Nesse estudo foram obtidas maiores taxas de acerto utilizando AGs se comparado aos resultados obtidos utilizando valores fixos para os parâmetros. Esse resultado ocorreu para as quatro bases de dados testadas. Em (Huang e Wang, 2006) e (Souza e Carvalho, 2005) foram utilizados AGs para selecionar características (atributos) de conjuntos de dados e ajustar parâmetros de SVMs simultaneamente. No primeiro caso, foram realizados testes com vários conjuntos de dados e a abordagem baseada em AGs teve boa taxa de acerto se comparada com a técnica *grid search*. No segundo caso, foi utilizado um conjunto de dados de expressão gênica e os resultados obtidos foram equivalentes a outros encontrados na literatura.

Em (Souza et al., 2006), a técnica de otimização por enxame de partículas (PSO) (Kennedy e Eberhart, 1995) foi utilizada para ajustar parâmetros de SVMs multiclasses com *kernel* gaussiano. Foram testadas quatro bases de dados e os resultados obtidos foram comparados com os de outras técnicas de ajuste. Essas técnicas foram denominadas *grid search*, que usa a busca exaustiva, *Naive*, que usa os mesmos valores padrão da biblioteca LIBSVM (Chang e Lin, 2001) para todas as SVMs binárias, e *Global*, que usa um conjunto de validação para estimar o erro de generalização. O PSO conseguiu os menores erros de classificação apenas para uma base, enquanto a técnica *grid search* obteve os melhores resultados para duas bases e a técnica Global para a base restante. Apesar disso, os resultados obtidos pela PSO ficaram próximos aos melhores alcançados e, segundo os autores, não foi possível determinar o melhor método para todas as bases de dados testadas.

Quatro algoritmos bioinspirados, dentre eles um AG, foram usados em (Rossi e Carvalho, 2008) para ajustar os parâmetros de SVMs com *kernel* gaussiano. Os modelos obtidos foram aplicados para classificar quatro bases de dados de análise de expressão gênica e foram comparados com a técnica *grid search* e com os valores padrão da biblioteca LIBSVM. Os resultados obtidos entre todos os algoritmos e técnicas foram semelhantes. Para uma das bases de dados o AG não conseguiu bons resultados, pois ficou preso a mínimos locais. Os valores padrão mostraram-se robustos, conseguindo

melhores resultados para duas bases testadas.

Ajuste de Parâmetros de Redes Neurais

As Redes Neurais Artificiais (RNs) são inspiradas no cérebro e no sistema nervoso e são compostas por unidades de processamento simples, denominados neurônios ou nós, dispostos em uma ou mais camadas e interligados por meio de conexões. Os pesos associados a cada conexão entre os neurônios (sinapse) são responsáveis pelo armazenamento do conhecimento. Os neurônios artificiais são unidades de processamento de informação que realizam um trabalho relativamente simples: recebem entradas de outros neurônios ou do meio externo e usam essas entradas para computar um sinal de saída que é propagado para outras unidades ou para o meio externo. Um algoritmo de aprendizado de RNs deve ser capaz de atribuir pesos a essas conexões durante o processo de treinamento, de maneira que a rede neural seja capaz de classificar corretamente o maior número de exemplos possíveis do conjunto de treinamento e, também, seja capaz de generalizar para novos exemplos.

Há uma grande variedade de arquiteturas e algoritmos de treinamento para RNs e, normalmente, um grande número de parâmetros precisam ser especificados pelo usuário. Em (Basheer e Hajmeer, 2000) os autores afirmam que bons valores para os parâmetros são encontrados, geralmente, por meio de tentativa e erro. Segundo os autores, a escolha de valores para os parâmetros do algoritmo de aprendizado *backpropagation* para RNs influencia na convergência do aprendizado e no desempenho geral da rede.

Algoritmos evolutivos são empregados para ajuste de parâmetros e treinamento de RNs desde o início da década de 90, como pode ser visto em (Miller et al., 1989; Dodd, 1990; Hintz e Spofford, 1990; Braun e Weisbrod, 1993), e ainda são utilizados nos dias atuais. Os trabalhos de (Leung et al., 2003) e (Tsai et al., 2006) propõem modificações nos AGs para a otimização automática e simultânea da topologia (estrutura) das RNs e dos parâmetros do algoritmo de treinamento. Os resultados obtidos pelo AG proposto em (Leung et al., 2003) superaram os obtidos pelo AG padrão, que utilizou cruzamento aritmético e mutação não-uniforme, na otimização de funções de *benchmark*. Posteriormente, duas aplicações foram testadas com as RNs para ilustrar o desempenho dos algoritmos propostos. Em ambas, o AG proposto superou o AG padrão, conseguindo uma rede de menor complexidade (menor número de conexões) e com maior acurácia. Em (Tsai et al., 2006) foi proposto um algoritmo híbrido, que combina AG com o método *Taguchi* (Phadke, 1995), o qual denominaram de HTGA (do inglês, *hybrid Taguchi-Genetic Algorithm*), para ajustar a estrutura e os parâmetros de RNs. Os autores utilizaram as mesmas aplicações testadas em (Leung et al., 2003) e compararam os resultados obtidos. O HTGA foi superior em relação à acurácia, enquanto que a complexidade da rede variou conforme a aplicação.

Um estudo comparativo entre sistemas evolutivos híbridos para geração e otimização da estrutura de RNs de múltiplas camadas foi realizado em (Castillo et al., 2007). Esse estudo usou um método chamado *G-Prop* para otimizar o número de neurônios ocultos e os pesos iniciais das RNs e outro método chamado *ev-QP* para buscar pelos parâmetros de um algoritmo de treinamento. Utilizou-se também um algoritmo co-evolutivo, chamado *co-evolutive*, para tratar dos parâmetros referente à arquitetura, aos pesos iniciais e aos parâmetros do algoritmo de treinamento. Os resultados mostraram que os algoritmos evolutivos apresentaram melhor habilidade de classificação do que o QuickProp (Fahlman, 1988). O primeiro método (GProp) obteve pequenas taxas de erro de classificação, porém o tempo de processamento foi um pouco superior se comparado a outros métodos. O segundo método (ev-QP) teve o menor tempo de processamento, mas produziu as maiores taxas de erro. O método co-evolutivo obteve os menores tempo de processamento e, ao mesmo tempo, melhorou a habilidade de classificação.

Um outro estudo foi realizado em (Rossi et al., 2008) comparando quatro algoritmos bioinspirados, entre eles um AG, para ajuste de parâmetros de RNs de múltiplas camadas. Para essa comparação foram usados quatro bases de dados de análise de expressão gênica. As melhores redes obtidas pelos

algoritmos usando um conjunto de dados de validação foram aplicados para novos dados. O AG apresentou resultados semelhantes aos outros algoritmos bioinspirados em termos de taxa de erro, apesar do menor espaço de busca analisado analisado pelo AG.

Em (Gao et al., 2006), foi proposto um algoritmo PSO modificado, chamado de *SPSO*, para ajustar simultaneamente a estrutura e os pesos das conexões de RNs. Os resultados obtidos foram comparados com o algoritmo *backpropagation* e com um AG desenvolvido para o mesmo propósito. As redes foram aplicadas para o problema de pontuação de crédito, sendo que o SPSO convergiu mais rapidamente e conseguiu maior acurácia do que o algoritmo *backpropagation*. O AG foi o método que obteve as menores taxas de acurácia.

Um Framework para Ajuste de Parâmetros de classificadores por AGs

Para o problema de ajuste de parâmetros de classificadores, os AGs podem ser empregados usando diferentes representações para os cromossomos, diferentes operadores genéticos (segundo a representação escolhida) e diferentes funções de aptidão. Nesta seção são apresentadas as opções comumente usadas na literatura.

Representação do AG

Grande parte das técnicas de classificação possuem parâmetros cujos valores são contínuos. Portanto, é mais natural representar os genes usando números contínuos ao invés da representação binária. O uso do alfabeto binário para representar valores no espaço contínuo pode produzir o efeito denominado penhasco de Hamming (*Hamming cliff*). Isso acontece quando a codificação binária de dois valores adjacentes difere em todos os bits. Por exemplo, os valores 31 e 32 são representados por 011111 e 100000, respectivamente (Herrera et al., 1998).

Com a utilização da codificação real para problemas de domínio contínuo, não há diferença entre a codificação e o espaço de busca (Herrera et al., 1998), pois cada gene representa uma variável do problema. Dessa forma, o tamanho do cromossomo tem o mesmo tamanho do vetor de soluções para o problema, que neste caso é o número de parâmetros a serem ajustados. Se apenas valores inteiros são permitidos para um parâmetro, a solução obtida pelo AG pode ser arredondada para o valor inteiro mais próximo.

Suponha que se queira ajustar o número de neurônios na camada oculta de uma RN, a taxa de aprendizado do algoritmo *backpropagation* e o termo momentum. Para isso, um cromossomo seria representado por três números reais. O primeiro valor refere-se ao número de neurônios na camada oculta (γ), e pode, por exemplo, assumir um valor dentro do intervalo $[2, 100]$. Como esse parâmetro deve ser inteiro, o valor encontrado pelo AG deve ser arredondado para o valor inteiro mais próximo. O segundo valor, refere-se ao parâmetro taxa de aprendizado (η), e pode assumir qualquer valor real, normalmente dentro do intervalo $[0.05, 1]$. O terceiro valor refere-se ao parâmetro termo *momentum* (μ) e pode assumir qualquer valor real, normalmente dentro do intervalo $[0, 1]$. Na Figura 18.2 é ilustrado um exemplo de um indivíduo com $\gamma = 36$, $\eta = 0.1$ e $\mu = 0.8$.

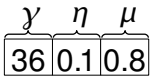


Figura 18.2: Representação de um cromossomo para ajuste de parâmetros de RNs.

Operadores genéticos

Para utilizar um AG com codificação real (AG_{CR}) na solução de problemas, foram desenvolvidos, desde a sua criação, diferentes operadores de cruzamento e mutação. Em (Herrera et al., 1998)

são analisados diferentes operadores para representação contínua. Segundo resultados obtidos neste trabalho, o operador de mutação não-uniforme (Michalewicz, 1992) e os operadores de cruzamento BLX-α (Eshelman e Schaffer, 1993) e *logical* FCB (Herrera et al., 1994) são os mais adequados para serem utilizados com o AG_{CR}. A seguir, o operador de cruzamento BLX-α e o operador de mutação não-uniforme são apresentados. Optou-se aqui por apresentar o BLX-α pela sua simplicidade se comparado ao *logical* FCB.

Sejam $C_1 = (c_1^1, \ldots, c_n^1)$ e $C_2 = (c_1^2, \ldots, c_n^2)$ dois cromossomos selecionados para aplicação do operador de cruzamento. Para o operador BLX-α, um descendente é gerado: $H = (h_1, \ldots, h_i, \ldots, h_n)$, onde h_i é um número aleatoriamente (uniformemente) escolhido no intervalo $[c_{min} - I \cdot \alpha, c_{max} + I \cdot \alpha]$, $c_{min} = minimo(c_i^1, c_i^2)$, $c_{max} = maximo(c_i^1, c_i^2)$, $I = c_{max} - c_{min}$ e α determina o balanceamento entre prospecção (*exploitation*) e exploração (*exploration*), e seu valor deve ser definido pelo usuário. Em (Herrera et al., 1998), diferentes valores para α foram testados e o melhor balanceamento foi alcançado quando $\alpha = 0.5$. A seguir será explicado o operador de mutação não-uniforme.

Seja $C = (c_1, ..., c_i, ..., c_n)$ um cromossomo e $c_i \in [a_i, b_i]$ um gene que foi selecionado aleatoriamente para mutação. A aplicação da mutação não-uniforme sobre esse gene resultará em um gene c_i', cujo valor é definido da seguinte forma. Seja t a geração em que o operador está sendo aplicado e t_{max} o número máximo de gerações. Então

$$c_i' = \begin{cases} c_i + \Delta(t, b_i - c_i) & se \quad \tau = 0 \\ c_i - \Delta(t, c_i - a_i) & se \quad \tau = 1 \end{cases}$$

em que τ é um número aleatório binário (0 ou 1) e

$$\Delta(t, y) = y \left(1 - r^{\left(1 - \frac{t}{t_{max}}\right)^b} \right),$$

sendo r um número aleatório pertencente ao intervalo $[0, 1]$ e b um parâmetro escolhido pelo usuário que determina o grau de dependência do número de gerações. Em (Herrera et al., 1998) é sugerido $b = 5$.

Função de aptidão

A função de aptidão dos AGs para o ajuste de parâmetros de classificadores é o desempenho (taxa de acerto ou erro, por exemplo) dos modelos induzidos para dados de um conjunto de validação separado para esse propósito. O método experimental apresentado a seguir é baseado em dois laços aninhados, evitando que dados de teste sejam usados durante a fase de ajuste de parâmetros.

Dos dois laços aninhados, o laço interno é usado para determinar a melhor combinação de valores para os parâmetros do classificador, ou seja, os valores de parâmetros em que foi obtido o melhor desempenho para o conjunto de dados de validação. O laço externo é usado para estimar o desempenho do classificador gerado com os melhores parâmetros encontrados no laço interno. Os parâmetros são ajustados para cada partição do laço externo, ou seja, a otimização dos classificadores para o conjunto de validação é feita independentemente para cada partição do laço externo. Neste texto usaremos a taxa de erro como exemplo de medida de aptidão para os AGs.

A confiança das estimativas obtidas, tanto para o laço interno como para o laço externo, são afetadas pela natureza aleatória dos exemplos usados para treinamento, validação e teste. Portanto, para reduzir a variância existente, a amostragem por validação cruzada é usada para os dois laços. O número de partições para o laço interno (N_P) pode ser de qualquer tamanho desejado. Geralmente, usa-se o número de partições para o laço externo N_D menos um ($N_P = N_D - 1$), como em (Statnikov, Tsamardinos, Dosbayev e Aliferis, 2005). No laço externo, uma, dentre as N_D partições, é usada como conjunto de teste. No laço interno, uma, dentre as N_P partições, é usada como conjunto de validação.

A Figura 18.3 mostra um exemplo do uso de validação cruzada para os dois laços aninhados, com $N_D = 4$ e $N_P = 3$. O conjunto de dados D, no laço externo, é dividido em quatro partições: $d1$, $d2$, $d3$ e $d4$. As partições de treinamento utilizadas no laço externo são usadas no laço interno para treinamento e teste. A taxa de erro médio obtida no laço interno é a estimativa para a taxa de erro de teste de uma partição do laço externo. No exemplo da Figura 18.3, a taxa de erro médio de validação, que é a aptidão de um cromossomo, é 10.3%. Esta taxa de erro é a estimativa para a partição de teste $d1$, em que a taxa de erro é 10%. A taxa de erro de teste para uma solução (combinação de valores para os parâmetros) para o conjunto de dados D é a média da taxa de erro de teste para cada uma das partições do laço externo. Esse valor é 10.9% no exemplo da figura citada. A taxa de erro de validação para o conjunto de dados D, é a média das N_D taxas de erro médio obtidas no laço interno.

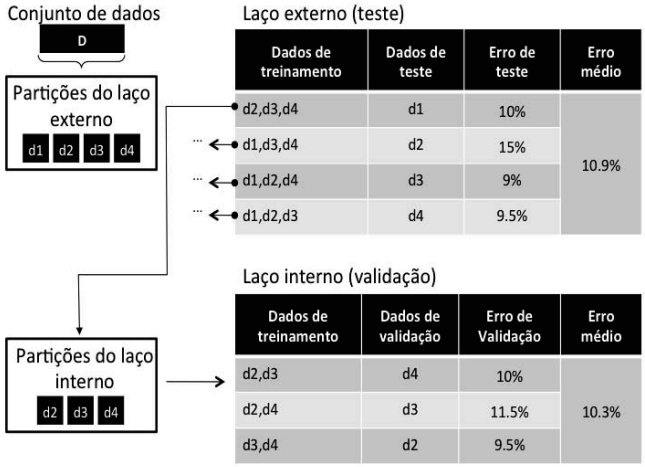


Figura 18.3: Método Experimental B.

A minimização da taxa de erro de validação é realizada pelo AG até que o critério de parada seja satisfeito. Como resposta, o AG fornece a melhor solução (valores para os parâmetros) obtida, ou seja, a solução com a menor taxa de erro (validação) médio obtida no laço interno. Esses valores para os parâmetros são usados para obter a estimativa do erro real (teste), utilizando as partições de teste no laço externo.

Suponha que exista apenas um parâmetro α a ser ajustado para um algoritmo de aprendizado A, e que α possa assumir m diferentes valores: $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_m$. O desempenho D_i de um classificador treinado pelo algoritmo de aprendizado A com o parâmetro α_i é estimado para $i = 1, \ldots, m$ no laço interno. O modelo final é construído treinando o algoritmo A com o parâmetro α_{melhor} no laço externo, em que $melhor = argmax(D_i)$, para $i = 1, 2, \ldots, m$. Para cada combinação de valores dos parâmetros, o algoritmo de aprendizado é executado $N_D \times N_P$ vezes. O Algoritmo 1 é usado para implementar este método experimental.

2. Indução de Árvores de Decisão com Algoritmos Genéticos

As Árvores de Decisão (ADs) são uma técnica muito poderosa e amplamente utilizada em problemas de classificação. Isso pode ser explicado por diversos fatores, dentre eles (Tan et al., 2005): (i) quando não muito grandes, são fáceis de interpretar em virtude da forma de representação do conhecimento adquirido - uma AD é uma representação gráfica e pode ser facilmente convertida em

Algoritmo 1 Método experimental para ajuste de parâmetros.

```
D ← conjunto de dados
l ← 1
enquanto l ≤ N_D faça
  conjunto_treino ← (N_D − 1) partições de D
  conjunto_teste ← partição restante de D
  i ← 1
  enquanto i ≤ m faça
    n ← 1
    enquanto n ≤ N_P faça
      conjunto_treino_validação ← (conjunto_treino − 1) partições
      conjunto_teste_validação ← partição restante de conjunto_treino
      C ← Treinar A com conjunto_treino_validação com parâmetro α_i
      P(n) ← Testar classificador C para o conjunto_teste_validação
      n ← n + 1
    fim enquanto
    D(i) ← média(P)
    i ← i + 1
  fim enquanto
  α_melhor ← argmax(D(i))
  M ← Treinar A com conjunto_treino usando α_melhor
  ρ(l) ← Testar classificador M para conjunto_teste
  l ← l + 1
fim enquanto
retorno média(ρ)
```

regras; (ii) robusta à presença de ruídos; (iii) baixo custo computacional tanto para indução como para utilização das árvores, mesmo para grandes conjuntos de dados; e (iv) lida bem com atributos redundantes e irrelevantes, os quais prejudicam o desempenho dos modelos.

Para ilustrar o funcionamento básico de uma árvore de decisão, pode ser considerado o problema de classificar clientes como bons ou maus pagadores. Supondo que um cliente solicite um crédito no banco, como verificar se o perfil do cliente indica se ele será um bom ou um mau pagador? A primeira pergunta que pode ser feia ao cliente é a sua renda. Se a renda for alta, a chance de ser um bom pagador deve ser um pouco maior. Também é importante saber o estado civil do cliente e a quantidade de filhos, se houver. Se o cliente for casado, também é importante a informação sobre a renda do cônjuge, o que pode ser decisivo na decisão.

O exemplo anterior ilustra uma forma de solucionar um problema de classificação por meio de perguntas sobre uma série de características de um objeto, que no caso é o cliente. A cada pergunta respondida, outra pode se realizada até que se chegue a uma conclusão sobre a classe que pertence o objeto. Essa série de perguntas e suas possíveis respostas podem ser organizadas na forma de uma árvore de decisão, a qual é uma estrutura hierárquica composta por nodos e arestas. A Figura 18.4 ilustra uma possível árvore decisão para o problema dos clientes, os quais podem ser classificados como bom pagador (SIM) ou mau pagador (NÃO). Segundo (Tan et al., 2005), a árvore possui três tipos de nodo:

- um **nodo raiz** que não possui nenhuma aresta de entrada e zero ou mais arestas de saída.
- **nodos internos**, cada qual com exatamente uma aresta de entrada e duas ou mais arestas de saída.

- nodos **folha** ou **terminal**, cada qual com uma única aresta de entrada e nenhuma de saída.

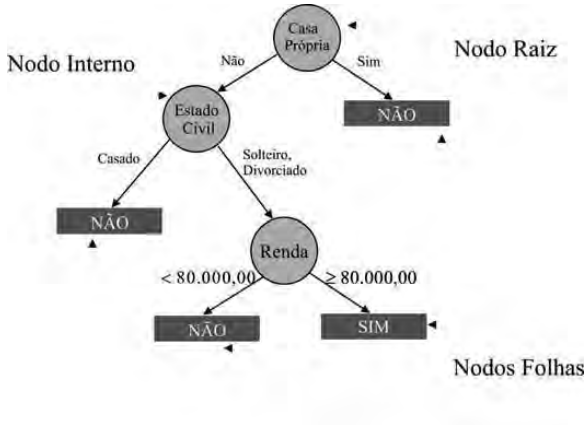


Figura 18.4: Uma árvore de decisão para o problema de classificação de clientes.

Assim, é possível utilizar uma árvore de decisão para classificar um novo cliente como bom ou mau pagador. Para isso, basta partir do nodo raiz da árvore e ir percorrendo-a, através das respostas aos testes dos nodos não-folha, até chegar em um nodo folha, o qual indica a classe correspondente do novo cliente. Além da obtenção da classe, a grande vantagem é que a trajetória percorrida até o nodo folha representa uma regra, facilitando a interpretabilidade do modelo pelo usuário.

Indução de árvore de decisão

Há muitas maneiras de uma árvore de decisão ser estruturada a partir de um conjunto de atributos. De forma exaustiva, o número de árvores de decisão possíveis cresce exponencialmente à medida que aumenta o número de atributos. Logo, fica impraticável encontrar exaustivamente a estrutura da árvore de decisão ótima para um determinado problema devido ao elevado custo computacional envolvido nessa busca. Nesse sentido, algoritmos baseados em heurísticas têm sido desenvolvidos para a indução de árvores de decisão. Mesmo que eles não garantam a solução ótima, apresentam resultados satisfatórios em tempo factível.

Geralmente, algoritmos de indução de AD usam estratégia gulosa, *top down* e com particionamento recursivo para a construção da árvore. No entanto, há pelo menos dois problemas relacionados a essas características: (i) estratégias gulosas geralmente produzem soluções ótimas locais ao invés de globais, (ii) particionamento recursivo degrada iterativamente a qualidade do *dataset*, pois quanto maior o número de vezes que o *dataset* é particionado, menores são os sub-conjuntos resultantes dessas partições e, conseqüentemente, tornam estatisticamente não significativos os resultados obtidos por quaisquer critérios de manipulação de dados (ex: cálculo de entropia). Além disso, o número de exemplos pertencentes a um nodo folha da árvore pode ser tão pequeno que torna também não estatisticamente significativa a decisão sobre a classe representada por esse nodo folha, contribuindo para o super-aprendizado (*overfitting*) do conjunto de treino (Tan et al., 2005).

Para lidar com essas dificuldades, diferentes abordagens foram sugeridas, mas que também apresentam problemas. Tais abordagens podem ser divididas em duas tendências básicas: (i) divisões múltiplas nos nodos internos; e (ii) geração de múltiplas árvores, combinando diferentes visões sobre o mesmo domínio. Abordagens baseadas em (i) resultam nas chamadas *Option Trees* (Buntine, 1993). *Option Trees* são difíceis de interpretar, comprometendo o que pode ser considerada a principal virtude das árvores de decisão. As abordagens baseadas em (ii) são mais interessantes, pois podem

fornecer uma única classificação a partir de diferentes árvores de decisão de acordo com um dado critério (ex: a classe resultante pela maioria das árvores). Uma vez que essa abordagem pode gerar melhores resultados quando comparada aos algoritmos que induzem uma única AD, em geral não resolve o problema de convergir para um ótimo local. Além disso, também degrada a interpretabilidade do modelo resultante, pois é composto por várias árvores de decisão, as quais podem ser totalmente diferentes umas das outras.

Métodos *ensemble* como *random forests*, *boosting* e *bagging* são as mais conhecidas abordagens baseadas em (ii). *Random forests* (Nadeau e Bengio, 2003) criam um conjunto inicial de árvores baseadas nos valores do conjunto independentes de vetores aleatórios, gerados por uma distribuição de probabilidades de um conjunto de treino. Um esquema de voto majoritário é usado para combinar os resultados das árvores. Algoritmos *boosting* (Quinlan, 1996) criam um conjunto inicial de árvores ao adaptativamente alterar a distribuição do conjunto de treino. Essas árvores são então combinadas considerando um sistema de votação ponderada. *Bagging* (Quinlan, 1996) quando aplicado em árvores de decisão pode ser considerado um caso particular de *random forests*. As árvores também são combinadas por uma regra de votação majoritária.

Todos esses métodos Ensemble têm um problema em comum, que é o esquema de combinar as árvores geradas. É de conhecimento geral que um *ensemble* de classificadores melhora a acurácia preditiva quando comparada a único classificador. Em contra partida, o uso de *ensembles* também tende a reduzir a interpretabilidade do modelo preditivo quando comparado a um único classificador. Um único modelo preditivo compreensível pode ser interpretado por um especialista, porém na prática é bastante complicado para um especialista interpretar uma combinação de modelos compreensíveis. Além de ser tediosa e demorada tal interpretação para o especialista, há também um problema fundamental: os modelos de classificação que compõem um *ensemble* geralmente estão sujeitos à inconsistências, as quais são necessárias para aumentar a acurácia preditiva do *ensemble*. Uma vez que cada modelo pode ser considerado como uma hipótese para explicar os padrões preditivos, isso significa que um *ensemble* não representa uma única hipótese coerente sobre os dados, mas sim um conjunto grande de hipóteses inconsistentes, as quais em geral podem confundir o especialista do domínio (Freitas et al., 2008).

Técnicas de algoritmos evolutivos também já foram exploradas no contexto de indução de árvores de decisão. Trabalhos como (Cantú-Paz e Kamath, 2000) e (Kretowski, 2004) usam tais técnicas para indução de Árvores de Decisão Oblíquas. Esse tipo de árvore, entretanto, difere das árvores de decisão tradicionais por considerar, em cada nodo interno, combinação linear de atributos, ao invés de um único atributo, para particionar o conjunto de exemplos. Como é de conhecimento geral, encontrar a melhor árvore de decisão oblíqua é um problema NP-completo, motivando os autores de (Cantú-Paz e Kamath, 2000) e (Kretowski, 2004) a proporem técnicas evolutivas para selecionar a melhor combinação de atributos em cada divisão, evitando a estratégia gulosa.

Uma das sub-áreas de AE, Programação Genética tem sido amplamente utilizada como método de indução de árvores de decisão, em trabalhos como (Koza, 1991; Zhao, 2007; Estrada-Gil et al., 2007; Bot e Langdon, 2000; Loveard e Ciesielski, 2001, 2002; Shirasaka et al., 1998; Zhao e Shirasaka, 1999; Tür e Güvenir, 1996). O trabalho de Koza (Koza, 1991) foi o pioneiro na indução de AD com PG, convertendo os atributos do problema das condições climáticas (Quinlan, 1986) em funções descritas em expressões S em LISP. Em (Zhao, 2007), é proposta uma ferramenta que permite ao usuário configurar diferentes parâmetros para gerar o melhor programa de computador para induzir árvores de classificação. Os autores de (Zhao, 2007) consideram o custo dos erros de classificação em uma abordagem multi-objetiva para definir uma árvore ótima. Em (Estrada-Gil et al., 2007), foi implementado um algoritmo de indução de árvores no contexto de bioinformática para detectar interações em variantes genéticas. Bot e Langdon (Bot e Langdon, 2000) propuseram uma solução para indução de árvores de classificação usando PG, em que um nodo intermediário é uma combinação linear de atributos. Em (Loveard e Ciesielski, 2001), os autores propuseram diferentes alternativas

para representar um problema de classificação multi-classes através de PG, e após estenderam seu trabalho (Loveard e Ciesielski, 2002) para lidar com atributos nominais. Em (Shirasaka et al., 1998) e (Zhao e Shirasaka, 1999), os autores propuseram o projeto de árvores de classificação binária usando PG, e em (Tür e Güvenir, 1996) é descrito um algoritmo de PG para indução de árvores considerando somente atributos binários.

Neste pronto, é importante discutir uma questão de terminologia. Em PG cada indivíduo da população é um programa de computador (contendo dados e operadores/funções aplicadas aos dados), e esse indivíduo geralmente é representado como uma estrutura de árvore. Idealmente, um programa de computador deveria ser genérico suficiente para processar qualquer instância do problema alvo (no nosso caso, o programa evoluído deveria ser hábil para induzir árvores de decisão para qualquer dataset de classificação para qualquer domínio de aplicação). Pode ser notado, entretanto, que nos algoritmos de PG citados cada indivíduo é uma árvore de decisão para um *dataset* a ser minerado, e não um programa de computador genérico responsável por induzir árvores de decisão para qualquer *dataset*. Uma vez que os trabalhos mencionados nomeiam seus respectivos algoritmos como PG, na realidade são AGs, pois enfatiza o fato de evoluir árvores de decisão (as quais não são programas de computador), isto é, a solução não é um algoritmo de indução (como o C4.5) mas sim a própria árvore de decisão.

Considerado os AGs, não há muitos trabalhos relacionados que os utilizam para indução de árvores de decisão. Em (Carvalho e Freitas, 2004), os autores combinam árvores de decisão e AG para lidar com problemas de pequenos disjuntos (basicamente, regras que cobrem poucos exemplos) em problemas de classificação. Essa abordagem busca incrementar a acurácia de um algoritmo *top-down* tradicional ($C4.5$) ao usar um AG para descobrir regras de pequenos disjuntos, uma vez que grandes disjuntos são classificados com sucesso pelo $C4.5$. Tal abordagem apresentou resultados interessantes com um ganho significativo de acurácia, porém mais custoso computacionalmente. Esse trabalho não tem por objetivo gerar uma árvore ótima, mas sim gerar uma árvore usando o $C4.5$ e então substituindo algumas regras extraídas das árvores (as regras de pequenos disjuntos) por regras evoluídas pelo AG.

A indução de árvores de decisão por meio de AG é apresentado em (Kretowski e Grzes, 2005) e (Z. Bandar e McLean, 1999). Em (Kretowski e Grzes, 2005), a função de *fitness* seleciona qual indivíduo tem a maior probabilidade de sobreviver para a próxima geração por dois conceitos: Q_{RCLass}, qualidade de classificação estimada no conjunto de treino, e tamanho da árvore, a qual é controlada por um parâmetro indicado pelo usuário. Embora essa abordagem seja considerada muito interessante, não está claro no artigo como os autores calculam a medida Q_{RClass}, nem como os autores aplicaram essa medida para comparar seu algoritmo com o $C4.5$. Em (Z. Bandar e McLean, 1999) cada árvore de decisão tem um número fixo de nodos que são representados como um vetor estático, e somente divisões binárias dos nodos são permitidas. Essa abordagem impõe a restrição de que atributos nominais com mais de dois valores possíveis devem ter seus conjuntos de valores reduzidos para dois via um agrupamento de valores, o qual é um processo não natural e pode degradar a acurácia em alguns casos. Embora o número de divisões igual a dois possa induzir o leitor a pensar que as árvores resultantes são mais simples, isso não é verdade pois as divisões binárias geram árvores com mais níveis de profundidade que, conseqüentemente, resultarão em regras mais difíceis de serem interpretadas.

Como alternativa aos algoritmos apresentados anteriormente, Basgalupp et al. propuseram em (Basgalupp, Barros, de Carvalho, Freitas e Ruiz, 2009) e (Basgalupp, de Carvalho, Barros, Ruiz e Freitas, 2009) o algoritmo LEGAL-Tree (LExicographic Genetic ALgorithm for decision Tree induction), um algoritmo genético baseado no critério multi-objetivo lexicográfico para indução de árvores de decisão precisas e interpretáveis. Esses trabalhos são pioneiros na utilização da avaliação lexicográfica para indução de árvores de decisão, considerando não somente a acurácia preditiva das árvores mas também a interpretabilidade das mesmas, a qual é medida pelo número de nodos da árvore. Os resultados obtidos foram satisfatórios, superando o algoritmo C4.5, bastante conhecido na literatura. A seguir, é feita uma descrição sobre cada detalhe de implementação do algoritmo LEGAL-Tree.

LEGAL-Tree

Representação da solução

Na grande maioria dos trabalhos que utilizam algoritmos genéticos, cada indivíduo é representado por *strings* binários ou *strings* numéricos. Em LEGAL-Tree, cada indivíduo é uma própria árvore de decisão, que é uma representação bastante lógica e intuitiva. Cada indivíduo é, portanto, um conjunto de nodos, que podem ser terminais (folhas) ou não-terminais (nodos de decisão). Cada nodo não-terminal representa um teste sobre um atributo preditivo e cada folha é rotulada com um valor do atributo classe. Um conjunto de arestas ligando cada nodo não-terminal com seus respectivos filhos também faz parte da representação da árvore. Há duas situações possíveis de relacionamento entre nodos:

- Relacionamento entre um nodo categórico e seus filhos: se um nodo x representa um atributo categórico, o nodo irá conter n arestas, em que n é o número total de categorias (valores) para o atributo;
- Relacionamento entre um nodo numérico e seus filhos: se um nodo x representa um atributo numérico, haverá uma divisão binária de acordo com um valor de *threshold*, o qual é determinado automaticamente pelo algoritmo.

Gerando a população inicial

Mesmo que construção totalmente aleatória de árvores seja a técnica mais comum para gerar populações iniciais em aplicações de AG, seria bem mais interessante incorporar conhecimento da tarefa de classificação para induzir as árvores de decisão. Assim, utilizando conhecimento sobre o significado das árvores de decisão no contexto da tarefa de classificação, espera-se derivar melhores soluções, ou pelo menos soluções tão boas quanto, em menos gerações. Nesse contexto, optou-se por utilizar uma estratégia de incorporar conhecimento específico no AG, a qual pode ser definida no pseudo-código ilustrado em Algoritmo 2.

Algoritmo 2 Geração dos *Decision Stumps*

```
dado x como o conjunto de treino
divida x em 10 partes diferentes
dado dsList como a lista de decision stumps
dado y_j representando a j^th parte de x
para y_i = 1 to 10 faça
  dado a_i como o i^th atributo do conjunto de treino
  para a_i = 1 até numeroDeAtributos faça
    se a_i é numérico então
      threshold = infoGain(a_i)
      dsList.add(new numericDS(a_i,threshold))
    senão
      dsList.add(new categoricalDS(a_i))
    fim se
  fim para
fim para
```

Um *decision stump* (DS) é o caso mais simples de árvores de decisão, o qual consiste de um único nodo de decisão (não-terminal) e duas folhas (Freund e Mason, 1999). Em LEGAL-Tree, esse conceito foi estendido para suportar atributos categóricos, em que cada aresta que representa uma categoria

será ligada a um nodo folha. Assim, poderá haver mais de duas folhas caso o nodo de decisão represente um atributo que possui mais de duas categorias possíveis. O pseudo-código ilustrado no Algoritmo 2 irá basicamente gerar um conjunto de *decision stumps* (na verdade, 10 X n *decision stumps*, em que n é o número de atributos preditivos do conjunto de treino). Ao dividir o conjunto de treino em 10 partições disjuntas, espera-se atingir um certo nível de heterogeneidade para os *decision stumps*, pois eles serão essenciais para a geração da população inicial, a qual deve ser mais diversificada possível. Tal processo de geração de *decision stumps* não deixa de ser aleatório, porém evita geração de soluções de baixa qualidade e não factíveis, que podem atrasar o processo de convergência. Assim, espera-se convergir para boas árvores de decisão em um menor número de gerações que em um processo de geração totalmente aleatório.

O passo final da geração da população inicial é combinar os diferentes *decision stumps* que foram criados. O usuário configura uma profundidade máxima para as árvores que formarão a população inicial, e o algoritmo seleciona os *decision stumps* aleatoriamente e então constrói as árvores de acordo com a profundidade também gerada aleatoriamente, variando de 1 até o valor máximo informado pelo usuário. A Figura 18.5 ilustra esse raciocínio, em que três *decision stumps* (A, B e C) são selecionados e usados para construir uma árvore completa de profundidade 2 (dois). Para o nodo raiz da árvore, foi selecionado o *decision stump* A, cujos filhos foram substituídos pelos *decision stumps* B e C, respectivamente. Observa-se que todas as árvores geradas na população inicial são árvores completas. Entretanto, as árvores que irão formar as próximas gerações da população não serão necessariamente completas em virtude da aplicação dos operadores genéticos, os quais afetarão suas estruturas.

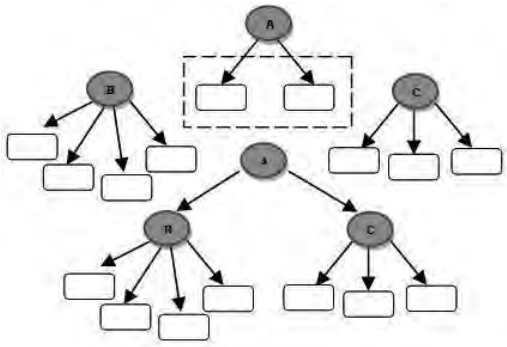


Figura 18.5: Combinando três decision stumps (A, B e C) para construção de um indivíduo de profundidade 2.

As duas principais vantagens desta abordagem para a geração da população inicial de árvores de decisão são que: (a) ao invés de aleatória, a escolha dos *thresholds* para atributos numéricos em nodos não-terminais é baseada no ganho de informação (Tan et al., 2005), utilizando conhecimento do conjunto de dados; e (b) a classe associada ao nodo folha é sempre a mais freqüente nos exemplos cobertos pelo nodo ao invés de uma classe escolhida aleatoriamente. Ambas as vantagens são resultado da utilização do conhecimento geral sobre a tarefa de classificação (ao invés de um conhecimento específico do domínio de aplicação) no AG, o qual tende a melhorar sua efetividade, como discutido previamente.

Tratamento de valores desconhecidos

Foi escolhida a seguinte estratégia para tratamento de valores desconhecidos:

- Atributos categóricos: se o nodo a ser analisado representa um atributo categórico, os valores ausentes são substituídos pela moda de todos os valores do atributo no conjunto de treino;

- Atributos numéricos: se o nodo a ser analisado representa um atributo numérico, os valores ausentes são substituídos pela média aritmética de todos os valores do atributo no conjunto de treino.

Função de *fitness* multi-objetiva baseada na abordagem lexicográfica

Uma questão crucial em mineração de dados é como avaliar a qualidade de um modelo candidato (Freitas, 2004). Considerando LEGAL-Tree, em que cada indivíduo é um candidato a modelo de classificação (árvore de decisão), a função de *fitness* é vital para determinar os melhores indivíduos de forma a evoluir a população e convergir para soluções próximas à(s) ótima(s).

É também de senso comum que em muitos domínios de aplicação o conhecimento descoberto por um algoritmo de mineração não deve ter somente uma boa acurácia (taxa de acerto) mas também deve ser compreensível pelo usuário, em que compreensão (interpretabilidade) é normalmente estimada pelo tamanho do classificador - classificadores menores são preferíveis, uma vez que outros critérios permaneçam iguais. Com isso, em LEGAL-Tree é proposta uma função de *fitness* multi-objetiva para cada indivíduo, isto é, considera mais que uma medida para avaliar a sua qualidade.

Nesse contexto, é apresentada a abordagem lexicográfica, que consiste basicamente em estabelecer diferentes prioridades a diferentes objetivos (medidas) e então focar na otimização desses objetivos de acordo com as respectivas ordens de prioridade (Freitas, 2004). Até o momento, LEGAL-Tree considera somente a acurácia e tamanho da árvore de decisão como objetivos, uma vez que são as medidas mais utilizadas para avaliação de árvores de decisão. Entretanto, esse conjunto de objetivos pode ser facilmente estendido para um número n de objetivos de acordo com o domínio de aplicação a ser considerado, tais como profundidade da árvore, custo dos atributos utilizados, desempenho, etc.

A abordagem convencional de fórmula ponderada sofre de diversos problemas, tais como o problema dos "números mágicos"(configurar os pesos na fórmula é um procedimento *ad-hoc*), misturar maças com laranjas (misturar critérios não comparáveis como acurácia e tamanho da árvore) e misturar diferentes unidades de medidas (realizar operações com diferentes escalas e induzir *bias* quando escolher um procedimento de normalização) (Freitas, 2004). A abordagem de Pareto também tem seus problemas, e o principal deles é a dificuldade de escolher a melhor solução a ser utilizada na prática, pois o resultado dessa abordagem não é apenas uma árvore e sim um conjunto de árvores não-dominadas. Uma alternativa seria combinar as árvores encontradas, porém seria considerado o caso de um *ensemble*, que apresentam problemas indesejáveis no contexto deste trabalho. Outro problema da abordagem de Pareto é a dificuldade de lidar com diferentes níveis de prioridade, isto é, quando um objetivo é significativamente mais importante que um outro. Em árvores de decisão, a acurácia é consideravelmente mais importante que tamanho da árvore, mas a abordagem de Pareto assume que ambos os objetivos são igualmente relevantes. A abordagem lexicográfica não sofre dos problemas mencionados, é conceitualmente simples, fácil de usar, implementar e estender para objetivos adicionais.

Seleção

Para selecionar os indivíduos que formação as próximas gerações da população, LEGAL-Tree usa o método do torneio, um método popular e bastante efetivo. É também implementada a seleção por elitismo, a qual mantém os melhores indivíduos nas próximas gerações. O número indivíduos que fazem parte da elite é um parâmetro que pode ser definido pelo usuário.

Cruzamento

LEGAL-Tree implementa a operação de cruzamento como segue. De acordo com um número gerado aleatoriamente entre 1 (nodo raiz) e n (número total de nodos da árvore), LEGAL-Tree realiza uma busca pré-fixada, visitando recursivamente o nodo raiz e então seus filhos da esquerda para a direita. Para nodos numéricos, o método de busca é equivalente à tradicional busca binária pré-fixada. Nos caso em que o nodo é categórico e possui mais de 2 filhos, o método de busca visita cada filho da esquerda para a direita, de acordo com um índice que identifica cada nodo filho. Após identificar os nodos que representam o número sorteado para ambas as as árvores, LEGAL-Tree troca as respectivas sub-árvores representadas pelos nodos sorteados, os quais representam os nodos raíz de cada sub-árvore.

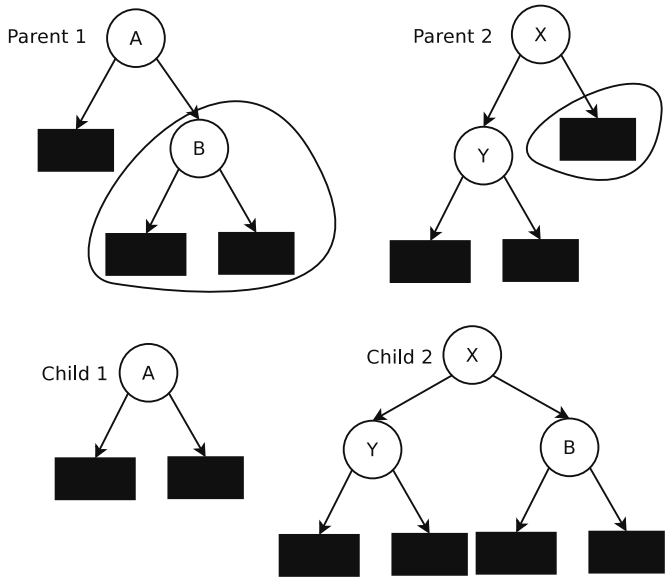


Figura 18.6: Cruzamento entre dois indivíduos, $Parent1$ e $Parent2$, formando dois novos indivíduos, $Child1$ e $Child2$.

Considere os dois indivíduos apresentados como "*Parents*"na Figura 18.6. Para $Parent1$, o nodo sorteado foi o C, enquanto que para o $Parent2$ foi sorteado o nodo M. Após identificar esses dois nodos pelo método de busca supracitado, a operação cruzamento irá criar dois indivíduos filhos ($Child1$ e $Child2$) que serão resultados da estrutura de um dos pais com uma parte do outro. $Child1$ mantém a estrutura de $Parent1$, mas herda o nodo M (e tudo o que estiver abaixo) de $Parent2$. Similarmente, $Child2$ mantém a estrutura de $Parent2$ e herda o nodo C de $Parent1$.

Ao trocar sub-árvores inteiras dos nodos sorteados ao invés de nodos específicos, espera-se evitar problemas como irregularidade de domínios, pois cada aresta representa um possível valor para o atributo específico de um nodo. Entretanto, a operação de cruzamento não evita a possibilidade de regras redundantes (mesmo atributo categórico representado por um nodo x sendo utilizado em outro nodo das sub-árvores de x) e inconsistências (*thresholds* que resultam em intervalos irregulares ao utilizar nodos numéricos). A Seção "Factibilidade das soluções"mostra detalhes de como LEGAL-Tree trata essas questões.

Mutação

LEGAL-Tree implementa duas estratégias diferentes para mutação dos indivíduos. A primeira consiste em trocar uma sub-árvore qualquer de um indivíduo por um nodo folha representando a classe mais freqüente para os exemplos cobertos por essa folha. A segunda estratégia utiliza os *decision stumps*, os quais foram criados durante a geração da população inicial. Essa estratégia substitui um nodo folha, também selecionado aleatoriamente do indivíduo, por um *decision stump* qualquer. A Figura 18.7 ilustra ambas as estratégias apresentas, as quais são denominadas $Mutation1$ e $Mutation2$, respectivamente.

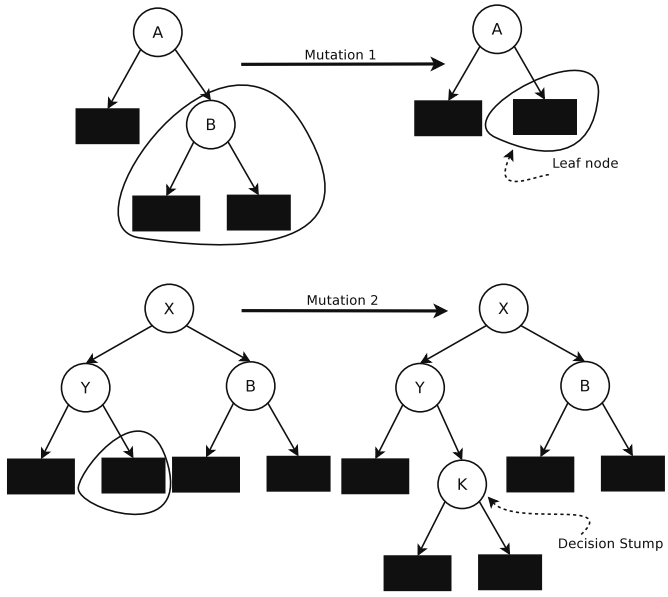


Figura 18.7: Duas estratégias, $Mutation1$ e $Mutation2$, para mutação dos indivíduos.

Tais estratégias podem aumentar ou diminuir o tamanho dos indivíduos, aumentando a diversidade da população e evitando convergências prematuras para ótimos locais. A taxa de mutação é um parâmetro configurado pelo usuário e geralmente possui um valor baixo.

Factibilidade das soluções

Após as operações de cruzamento e mutação e antes do ciclo do AG se repetir para as próximas gerações, há casos em que são gerados cenários inconsistentes. Por exemplo, considere que uma sub-árvore de uma árvore a foi substituída por uma sub-árvore de uma árvore b gerando um indivíduo filho durante o processo de cruzamento. Se a nova sub-árvore tem um nodo que representa um atributo já utilizado por um nodo ancestral, ações devem ser tomadas de modo a evitar regras redundantes ou *thresholds* que geram intervalos inconsistentes. Se o nodo é categórico e um ancestral representa o mesmo atributo, então caracteriza-se um caso de redundância que deve ser eliminado. Nesse caso, LEGAL-Tree substitui toda a sub-árvore por um nodo folha representando a classe mais freqüente. Por outro lado, se o nodo é numérico e seu *threshold* é inconsistente com algum nodo ancestral, então o *threshold* é ajustado para evitar caminhos que não contemplem nenhum exemplo.

Esses problemas são tratados por LEGAL-Tree por meio de filtros aplicados após as operações de cruzamento e mutação. Esses filtros são similares aos utilizados na geração da população inicial,

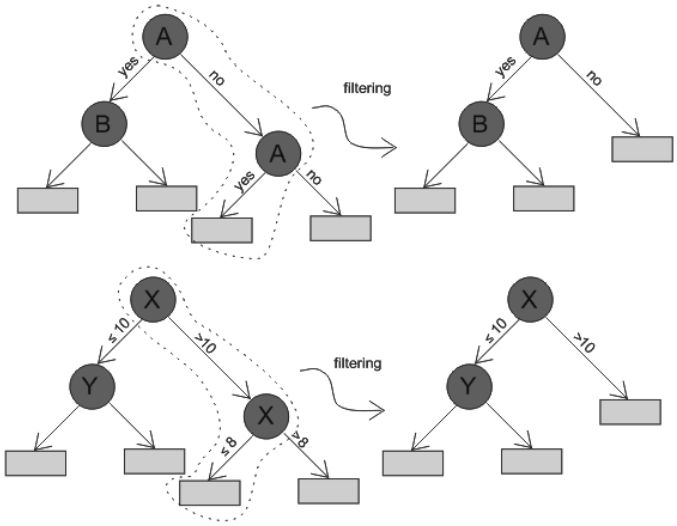


Figura 18.8: Dois tipos de inconsistência originados por repetição de atributos categórico (A) e numérico (X).

quando são evitados esses problemas durante a combinação dos *decision stumps* para a geração das árvores de decisão. A Figura 18.8 ilustra os dois casos de inconsistência supracitados.

Outro problema que pode ocorrer após as operações de cruzamento e mutação é quando um nodo folha deixa de representar a classe mais freqüente para os exemplos cobertos por ele. Essa inconveniência também é tratada por LEGAL-Tree pelos mesmos filtros, que recalculam o valor da classe mais freqüente para as folhas modificadas. O tratamento dessas questões foi considerado um grande passo na construção de LEGAL-Tree, aumentando a velocidade de convergência para soluções próximas à(s) ótima(s).

3. Decomposição de Problemas Multiclasses

Um problema de classificação com apenas duas classes é conhecido como problema de classificação binário. Um exemplo de um problema de classificação binário é o diagnóstico médico de uma doença em particular. Neste problema, o classificador induzido usa dados clínicos de um paciente para determinar se ele possui uma determinada doença. As classes representam a presença ou ausência da doença. Vários problemas reais, porém, envolvem a discriminação de mais que duas categorias ou classes. Exemplos incluem o reconhecimento de dígitos manuscritos (Knerr et al., 1992; LeCun et al., 1995), a distinção de múltiplos tipos de câncer (Statnikov, Aliferis, Tsamardinos, Hardin e Levy, 2005) e a categorização de textos (Berger, 1999; Ghani, 2000).

Várias técnicas de AM foram originalmente formuladas para a solução de problemas de classificação binários. Entre elas, pode-se mencionar SVMs (Cristianini e Shawe-Taylor, 2000; Lorena e Carvalho, 2008) e o algoritmo RIPPER (Cohen, 1995). Duas abordagens têm sido adotadas na literatura para generalizar essas técnicas a problemas multiclasses: (a) adaptação das operações internas de seus algoritmos de aprendizado; (b) combinação de classificadores binários por meio de uma decomposição do problema multiclasses em uma série de problemas de classificação binários.

A extensão de um algoritmo de aprendizado binário a uma versão multiclasses pode ser difícil

de ser realizada para algumas técnicas (Passerini et al., 2004). Para SVMs, em particular, Hsu and Lin (Hsu e Lin, 2002) observaram que a reformulação dessa técnica em versões multiclasses leva a algoritmos computacionalmente custosos. É comum recorrer-se então à alternativa de decompor o problema multiclasses em subproblemas binários, uma estratégia denominada decomposicional.

Segundo Moreira e Mayoraz (Moreira e Mayoraz, 1998), existe uma série de motivações para empregar estratégias decomposicionais na solução de problemas multiclasses. Além de haver técnicas cuja formulação é originalmente binária, alguns algoritmos não são adequados a problemas com um número elevado de classes ou apresentam dificuldades em lidar com grandes volumes de dados de treinamento. Mesmo que o algoritmo seja capaz de trabalhar com problemas de grande escala, o uso de um procedimento decomposicional pode reduzir a complexidade computacional envolvida na solução do problema total, por meio da divisão deste em subtarefas mais simples.

Pimenta (Pimenta, 2005) também aponta vantagens no uso da decomposição em problemas cujos erros de classificação das diferentes classes possuem pesos distintos. Pode-se assim gerar preditores binários de maneira a impor preferências para algumas classes. O autor menciona ainda o fato de que os subproblemas de classificação gerados em uma decomposição são independentes, podendo ser solucionados paralelamente.

O emprego de técnicas decomposicionais envolve dois passos. No primeiro, realiza-se a divisão do problema multiclasses em subproblemas binários, determinando assim os classificadores binários a serem gerados. A segunda etapa pode ser denominada agregação ou reconstrução e refere-se à forma como as saídas dos classificadores binários são combinadas na determinação da classe de um exemplo (Mayoraz e Moreira, 1997).

O *Framework* de Matrizes de Códigos

Várias estratégias podem ser empregadas na decomposição de um problema multiclasses em uma série de problemas de classificação binários. Essas decomposições podem ser genericamente representadas por meio do *framework* de matrizes de códigos descrito em (Allwein et al., 2000), em que as decomposições são representadas por uma matriz de códigos $\mathbf{M}$. As linhas dessa matriz contêm códigos que são atribuídos a cada classe. As colunas de $\mathbf{M}$ definem partições binárias das k classes e correspondem aos rótulos que essas classes assumem na geração dos classificadores binários. Tem-se então uma matriz de dimensão kxl, em que k é o número de classes do problema e l representa o número de classificadores binários utilizados na solução multiclasses. Na Figura 18.9 é apresentado um exemplo de matriz de códigos na qual o número de classes k é igual a 4 (quatro) e o número de classificadores binários l também é 4 (quatro). Abaixo dessa matriz de códigos são indicadas as partições binárias das classes realizadas por cada classificador binário representado em suas colunas.

Cada elemento da matriz $\mathbf{M}$ assume valores em $\{-1, 0, +1\}$. Um elemento m_{ij} com valor $+1$ indica que a classe correspondente à linha i assume rótulo positivo na indução do classificador f_j. O valor -1 designa um rótulo negativo e um valor 0 indica que os dados da classe i não participam do processo de indução do classificador f_j. Classificadores binários são então treinados de forma a aprender os rótulos representados nas colunas de $\mathbf{M}$.

Um novo dado $\mathbf{x}$ pode ser classificado avaliando-se as predições dos l classificadores, que geram um vetor $\mathbf{f}(\mathbf{x})$ de tamanho l na forma $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_l(\mathbf{x}))$. Esse vetor é então comparado às linhas de $\mathbf{M}$. O dado é atribuído à classe cuja linha de $\mathbf{M}$ é mais próxima de $\mathbf{f}(\mathbf{x})$ de acordo com alguma medida de distância. Esse processo de reconstrução também é referenciado como decodificação em alguns trabalhos (Passerini et al., 2004).

O número total de diferentes preditores binários para um problema com k classes é $0{,}5\big(3^k + 1\big) - 2^k$, considerando que $f = -f$, ou seja, que a inversão das classes positivas e negativas produz o mesmo classificador (Mayoraz e Moreira, 1997). Desses, $2^{k-1} - 1$ incluem todas as classes simultaneamente, ou seja, possuem somente rótulos $+1$ e -1, sem o elemento 0 (Mayoraz e Moreira, 1997). Para quatro

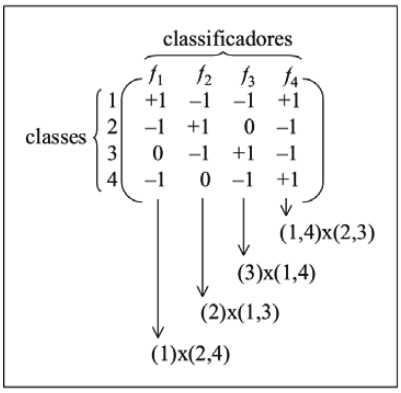


Figura 18.9: Exemplo de matriz de códigos para um problema com quatro classes

classes, tem-se nesse caso a matriz representada na Figura 18.10b. Por sua vez, a decomposição mais compacta de um problema com k classes pode ser realizada com o uso de $l = \lceil \log_2(k) \rceil$ classificadores binários (Mayoraz e Moreira, 1997). Um exemplo de matriz compacta para um problema com quatro classes é apresentado na Figura 18.10a.

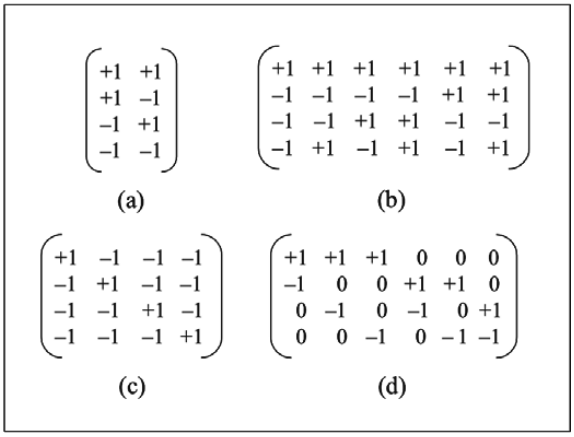


Figura 18.10: Matrizes de diferentes decomposições para um problema com quatro classes

Entre as estratégias decomposicionais mais comuns encontradas na literatura estão a um-contra-todos (*one-against-all* - OAA), a todos-contra-todos (*all-against-all*, também denominada *one-against-one* - OAO) (Hastie e Tibshirani, 1998) e a baseada em códigos de correção de erros (Dietterich e Bariki, 1995), que são descritas a seguir.

Na estratégia OAA, dado um problema com k classes, k classificadores binários $f_i(\mathbf{x})$ são gerados. Cada um deles é treinado de forma a distinguir uma classe i das demais. A representação dessa técnica é dada por uma matriz de dimensão kxk, na qual os elementos da diagonal possuem o valor $+1$ e os demais, o valor -1. Uma matriz do tipo OAA para um problema de quatro classes é apresentada na Figura 18.10c. Usualmente, na integração dos classificadores OAA, escolhe-se a classe correspondente ao classificador que produz a maior saída (Rifkin e Klautau, 2004).

A decomposição OAA pode apresentar desvantagens quando a proporção de exemplos de uma classe é muito pequena em relação a do conjunto formado pelos dados das outras classes. Esse tipo

de desbalanceamento pode dificultar a indução de um preditor que apresente bom desempenho no reconhecimento da classe considerada.

Na decomposição OAO, dadas k classes, $\frac{k(k-1)}{2}$ classificadores binários são gerados. Cada um deles é responsável por diferenciar um par de classes (i, j), em que $i \neq j$. A matriz de códigos nesse caso possui então dimensão $k\text{x}\frac{k(k-1)}{2}$ e cada coluna corresponde a um classificador binário para um par de classes. Em uma coluna representando o par (i, j), o valor do elemento correspondente à linha i é $+1$ e o valor do membro correspondente a j é igual a -1. Todos os outros elementos da coluna possuem o valor 0, indicando que os dados de outras classes não participam do processo de indução do classificador. Na Figura 18.10d tem-se uma matriz OAO para um problema com quatro classes.

Embora o número de classificadores gerados na decomposição OAO seja da ordem de k^2, o treinamento de cada um deles envolve dados de apenas duas classes. Com isso, mesmo com um número elevado de classes, o tempo total despendido na geração dos preditores geralmente não é grande.

A integração usual dos classificadores gerados na estratégia OAO é realizada por meio de um esquema de votação por maioria (Hastie e Tibshirani, 1998). Dado um novo exemplo $\mathbf{x}$, cada classificador fornece um voto em sua classe preferida. O resultado é então dado pela classe que recebeu mais votos. Um problema apontado na decomposição OAO é que a resposta de um preditor para um par de classes (i, j) na realidade não fornece informação alguma quando o exemplo não pertence às classes i ou j (Alpaydin e Mayoraz, 1999).

Em uma estratégia decomposicional alternativa, Dietterich e Bariki (Dietterich e Bariki, 1995) propuseram o uso de códigos de correção de erros para representar as k classes de um problema. Para tal, os autores sugerem que o valor do número de classificadores l seja maior que o número mínimo necessário para diferenciar cada classe unicamente. Os bits adicionais introduzem uma redundância na codificação das classes e têm a utilidade de prover ao sistema a capacidade de se recuperar de eventuais erros cometidos por alguns preditores no processo de classificação de um novo exemplo. Por esse motivo, essa técnica possui a denominação de decomposição por códigos de correção de erros (ECOC, do Inglês *Error Correcting Output Codes*). As matrizes de códigos nesse caso assumem valores em $\{-1, +1\}$ e utiliza-se a distância de *Hamming* na decodificação das saídas dos classificadores.

Com esse procedimento de correção de erros, Dietterich e Bariki (Dietterich e Bariki, 1995) sugerem que a solução de um problema multiclasses por uma técnica de AM seja vista como uma tarefa de comunicação. Dado um novo exemplo, a sua classe correta é transmitida por meio de um canal, o qual é constituído de seus atributos, dos dados de treinamento e do algoritmo de aprendizado. Devido aos erros que podem estar presentes nos atributos do exemplo, nos dados de treinamento utilizados ou em falhas no processo de aprendizado, a informação pode ser corrompida. Para prover ao sistema a capacidade de se recuperar desses erros de transmissão, a classe é codificada por um código de correção de erros e cada bit do mesmo é transmitido separadamente, ou seja, por execuções separadas do algoritmo de aprendizado.

Para possibilitar a correção de erros, Dietterich e Bariki (Dietterich e Bariki, 1995) sugerem que os códigos das classes contidos em $\mathbf{M}$ sejam bem separados segundo a distância de *Hamming*. Sendo d_l a distância mínima entre qualquer par de linhas de $\mathbf{M}$, então o classificador multiclasses final é capaz de corrigir ao menos $\left\lfloor \frac{d_l-1}{2} \right\rfloor$ bits incorretos em uma predição. Como segundo a função de decodificação de *Hamming* cada bit incorreto implica em se afastar em uma unidade do código da classe correta, cometendo $\left\lfloor \frac{d_l-1}{2} \right\rfloor$ erros, o código mais próximo ainda será o correto (Dietterich e Bariki, 1995). Segundo esse princípio, a codificação OAA é incapaz de se recuperar de qualquer erro, uma vez que $d_l = 2$.

Além disso, na construção de bons códigos de correção de erros, deve-se estimular também que os erros dos classificadores binários gerados sejam não correlacionados. Exige-se então a separação entre as colunas de $\mathbf{M}$, ou seja, a distância de *Hamming* entre cada par de colunas deve ser grande. Se no algoritmo de aprendizado a inversão das classes positivas e negativas produz o mesmo classificador

(ou seja, $f = -f$), então deve-se também fazer com que a distância de *Hamming* entre cada coluna e o complemento das outras seja grande.

Com base nessas observações, Dietterich e Bariki (Dietterich e Bariki, 1995) propuseram quatro técnicas para o desenvolvimento de matrizes de códigos com boa capacidade de correção de erros. A escolha de cada uma delas é determinada pelo número de classes do problema. Para $k \leqslant 7$, esses autores recomendam o uso de um código exaustivo, que consiste da combinação de $2^{k-1} - 1$ classificadores binários unicamente com rótulos $+1$ e -1, conforme ilustrado na Figura 18.10b para um problema com quatro classes. A distância d_l em uma matriz gerada pelo método exaustivo é de 2^{k-2}. Se $8 \leqslant k \leqslant 11$, é aplicado um método que seleciona colunas do código exaustivo. Para $k > 11$, tem-se duas opções: um método baseado no algoritmo *hill-climbing* e a geração de códigos BCH (Boser e Ray-Chaudhuri, 1960), os quais provêm da teoria de códigos de correção de erros em comunicação.

Uma crítica comum às estratégias OAA, OAO e ECOC é que todas realizam a decomposição do problema *a priori*, sem levar em consideração as propriedades e as características de cada aplicação. Essa foi a motivação de diversos trabalhos para a proposição de técnicas para encontrar matrizes de códigos adaptadas à solução de cada problema multiclasses, tais como (Lorena e Carvalho, 2007b,a; Lorena, 2006; Mayoraz e Moreira, 1997; Alpaydin e Mayoraz, 1999; Dekel e Singer, 2003; Ratsch et al., 2003), entre outros.

Na próxima seção discute-se como Algoritmos Genéticos podem ser empregados na construção de matrizes de códigos para problemas multiclasses.

Construindo Matrizes de Códigos para Problemas Multiclasses

A construção de matrizes de códigos para problemas de classificação multiclasses pode ser formulada como um problema de busca e otimização. Conforme mencionado na seção anterior, há $0.5\left(3^k + 1\right) - 2^k$ diferentes preditores binários para um problema com k classes. Um número combinatorial de combinações desses classificadores é possível, correspondendo a diferentes decomposições do problema multiclasses. Baseado nessa observação, alguns trabalhos empregaram Algoritmos Genéticos na construção de matrizes de códigos.

Verifica-se na literatura de comunicação o vasto uso de AGs para a obtenção de Códigos de Correção de Erros (*Error Correcting Codes* - ECC). Como exemplo, pode-se citar (Alba e Chicano, 2004; Alba et al., 2002; Simón et al., 2006; Wallis e Houghten, 2002; Dontas e De Jong, 1990). A construção de ECCs nesse contexto pode ser resumida a encontrar k códigos com l bits cada, que sejam capazes de corrigir um número máximo de erros pré-determinado. Esse é um problema de otimização NP-difícil (Alba e Chicano, 2004), em que geralmente se tem objetivos conflitantes: encontrar códigos com o menor tamanho possível (implicando em rápida transmissão) e maximizar d_m (para uma maior capacidade de correção de erros), o que sugere incluir mais redundância nos códigos (mais bits e, então, o uso de códigos maiores). Entretando, agrupando os k códigos em uma matriz, não se requer a separação entre colunas e também não é evitada a presença de colunas constantes (com o mesmo elemento em todas as linhas), dificultando o uso direto desses algoritmos para encontrat ECOCs para problemas multiclasses - ou seja, matrizes de códigos com propriedades de correção de erros segundo a abordagem proposta em (Dietterich e Bariki, 1995).

Pimenta (Pimenta, 2005) adaptou um algoritmo denominado GARA (*Genetic Algorithm with Repulsion Algorithm*) (Alba e Chicano, 2004) da teoria de telecomunicações para a geração de matrizes ECOC. Os cromossomos desse AG são cadeias binárias com kxl elementos, formadas pela concatenação dos códigos na matriz de códigos $\mathbf{M}$, como ilustrado na Figura 18.11. Um alfabeto binário é usado nas matrizes desse trabalho.

A função de avaliação da aptidão dos individuos neste AG considera o quanto os códigos em $\mathbf{M}$ estão separados no espaço de l dimensões. Há ainda um termo que penaliza a presença de colunas iguais e complementares nas matrizes obtidas, garantindo assim a separação entre as colunas da matriz.

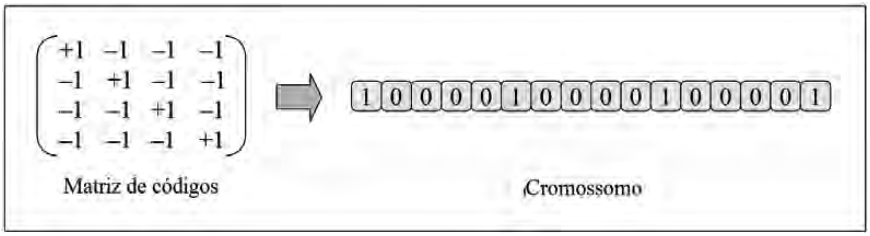


Figura 18.11: Exemplo de cromossomo na abordagem GARA (Alba e Chicano, 2004)

Empregou-se torneio binário na seleção de indivíduos para reprodução, e o operador de cruzamento utilizado foi o de um ponto. Para mutação, uma iteração do algoritmo RA (*Repulsion Algorithm*) foi utilizada como um procedimento de busca local. O algoritmo RA é baseado no comportamento físico de partículas com mesma carga em uma esfera. Nessa situação, as particícúlas movem pela esfera até que um equilíbrio seja atingido. Nessa abordagem, cada código de uma classe foi considerado como uma partícula carregada, posicionada no vértice de um hipercubo. Permite-se o movimento de códigos entre vértices vizinhos, o que equivale a inverter um bit dentro do código. O AG pára quando um número máximo de gerações é atingido ou quando uma matriz ótima, de acordo com a função de avaliação, é encontrada.

O algoritmo GARA foi comparado ao RA e a um algoritmo denominado *Persecution Algorithm* (PA), também proposto em (Pimenta, 2005). O PA adiciona e remove colunas de um ECOC inicial, buscando maximizar a sua qualidade, a qual é medida de acordo com sua capacidade de correção de erros. Em geral, o PA obteve melhores resultados para ECOCs, embora o GARA também tenha obtido bons ECOCs. A validade foi medida pelo critério de se evitar colunas iguais, complementares ou constantes nas matrizes de códigos obtidas.

Lorena e Carvalho (Lorena, 2006; Lorena e Carvalho, 2007b), por sua vez, empregaram AGs na obtenção de matrizes de códigos adaptadas a cada problema multiclasses considerado. Os AGs formulados lidam com três objetivos: (a) minimizar o erro da matriz quando esta é empregada na solução do problema multiclasses; (b) minimizar o número de colunas na matriz de códigos, representando a busca por decomposições mais simples, que empreguem menos classificadores binários; (c) evitar a presença de colunas equivalentes na matriz, colunas iguais ou complementares, as quais representam a repetição de um mesmo classificador binário na decomposição. O objetivo foi buscar uma matriz de códigos sem colunas equivalentes com um bom desempenho na solução do problema multiclasses e também com um número reduzido de classificadores binários. Duas variantes de AGs foram empregadas nesse problema multi-objetivo: uma lexicográfica, em que os múltiplos objetivos são ordenados segundo uma preferência, e outra baseada no algoritmo SPEA2 (*Strength Pareto Evolutionary Algorithm* 2) (Zitzler et al., 2002), baseado em uma técnica de otimização multiobjetivo.

Os cromossomos foram diretamente representados como matrizes de códigos, com uma codificação ternária. Assim, cada indivíduo corresponde a uma possível matriz de códigos $\mathbf{M}$ de tamanho kxl e com elementos no conjunto $\{-1, 0, +1\}$. Essa representação foi considerada mais intuitiva para representar as soluções para o problema em questão. Para determinar o número de classificadores binários nas matrizes, os indivíduos em uma mesma população continham diferentes números de colunas l. De acordo com essa estratégia, dois possíveis indivíduos para um problema com quatro classes podem ser visualizados na Figura 18.12. O usuário limita o número máximo de classificadores permitidos nas matrizes de códigos manipuladas pelo AG. As matrizes também devem ter no mínimo $\lceil \log_2 k \rceil$ classificadores binários, o que corresponde ao mínimo necessário para dividir k classes (Mayoraz e Moreira, 1997).

Na geração da população inicial, um teste de consistência foi aplicado a cada coluna das matrizes,

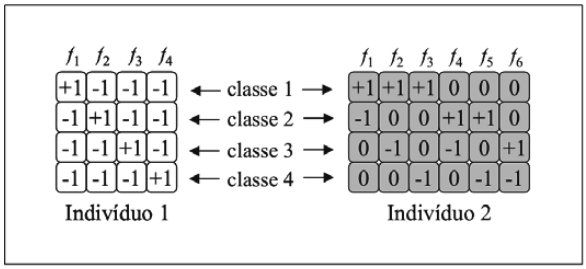


Figura 18.12: Ilustração de dois possíveis indivíduos para um problema com quatro classes

garantindo que elas possuam rótulos positivos e negativos, representando assim partições binárias válidas. Também é possível fornecer à população inicial os códigos das abordagens OAA, OAO e ECOC tradicionais.

Para avaliar cada indivíduo, os AGs consideram o poder preditivo do conjunto de classificadores binários, representado em sua matriz de códigos, na solução do problema multiclasses. Um erro em um conjunto de validação é calculado. Classificações desconhecidas, que ocorrem caso mais de uma linha da matriz de códigos apresente distância mínima em relação à cadeia de predições, também foram consideradas como erro. Essa medida de erro deve ser minimizada pelos AGs. O segundo objetivo de ambos AGs formulados é a minimização do número de classificadores binários nas matrizes. A versão lexicográfica favorece a minimização do erro, colocando a redução do número de classificadores binários em segunda ordem de importância. No SPEA2, esse valor foi considerado como um segundo objetivo a ser minimizado usando as relações de dominância de Pareto.

Durante a operação dos AGs, uma matriz de códigos gerada pode possuir colunas equivalentes em sua composição. Para inibir essa ocorrência, cada variante de AG empregou uma estratégia. Na versão lexicográfica, essa característica foi penalizada na função de avaliação dos indivíduos, enquanto que o algoritmo SPEA2 considerou a proporção de colunas equivalentes nos indivíduos como um terceiro objetivo a ser minimizado simultaneamente ao erro de validação e ao número de colunas das matrizes.

A ordenação de objetivos no AG lexicográfico foi realizada nos passos de elitismo e de seleção de indivíduos para reprodução. A cada ocorrência de empate entre duas matrizes, considerando sua aptidão em solucionar o problema multiclasses, o indivíduo com menor número de classificadores era escolhido.

Os AGs foram executados até que um número máximo de gerações fosse atingido. A seleção de indivíduos foi realizada segundo o torneio binário, e os operadores de cruzamento e mutação foram desenvolvidos considerando a representação dos indivíduos e as características do problema de busca de matrizes de códigos formulado. Um tipo de cruzamento empregado, por exemplo, considerou trocas de colunas entre dois indivíduos. Essa operação corresponde à troca de classificadores binários entre duas matrizes, motivado pelo fato de que um classificador binário pode ser mais efetivo em uma combinação multiclasses alternativa. No caso da mutação, empregou-se a alteração de elementos aleatórios nas matrizes e a adição e a remoção de colunas em indivíduos.

Como foram formulados diferentes operadores de cruzamento e mutação, utilizou-se um critério proposto em (Martí et al., 2005) na opção por cada um deles. Nesse caso, cada operador é selecionado probabilisticamente de acordo com o seu desempenho na obtenção de boas matrizes em gerações anteriores. Ou seja, operadores que produzem melhores soluções em gerações prévias têm maiores chances de serem aplicados e sua importância é então adaptada pelos AGs.

A cada execução do SPEA2, um conjunto de soluções é obtido. Para escolher uma solução em particular, foi considerada a distância a um ponto de referência. Esse ponto possui taxa de erro nula, um número mínimo de classificadores binários e não contém colunas equivalentes. Assim, a solução

com avaliações mais próximas a esse ponto é a escolhida.

Os AGs descritos foram avaliados em conjuntos de dados reais (da área de Bioinformática) e de *benchmarks*. Eles foram empregados na busca de matrizes de códigos com desempenho preditivo similar ou superior ao da decomposição OAA usando SVMs como classificadores base. O AG lexicográfico obteve matrizes de códigos com bom desempenho preditivo e com um número menor de classificadores binários que a abordagem OAA. Por outro lado, o SPEA2 não obteve sucesso nesse problema, pois, embora tenha obtido matrizes com menos classificadores binários, as taxas de acerto obtidas por elas não eram comparáveis às da matriz OAA. Além disso, o SPEA2 também encontrou matrizes com colunas equivalentes.

A busca por uma decomposição para um problema multiclasses também pode ser vista como a busca de um comitê (*ensemble*) de classificadores binários para a sua solução. Baseado nisso, Kuncheva (Kuncheva, 2005) propôs o uso de medidas de diversidade da literatura de comitês de classificadores para a geração de matrizes de códigos para problemas multiclasses. Nesse caso, as matrizes consideradas possuíam apenas elementos binários. A autora propôs uma medida conjunta que considera a diversidade das linhas e das colunas de uma dada matriz de códigos. Assim, AGs forma empregados para encontrar matrizes que maximizassem a medida de diversidade formulada. Como no GARA, cada cromossomo foi representado por uma cadeia formada pela concatenação dos códigos da matriz de códigos (Figura 18.11). Somente a mutação foi aplicada como operador genético, implementada por um procedimento de mudança de bits, e um número máximo de iterações foi utilizado como critério de parada. A avaliação experimental considerou a capacidade do AG em otimizar as medidas de diversidade formuladas, mas não foram incluídos experimentos em conjuntos de dados reais ou de *benchmarks*.

Empregando conceitos do trabalho de Kuncheva (Kuncheva, 2005), Shen e Tan (Shen e Tan, 2005) também usaram AGs na busca de matrizes de códigos. Eles adaptaram as medidas de diversidade do trabalho anterior considerando um alfabeto ternário para as matrizes de códigos e as empregaram na formulação de uma medida que considera margens de separação entre os códigos. Assim, procurou-se então maximizar essa margem de separação. A codificação dos cromossomos foi a mesma empregada anteriormente. Previamente ao cálculo da aptidão de um indivíduo, foram removidas as colunas equivalentes ou constantes de sua matriz de códigos. Contudo, o tamanho original l dos códigos foi mantido no cálculo das aptidões, resultando na penalização desse tipo de matriz. O AG pára quando os valores de aptidão não se alteram por um dado período de tempo ou após um número máximo de gerações. Empregou-se cruzamento de um ponto, mutação uniforme e a seleção por roleta. O AG proposto foi avaliado experimentalmente em dois conjuntos de dados para o diagnóstico de câncer. As matrizes de códigos do AG, usando SVMs lineares como classificadores base, foram superiores a outras matrizes de códigos, como OAA e OAO, assim como ao algoritmo k-vizinhos mais próximos (*k-nearest neighbor* - kNN) e Árvores de Decisão (ADs). Foram utilizados $l = \lceil 10 * \log_2(k) \rceil$ classificadores binários.

4. Conclusões

Algoritmos evolutivos têm sido cada vez mais utilizados para diversas finalidades, tanto na computação como em outras áreas de aplicação. Este capítulo descreveu três importantes aplicações de algoritmos genéticos: ajuste de parâmetros dos algoritmos de indução de classificadores; indução de árvores de decisão; e decomposição de problemas multiclasses.

Grande parte dos algoritmos de AM possuem parâmetros cujos valores devem ser especificados pelo usuário. Tais valores influenciam diretamente no desempenho de modelos induzidos. A tarefa de encontrar os melhores valores para esses parâmetros é um problema de otimização, e foi possível observar neste capítulo que os algoritmos genéticos são muito bem aplicados nesse contexto, melhorando o desempenho e robustez dos modelos.

Também foi apresentado o algoritmo genético multi-objetivo para indução de árvores de decisão chamado LEGAL-Tree. LEGAL-Tree apresentou algumas diferenças em relação a outros algoritmos evolutivos para o mesmo propósito, com destaque para o método de geração da população inicial e, principalmente, pela abordagem multi-objetiva proposta na função de *fitness*. Enquanto outros trabalhos lidam apenas com um objetivo ou tratam problemas multi-objetivos por meio das abordagens de fórmula-ponderada ou de Pareto, LEGAL-Tree propôs o uso da abordagem lexicográfica, em que diversos critérios podem ser avaliados de acordo com suas respectivas prioridades para o domínio do problema. Essa abordagem é relativamente simples de implementar e controlar, e também não sofre dos problemas sofridos por fórmula-ponderada e Pareto.

Por fim, esse capítulo também descreveu alguns trabalhos relacionados ao uso de AGs na obtenção de matrizes de códigos para problemas multiclasses. As abordagens ilustradas consideraram: (a) a obtenção de códigos com boas propriedades de correção de erros de classificação; (b) a adaptação dos códigos a cada problema multiclasses considerado; (c) a maximização da diversidade dos classificadores combinados na decomposição.