LEGAL-Tree: Um algoritmo genético multi-objetivo lexicográfico para indução de árvores de decisão

Márcio Porto Basgalupp

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura:_____

LEGAL-Tree: Um algoritmo genético multi-objetivo lexicográfico para indução de árvores de decisão

Márcio Porto Basgalupp

Orientador: *Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho*
Co-orientador: *Prof. Dr. Alex Alves Freitas*

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP como parte dos requisitos necessários à obtenção para do título em Doutor em Ciências de Computação e Matemática Computacional.

USP - São Carlos
fevereiro/2010

Agradecimentos

À minha mãe Neida Mariza Veiga Porto e ao meu pai Jorge Luiz Basgalupp (*in memoriam*), responsáveis diretos por cada uma das minhas realizações.

Às minhas irmãs Ana Luiza, Suélen e Taiane, e à minha vó Eulina.

Ao meu afilhado Diogo, que é o gremista mais lindo que eu já vi.

Aos meus demais familiares e amigos, simplesmente por existirem e fazerem parte da minha vida.

Aos professores João Baptista da Silva, João Artur de Souza, Gertrudes Dandolini, Gil Carlos Medeiros, Karin Becker e Duncan Ruiz, os quais levarei sempre comigo por participarem efetivamente da minha vida acadêmica e pessoal.

A todos os colegas do LABIC e BIOCOMP, do ICMC-USP, e do GPIN, da PUCRS, com os quais compartilhei ótimos momentos e que, direta ou indiretamente, influenciaram na realização deste trabalho.

A todos os funcionários e professores do ICMC-USP por me proporcionarem o excelente ambiente de trabalho que tive durante todo o tempo do doutorado.

Ao colega e amigo Rodrigo Barros, que também participou efetivamente do desenvolvimento desta pesquisa.

Ao meu co-orientador, Prof. Dr. Alex Alves Freitas, não apenas pela dedicação na realização do meu trabalho, mas também por ser extremamente atencioso e prestativo durante o tempo em que estive na Inglaterra.

Ao meu orientador, Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho, o qual é uma das pessoas mais gentis que já conheci na minha vida. Se eu terminar a minha carreira com a metade de seu currículo, serei realizado profissionalmente. Mas, se eu terminar a vida com a metade de sua humildade, serei uma pessoa realizada.

Aos autores do algoritmo GALE, Xavier Llora and Josep Garrell, por gentilmente me fornecerem esse algoritmo.

À FAPESP, pelo financiamento deste trabalho.

Resumo

Dentre as diversas tarefas em que os algoritmos evolutivos têm sido empregados, a indução de regras e de árvores de decisão tem se mostrado uma abordagem bastante atrativa em diversos domínios de aplicação. Algoritmos de indução de árvores de decisão representam uma das técnicas mais populares em problemas de classificação. Entretanto, os algoritmos tradicionais de indução apresentam algumas limitações, pois, geralmente, usam uma estratégia gulosa, *top down* e com particionamento recursivo para a construção das árvores. Esses fatores degradam a qualidade dos dados, os quais podem gerar regras estatisticamente não significativas. Este trabalho propõe o algoritmo LEGAL-Tree, uma nova abordagem baseada em algoritmos genéticos para indução de árvores de decisão. O algoritmo proposto visa evitar a estratégia gulosa e a convergência para ótimos locais. Para isso, esse algoritmo adota uma abordagem multi-objetiva lexicográfica. Nos experimentos realizados sobre bases de dados de diversos problemas de classificação, a função de *fitness* de LEGAL-Tree considera as duas medidas mais comuns para avaliação das árvores de decisão: acurácia e tamanho da árvore. Os resultados obtidos mostraram que LEGAL-Tree teve um desempenho equivalente ao algoritmo SimpleCart (implementação em Java do algoritmo CART) e superou o tradicional algoritmo J48 (implementação em Java do algoritmo C4.5), além de ter superado também o algoritmo evolutivo GALE. A principal contribuição de LEGAL-Tree não foi gerar árvores com maior acurácia preditiva, mas sim gerar árvores menores e, portanto, mais compreensíveis ao usuário do que as outras abordagens, mantendo a acurácia preditiva equivalente. Isso mostra que LEGAL-Tree obteve sucesso na otimização lexicográfica de seus objetivos, uma vez que a idéia era justamente dar preferência às árvores menores (em termos de número de nodos) quando houvesse equivalência de acurácia.

Abstract

Among the several tasks evolutionary algorithms have been successfully employed, the induction of classification rules and decision trees has been shown to be a relevant approach for several application domains. Decision tree induction algorithms represent one of the most popular techniques for dealing with classification problems. However, conventionally used decision trees induction algorithms present limitations due to the strategy they usually implement: recursive top-down data partitioning through a greedy split evaluation. The main problem with this strategy is quality loss during the partitioning process, which can lead to statistically insignificant rules. In this thesis we propose the LEGAL-Tree algorithm, a new GA-based algorithm for decision tree induction. The proposed algorithm aims to prevent the greedy strategy and to avoid converging to local optima. For such, it is based on a lexicographic multi-objective approach. In the experiments which were run in several classification problems, LEGAL-Tree's fitness function considers two of the most common measures to evaluate decision trees: accuracy and tree size. Results show that LEGAL-Tree performs similarly to SimpleCart (CART Java implementation) and outperforms J48 (C4.5 Java implementation) and the evolutionary algorithm GALE. LEGAL-Tree's main contribution is not to generate trees with the highest predictive accuracy possible, but to provide smaller (and thus more comprehensible) trees, keeping a competitive accuracy rate. LEGAL-Tree is able to provide both comprehensible and accurate trees, which shows that the lexicographic fitness evaluation is successful since its goal is to prioritize smaller trees (fewer number of nodes) when there is equivalency in terms of accuracy.

Sumário

Lista de Abreviaturas **xv**

Lista de Figuras **xvii**

Lista de Tabelas **xix**

Lista de Algoritmos **xxi**

1 Introdução **1**

1.1 Motivação 2

1.2 Objetivos 4

1.3 Organização do trabalho 5

2 Árvores de Decisão **7**

2.1 Introdução 7

2.2 Funcionamento básico 10

2.3 Medidas para selecionar a melhor divisão 12

2.3.1 Ganho de Informação 13

2.3.2 Gain Ratio 13

2.3.3 Gini 14

2.3.4 Outras medidas 14

2.4 Representação dos nodos 14

2.4.1 Atributos categóricos 15

2.4.2 Atributos contínuos 16

2.5 Tratamento de valores desconhecidos 17

2.6 Poda 18

2.7 Avaliação de classificadores 19

2.8 Algoritmos de indução de árvores de decisão 22

2.8.1 ID3 23

2.8.2 C4.5 23

2.8.3 CART 24
2.8.4 NBTree 24
2.8.5 ADTree 25
2.8.6 LMT 26
2.8.7 BFTree 27
2.9 Considerações finais 28

3 Algoritmos Evolutivos 29
3.1 Introdução 29
3.2 Representação 31
3.2.1 Representação binária 31
3.2.2 Representação inteira 32
3.2.3 Representação por valores reais 32
3.2.4 Representação por árvores 32
3.3 População inicial 33
3.4 Função de fitness 33
3.4.1 Um único objetivo 34
3.4.2 Multi-objetivos 34
3.5 Seleção 39
3.6 Operadores genéticos 41
3.6.1 *Crossover* 41
3.6.2 Mutação 42
3.7 Indução de árvores de decisão 43
3.8 Considerações finais 46

4 Algoritmo LEGAL-Tree 49
4.1 Representação da solução 49
4.2 Gerando a população inicial 50
4.2.1 Decision stumps 50
4.2.2 BeamClassifier 52
4.3 Tratamento de valores desconhecidos 54
4.4 Função de fitness multi-objetiva baseada na abordagem lexicográfica 56
4.5 Seleção 57
4.6 Crossover 59
4.7 Mutação 60
4.8 Factibilidade das soluções 60
4.9 Considerações finais 62

5 Análise Experimental do Algoritmo LEGAL-Tree 63
5.1 Plano experimental 63
5.2 Análise dos resultados 66
5.3 Considerações finais 80

6 Conclusão **85**
6.1 Trabalhos futuros 86

Referências **94**

Lista de Abreviaturas

AE Algoritmo Evolutivo

AD Árvore de Decisão

AG Algoritmo Genético

AM Aprendizado de Máquina

CART *Classification and Regression Trees*

IA Inteligência Artificial

KDD *Knowledge Discovery in Databases*

MD Mineração de Dados

PG Programação Genética

RNA Redes Neurais Artificiais

SVM Máquinas de Vetores de Suporte

TDIDT *Top-Down Induction of Decision Tree*

WEKA *Waikato Environment for Knowledge Knowledge Analysis*

Lista de Figuras

2.1 Diagrama do processo de indução de um classificador e sua utilização na dedução de novos exemplos. 9
2.2 Uma possível árvore de decisão para o problema de diagnosticar pacientes. 11

3.1 Diagrama de atividades utilizado para representar o princípio básico de funcionamento de um algoritmo evolutivo. 30
3.2 Indivíduo codificado como um *string* binário. 31
3.3 Indivíduo codificado como um *string* de inteiros. 32
3.4 Indivíduo codificado como um *string* de valores reais. 32
3.5 Indivíduo codificado como uma árvore. 33
3.6 Curva de Pareto para o problema de minimizar a taxa de erro e o tamanho de uma árvore de decisão. 37
3.7 Roleta representando a proporção do intervalo de números reservado para cada um dos 10 (dez) indivíduos da população. 40
3.8 Um exemplo de *crossover* de 1 ponto. 41
3.9 Um exemplo de *crossover* multi-pontos. 42
3.10 Um exemplo de *crossover* uniforme. 42
3.11 Um exemplo de *crossover* entre dois indivíduos representados por árvores. 43
3.12 Um exemplo de mutação em um indivíduo com representação binária. . . . 43

4.1 Exemplo de representação de nodos categórico (Y) e numérico (X). 50
4.2 Combinando três *decision stumps* (A, B e C) para construção de um indivíduo de profundidade 2. 52
4.3 Exemplo de execução de BeamClassifier. 55
4.4 *Crossover* entre dois indivíduos, $P1$ e $P2$, formando dois novos indivíduos, $F1$ e $F2$. 59
4.5 Duas estratégias,"Mutação de redução" e "Mutação de expansão", para mutação dos indivíduos. 61
4.6 Filtros para eliminar redundâncias (Filtro Categórico) e inconsistências (Filtro Numérico). 61

5.1 Gráfico de barras das acurácias das árvores de decisão geradas 81
5.2 Gráfico de barras das acurácias das árvores de decisão geradas 82
5.3 Gráfico de barras dos tamanhos das árvores de decisão geradas 83
5.4 Gráfico de barras dos tamanhos das árvores de decisão geradas 84

Lista de Tabelas

2.1 Conjunto de dados para o diagnóstico da saúde de pacientes. 8
2.2 Exemplo de uma matriz de confusão para um problema de duas classes. . . 20

3.1 Representação dos valores de *fitness* pra cada indivíduo de uma população. 39
3.2 Características dos algoritmos propostos nos trabalhos relacionados. 46

5.1 Sumarização das bases de dados utilizadas nos experimentos. 64
5.2 Parâmetros utilizados por LEGAL-Tree nos experimentos 65
5.3 Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo LEGAL-Tree com a configuração LEGAL-DS. 67
5.4 Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo LEGAL-Tree com a configuração LEGAL-BC. 68
5.5 Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo LEGAL-Tree com a configuração LEGAL-ST. 69
5.6 Acurácia, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo GALE. 70
5.7 Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo J48. 71
5.8 Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo SimpleCart. 72
5.9 Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo BeamClassifier. 73
5.10 Médias das acurácias preditivas (%) e os respectivos desvios-padrão. 74
5.11 Considerando a acurácia para o conjunto de teste, são apresentadas as bases de dados em que LEGAL-DS, LEGAL-BC e LEGAL-ST são significativamente melhores que as outras abordagens de acordo com o teste-t. 75

5.12 Considerando a acurácia para o conjunto de teste, são apresentadas as bases de dados em que LEGAL-DS, LEGAL-BC e LEGAL-ST são significativamente piores que as outras abordagens de acordo com o teste-t. 76
5.13 Médias dos tamanhos das árvores (número de nodos) e os respectivos desvios-padrão. 77
5.14 Considerando o tamanho das árvores, são apresentadas as bases de dados em que LEGAL-DS, LEGAL-BC e LEGAL-ST são significativamente melhores que as outras abordagens de acordo com o teste-t. 79
5.15 Considerando o tamanho das árvores, são apresentadas as bases de dados em que LEGAL-DS, LEGAL-BC e LEGAL-ST são significativamente piores que as outras abordagens de acordo com o teste-t. 79

Lista de Algoritmos

1 Geração dos *Decision Stumps* 51
2 Algoritmo BeamClassifier 53
3 Função de *fitness* lexicográfica 58

Capítulo

1

Introdução

A mineração de dados (do inglês *Data Mining*) é a ciência e tecnologia de explorar dados com o intuito de descobrir padrões até então desconhecidos. Embora alguns autores considerem como sinônimo de *Knowledge Discovery in Databases* (KDD), a mineração de dados é apenas uma, e muito importante, das etapas do processo de KDD (Fayyad et al., 1996; Han, 2001; Tan et al., 2005; Rokach & Maimon, 2008).

Atualmente, existem muitas bases de dados no mundo todo, e a acessibilidade e abundância desses dados tornam a mineração de dados um assunto de considerável importância e necessidade. Uma das tarefas de mineração de dados mais utilizadas é a classificação (Tan et al., 2005), cujo objetivo geral é construir um modelo conciso de distribuição dos valores de um atributo denominado classe em função dos valores dos demais atributos, designados preditivos. Existem várias maneiras de um modelo de classificação representar o conhecimento adquirido. Regras de decisão, árvores de decisão, *naive Bayes*, redes bayesianas, k-vizinhos mais próximos, máquinas de vetores de suporte e redes neurais artificiais são alguns exemplos. Em especial, as árvores de decisão constituem uma técnica poderosa e amplamente utilizada em tarefas de mineração de dados. Seu sucesso pode ser explicado por diversos fatores, tais como (Tan et al., 2005): (i) interpretabilidade do conhecimento adquirido - uma árvore de decisão usa uma representação gráfica e pode ser facilmente convertida em regras; (ii) robustez na presença de ruídos; (iii) algoritmos de indução de árvores de decisão não são computacionalmente custosos, mesmo para grandes conjuntos de dados; e (iv) capacidade de lidar com atributos redundantes e irrelevantes, os quais, se não tratados adequadamente, podem prejudicar a acurácia do modelo de classificação. Alguns algoritmos bem conhecidos para indução de árvores de decisão são o ID3 (Quinlan, 1986), o C4.5 (Quinlan, 1993) e o CART (Breiman et al., 1984).

1.1 Motivação

Algoritmos de indução de árvores de decisão geralmente usam uma estratégia gulosa, *top down* e com particionamento recursivo para a construção das árvores. No entanto, há pelo menos dois problemas relacionados a essas características. Primeiro, as estratégias gulosas geralmente produzem soluções ótimas locais ao invés de globais. Segundo, o particionamento recursivo degrada iterativamente a qualidade da base de dados, pois quanto maior o número de vezes em que uma base de dados é particionada, menores são os sub-conjuntos resultantes dessas partições e, conseqüentemente, tornam não estatisticamente significativos são os resultados obtidos por quaisquer critérios de manipulação de dados (por exemplo, por meio do cálculo de entropia). Além disso, o número de exemplos pertencentes a um nodo folha da árvore pode ser tão pequeno que tende a tornar também não confiável a decisão sobre a classe representada por esse nodo folha. Isso pode gerar uma quantidade grande de nodos folhas representando classes com poucos exemplos, contribuindo para o super-aprendizado (*overfitting*) do conjunto de treinamento (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. Métodos *ensemble* como *random forests*, *boosting* e *bagging* são as mais conhecidas abordagens baseadas em (ii). *Random forests* (Nadeau & Bengio, 2003) criam um conjunto inicial de árvores baseadas nos valores de conjuntos independentes de vetores aleatórios, gerados por uma distribuição de probabilidades de um conjunto de treinamento. 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 treinamento. 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, em geral, melhora a taxa de acerto quando comparado a um único classificador. No entanto, o uso de *ensembles* tende a reduzir a interpretabilidade do modelo preditivo quando comparado a um único classificador. Um único modelo de classificação pode ser interpretado por um especialista; porém, na prática, é bastante complicado para um especialista interpretar uma combinação de classificadores, mesmo que eles sejam, isoladamente, fáceis de serem compreendidos. Além de ser tediosa e demorada, tal

interpretação para o especialista possui também um problema fundamental: os modelos de classificação que compõem um *ensemble* geralmente estão sujeitos a 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).

Com o objetivo de evitar esses problemas inerentes às abordagens supracitadas, alguns trabalhos propuseram o uso de algoritmos evolutivos para indução de árvores de decisão (Cantu-Paz & Kamath, 2000; Kretowski, 2004; Koza, 1991; Zhao, 2007; Estrada-Gil et al., 2007; Bot & Langdon, 2000; Loveard & Ciesielski, 2001, 2002; Shirasaka et al., 1998; Zhao & Shirasaka, 1999; Tür & Güvenir, 1996; Kretowski & Grzes, 2005; Z. Bandar & McLean, 1999; Llora & Garrell, 2001; Fu et al., 2003a,b, 2006; Gray & Fan, 2008). Ao invés de uma busca local, os algoritmos evolutivos realizam uma busca global no espaço de soluções candidatas (Goldberg, 1989), aumentando as chances de convergir para uma boa solução ou até para a solução ótima.

Conhecimentos descobertos em mineração de dados não devem ser apenas precisos, mas também interpretáveis para o usuário (Witten & Frank, 1999). A interpretabilidade pode ser estimada pelo tamanho do classificador - assume-se que classificadores menores são preferíveis quando outros critérios permanecem iguais. Isso é justificado por *Occam's razor* (Domingos, 1999), que essencialmente diz que, dentre múltiplas hipóteses que são equivalentemente consistentes com os dados, deve ser escolhida aquela mais simples. Por essa razão, muitos dos trabalhos mencionados procuraram não apenas gerar árvores de decisão com boas taxas de acerto, mas também árvores mais compreensíveis (considera-se que árvores menores são mais fáceis de serem interpretadas). Isso caracteriza um problema de otimização multi-objetivo, o qual é muito comum em mineração de dados e em algoritmos evolutivos em geral.

De acordo com Freitas (2004), há três abordagens para lidar com problemas multi-objetivos: (i) a convencional abordagem de fórmula ponderada, em que um problema multi-objetivo é transformado em um problema de um único objetivo; (ii) a abordagem lexicográfica; e (iii) a abordagem de Pareto, em que é utilizado o conceito de dominância de Pareto (Coello et al., 2006). 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 maçãs com laranjas" (misturar critérios não comparáveis como, por exemplo, acurácia e tamanho da árvore) e "misturar diferentes unidades de medidas" (realizar operações com diferentes escalas e induzir *bias* ao 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, uma vez que o resultado dessa abordagem não é uma única solução, mas sim um conjunto de soluções. 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 outro. No contexto de árvores de decisão, é evidente que a acurácia é consideravelmente mais importante que o tamanho da árvore. No entanto, a abordagem de Pareto assumiria que ambos os critérios são igualmente relevantes se os mesmos fossem utilizados como objetivos de otimização.

A abordagem lexicográfica, contudo, não sofre dos problemas supracitados, é conceitualmente simples, fácil de usar, de implementar e de estender para objetivos adicionais. A grande vantagem dessa abordagem está na capacidade de atribuir diferentes prioridades a diferentes critérios a serem otimizados sem apresentar os problemas associados com a abordagem de fórmula ponderada. Isso é uma vantagem importante, pois em muitos domínios de aplicação essa capacidade é indispensável. No contexto de evolução de árvores de decisão, e de classificadores em geral, é evidente que a acurácia é um critério que deve ser prioritário quando comparado à interpretabilidade (número de nodos na árvore), e a abordagem lexicográfica permite que essa ordem de prioridade seja considerada. No entanto, até onde se sabe, a abordagem lexicográfica não foi utilizada (antes desta pesquisa) para evolução de árvores de decisão.

1.2 Objetivos

O objetivo deste trabalho é desenvolver o LEGAL-Tree (LExicographical Genetic Algorithm for Learning decision Trees), o qual é um novo algoritmo genético multi-objetivo baseado na abordagem lexicográfica para indução de árvores de decisão. Além de evitar os problemas inerentes aos algoritmos tradicionais de indução de árvores de decisão, espera-se, também, que a abordagem lexicográfica permita que sejam geradas árvores de decisão coerentes com os critérios de avaliação considerados, respeitando as respectivas ordens de prioridade.

Os objetivos específicos deste trabalho são:

1. Propor uma função de *fitness* multi-objetiva lexicográfica que atenda aos interesses dos usuários de árvores de decisão, ou seja, que permita a convergência para uma solução que não somente tenha uma boa taxa de acerto, mas que também seja compreensível ao usuário.

2. Incorporar ao algoritmo o máximo possível sobre o conhecimento da tarefa de classificação, aumentando, assim, a velocidade de convergência para uma boa solução ou até mesmo para a solução ótima. Isso se dá por meio de dois métodos de geração da população inicial.

3. Realizar uma análise experimental com o algoritmo LEGAL-Tree e avaliar a qualidade das árvores de decisão geradas. Até o presente momento, essa qualidade é medida em termos de acurácia e interpretabilidade das árvores.

4. Comparar os resultados obtidos pelo algoritmo LEGAL-Tree com outros algoritmos propostos na literatura, tanto algoritmos tradicionais como algoritmos evolutivos.

1.3 Organização do trabalho

Este trabalho é organizado como segue.

Capítulo 2: Árvores de Decisão
São descritos os principais conceitos sobre árvores de decisão, bem como os algoritmos tradicionais de indução de tais modelos.

Capítulo 3: Algoritmos Evolutivos
É apresentada uma revisão sobre algoritmos evolutivos. Também são apresentados trabalhos relacionados, os quais utilizam abordagens evolutivas para indução de árvores de decisão.

Capítulo 4: Algoritmo LEGAL-Tree
É descrito um novo algoritmo de indução de árvores de decisão, LEGAL-Tree, o qual é baseado em um algoritmo genético multi-objetivo lexicográfico.

Capítulo 5: Análise Experimental do Algoritmo LEGAL-Tree
São apresentados resultados experimentais em que o algoritmo LEGAL-Tree é aplicado a diversos domínios. É realizada também uma análise comparativa de LEGAL-Tree com outras abordagens, tanto tradicionais quanto evolutivas, para indução de árvores de decisão.

Capítulo 6: Conclusões
São apresentadas as conclusões deste trabalho e as sugestões para trabalhos futuros.

Capítulo

2

Árvores de Decisão

Este capítulo apresenta os principais conceitos sobre Árvores de Decisão (ADs). São destacados os algoritmos mais comuns para indução de ADs encontrados na literatura, bem como as principais variações que podem ocorrer entre diferentes algoritmos.

2.1 Introdução

Algoritmos de Aprendizado de Máquina (AM) são muito utilizados e se mostram muito poderosos para resolver problemas em vários domínios de aplicação (Mitchell, 1997). Eles são especialmente utilizados em: (i) problemas de mineração de dados em que grandes conjuntos de dados podem conter valiosas regularidades implícitas que podem ser descobertas automaticamente (ex: aprender regras gerais para análise de crédito financeiro); (ii) domínios mal compreendidos em que humanos não têm o conhecimento necessário para desenvolver algoritmos efetivos (ex: reconhecimento facial por imagens); e (iii) domínios em que o programa deve se adaptar dinamicamente a mudanças. Algoritmos de AM envolvem diferentes áreas de conhecimento, tais como Inteligência Artificial (IA), Probabilidade e Estatística, Complexidade Computacional, Teoria da Informação, Psicologia, Neurobiologia e Teoria de Controle.

Muitos algoritmos de AM utilizam conhecimentos da IA e da Estatística para construção de modelos capazes de adquirir conhecimento a partir de um conjunto de dados. Esse conjunto de dados é composto por objetos (ou exemplos) que são descritos por vários atributos (também chamados de propriedades ou características), e a cada um desses exemplos pode ser atribuída uma classe dentre um conjunto finito de classes possíveis. Atributos são variáveis observáveis e independentes, e seus domínios podem ser tanto contínuos (numéricos) como discretos (categóricos). A classe é uma variável dependente,

e seu valor é determinado a partir dos valores das respectivas variáveis independentes.

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 & Baranauskas, 2003). Todo algoritmo de aprendizado, ou algoritmo de indução, ou simplesmente indutor, possui um *bias* indutivo, que é a preferência de uma hipótese sobre outra dentre as hipóteses que são igualmente consistentes com os dados.

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, os valores da classe (atributo classe) dos exemplos do conjunto de treinamento são conhecidos. Se o domínio do atributo classe é discreto, tem-se um problema de classificação. Se é numérico, tem-se um problema de regressão ou aproximação de funções.

De acordo com Tan et al. (2005), classificação é a tarefa de aprender uma função alvo f que mapeia cada conjunto de atributos x, ou seja, cada exemplo, a uma das y classes pré-definidas. Em um problema de classificação, os dados de entrada podem ser descritos pelo par ordenado $(\bar{x}, y)$, em que $\bar{x}$ é um vetor que representa os atributos preditivos, $\bar{x} = (x_1, x_2, x_3, ..., x_n)$, e y é o rótulo da classe à qual esse exemplo pertence. Na Tabela 2.1, é apresentado um conjunto de dados para classificação do estado de saúde de pacientes. 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.

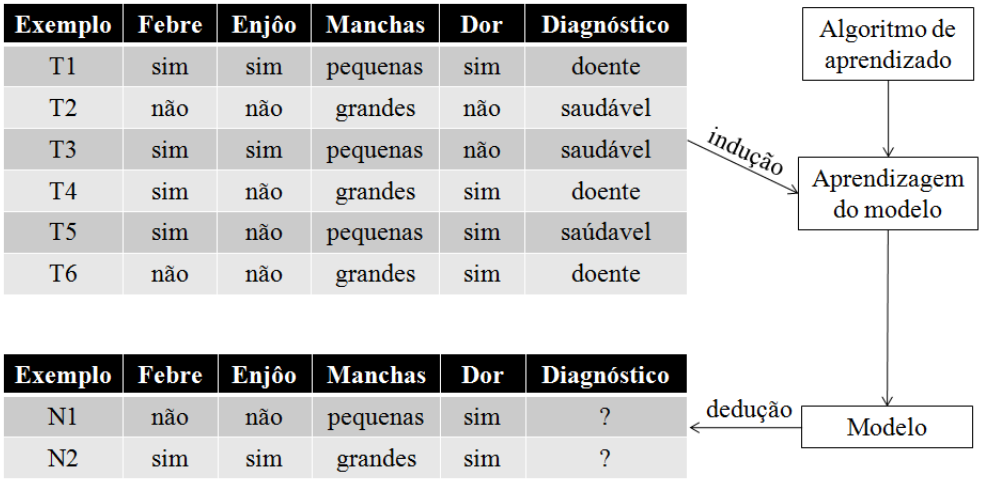
Tabela 2.1: Conjunto de dados para o diagnóstico da saúde de pacientes.

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 com sucesso em problemas de classificação e cada uma utiliza um algoritmo de aprendizado para construir um modelo (classificador) capaz de inferir a classe baseado nos atributos preditivos. Um desafio 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 utilizados durante a construção do modelo (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 da classe. Um exemplo de classificador desse tipo são as Redes Neurais Artificiais (RNAs). Os classificadores orientados a conhecimento, como as ADs e sistemas *fuzzy*, criam estruturas simbólicas que normalmente são mais compreensíveis do que os classificadores do tipo caixa-preta (Monard & Baranauskas, 2003).



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	saúdavel
T6	não	não	grandes	sim	doente

Exemplo	Febre	Enjôo	Manchas	Dor	Diagnóstico
N1	não	não	pequenas	sim	?
N2	sim	sim	grandes	sim	?

Figura 2.1: Diagrama do processo de indução de um classificador e sua utilização na dedução de novos exemplos.

Uma tarefa de classificação bastante conhecida é o diagnóstico médico, em que um paciente é descrito por atributos numéricos (ex: idade, altura, peso, temperatura do corpo, batimento cardíaco, pressão, etc.) e por atributos categóricos (ex: sexo, cor da pele, local da dor, etc.). A tarefa do classificador é, então, produzir um diagnóstico para o paciente, ou seja, atribuir ao paciente um ou vários diagnósticos possíveis (ex: saudável, pneumonia, influenza A, etc.).

Na Figura 2.1, 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 dos exemplos do conjunto de teste, ou seja, exemplos cujas classes são desconhecidas .

De acordo com Tan et al. (2005), a classificação pode ser utilizada para os seguintes propósitos: modelagem descritiva e modelagem preditiva.

Na modelagem descritiva, um modelo de classificação é utilizado como uma ferramenta para distinguir exemplos de diferentes classes. Como exemplo, um médico pode utilizar um modelo de classificação para identificar quais são as principais causas (sintomas) de uma determinada doença. A partir disso, é possível chegar a conclusões, por exemplo, que na grande maioria dos casos o paciente que está com a doença Influenza A apresentou febre e pneumonia. Quando há o interesse em análise descritiva, é desejável que o modelo de classificação seja fácil de interpretar, ou seja, que fique claro ao usuário o porquê de um determinado exemplo pertencer à uma determinada classe.

Na modelagem preditiva, um modelo de classificação é utilizado para classificar exemplos cujas classes são desconhecidas, ou seja, exemplos que não foram utilizados na construção do modelo. Como exemplo, um médico pode utilizar um conjunto de dados históricos de seus pacientes, em que as classes (diagnóstico) já são conhecidas, para construir um modelo de classificação a ser utilizado para diagnosticar novos pacientes. Dependendo da área de aplicação, a interpretabilidade do modelo de classificação não é essencial para previsão. No exemplo anterior, seria interessante um modelo com boa interpretabilidade, tornando possível medicar o paciente para curar os sintomas apresentados e, conseqüentemente, curar a doença diagnosticada.

Um modelo de classificação pode representar o seu conhecimento adquirido de várias formas, tais como regras de decisão, ADs, *naive Bayes*, redes bayesianas, k-vizinhos mais próximos, Máquinas de Vetores de Suporte (SVM), e RNAs. As ADs constituem uma técnica muito poderosa e amplamente utilizada em problemas de classificação. Uma das razões dessa técnica ser bastante utilizada é pelo fato do conhecimento adquirido ser representado por meio de regras. Essas regras podem ser expressas em linguagem natural, facilitando o entendimento por parte das pessoas.

2.2 Funcionamento básico

Para ilustrar o funcionamento básico de uma árvore de decisão, pode ser considerado novamente o problema de diagnosticar pacientes (Tabela 2.1). Suponha que um novo paciente chegue ao consultório do médico. Como o médico poderia diagnosticar o paciente? A primeira pergunta que pode ser feita ao paciente é se ele tem sentido dor. A seguir, outras perguntas podem ser feitas. Por exemplo, se o paciente está tendo febre ou enjôos, ou ainda se tem notado alguma mancha no corpo. Isso caracteriza uma forma de solucionar um problema de classificação por meio de perguntas sobre uma série de características de um exemplo, que no caso é o paciente. A cada pergunta respondida, outra pode ser realizada até que se chegue a uma conclusão sobre a classe a que pertence o exemplo. 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 2.2 ilustra uma possível árvore de decisão para o problema de diagnosticar pacientes, os quais podem ser classificados como "SAUDÁVEL" ou "DOENTE".

Segundo Tan et al. (2005), uma árvore de decisão 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, pois é o nodo que determina a qual classe o exemplo pertence.

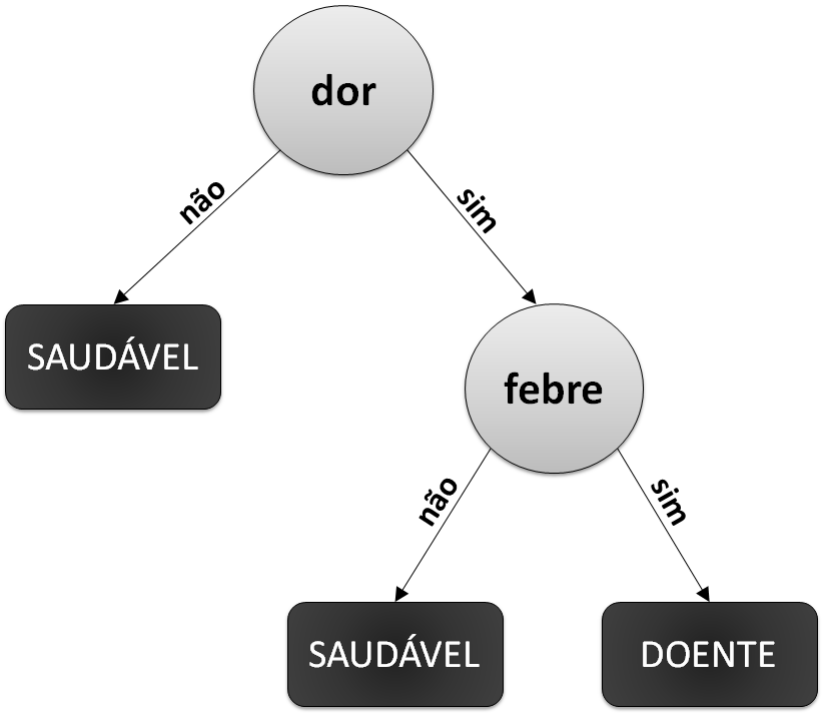


Figura 2.2: Uma possível árvore de decisão para o problema de diagnosticar pacientes.

Dessa forma, é possível utilizar uma árvore de decisão para classificar um novo paciente como saudável ou doente. Para isso, basta partir do nodo raiz da árvore e ir percorrendo-a, através das respostas aos testes dos nodos internos, até chegar em um nodo folha, o qual indica a classe correspondente do novo paciente. 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, no caso um médico.

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 fatorialmente à medida que o número de atributos aumenta. Logo, torna-se 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 uma solução ótima, apresentam resultados satisfatórios em tempo factível.

O *Top-Down Induction of Decision Tree* (TDIDT) é um algoritmo bem conhecido e é utilizado como base para muitos algoritmos de indução de árvores de decisão, dentre eles os mais conhecidos como ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993) e CART (Breiman et al., 1984). O TDIDT produz regras de decisão de forma implícita numa árvore de decisão, a qual é construída por sucessivas divisões dos exemplos de acordo com os valores de seus atributos preditivos. De acordo com Bramer (2007), esse processo é conhecido como particionamento recursivo. O esqueleto do algoritmo de TDIDT é baseado em três possibilidades sobre um conjunto de treinamento T contendo classes C_1, C_2, ..., C_k:

1. T contém um ou mais exemplos, sendo todos da classe C_j. Assim, a árvore de decisão para T é um nodo folha que identifica a classe C_j.

2. T não contém exemplos. A árvore de decisão também é um nodo folha, mas a classe associada deve ser determinada por uma informação externa. Por exemplo, pode-se utilizar o conhecimento do domínio do problema.

3. T contém exemplos pertencentes a mais de uma classe. Neste caso, a idéia é dividir T em sub-conjuntos que são, ou tendem a dirigir-se para, coleções de exemplos com classes únicas. Para isso, é escolhido um atributo preditivo A, que possui um ou mais possíveis resultados $O_1, O_2, ..., O_n$. T é particionado em sub-conjuntos T_1, T_2, ..., T_n, em que T_i contém todos os exemplos de T que têm resultado O_i para o atributo A. A árvore de decisão para T consiste de um nodo de decisão identificando o teste sobre o atributo A, e uma aresta para cada possível resultado, ou seja, n arestas. O mesmo algoritmo de indução de árvores de decisão (passos 1, 2 e 3) é aplicado recursivamente para cada sub-conjunto T_i, com i variando de 1 até n (número de possíveis resultados do atributo A).

Basicamente, o algoritmo TDIDT é um algoritmo recursivo de busca gulosa que procura, sobre um conjunto de atributos, aqueles que "melhor" dividem o conjunto de exemplos em sub-conjuntos. Inicialmente, todos os exemplos são colocados em um único nodo, chamado de raiz. A seguir, um atributo preditivo é escolhido para representar o teste desse nodo e, assim, dividir os exemplos em sub-conjuntos de exemplos. Esse processo se repete recursivamente até que todos os exemplos já estejam classificados ou então até que todos os atributos preditivos já tenham sido utilizados.

Cabe destacar que, por questões didáticas, o algoritmo TDIDT apresentado está considerando apenas atributos categóricos. No entanto, para lidar com atributos numéricos basta um simples ajuste no algoritmo.

2.3 Medidas para selecionar a melhor divisão

O critério de seleção define qual atributo preditivo é utilizado em cada nodo teste. Existem diferentes tipos de critério de seleção, sendo isso uma das variações entre distintos algoritmos de indução de árvores de decisão. Esses critérios são definidos em termos da distribuição de classe dos exemplos antes e depois da divisão (Tan et al., 2005).

A maioria dos algoritmos de indução de árvores de decisão trabalha com funções de divisão univariável, ou seja, cada nodo interno da árvore é dividido de acordo com um único atributo. Nesse caso, o algoritmo tenta encontrar o melhor atributo para realizar essa divisão. Os critérios de seleção para a melhor divisão são baseados em diferentes medidas, tais como impureza, distância e dependência. A maior parte dos algoritmos de indução busca dividir os dados de um nodo de forma a minimizar o grau de impureza dos nodos filhos. Quanto menor o grau de impureza, mais desbalanceada é a distribuição de classe.

Em um determinado nodo, a impureza é nula se todos os seus exemplos pertencerem à mesma classe. Analogamente, o grau de impureza é máximo no nodo se houver o mesmo número de exemplos para cada classe possível.

A seguir, são apresentadas as medidas mais utilizadas para seleção da melhor divisão.

2.3.1 Ganho de Informação

Uma das medidas baseadas em impureza é o Ganho de Informação, o qual usa a entropia como medida de impureza. O algoritmo ID3 (Quinlan, 1986), pioneiro em indução de árvores de decisão, utiliza essa medida. Para determinar o quão boa é uma condição de teste realizada, é necessário comparar o grau de entropia do nodo pai (antes da divisão) com o grau de entropia dos nodos filhos (após a divisão). O atributo que gerar uma maior diferença é escolhido como condição de teste na árvore. O ganho é dado pela Equação 2.1.

$$ganho = entropia(pai) - \sum_{j=1}^{n} \frac{N(v_j)}{N} entropia(v_j) \tag{2.1}$$

em que n é o número de valores do atributo, ou seja, o número de nodos filho, N é o número total de exemplos do nodo pai e $N(v_j)$ é o número de exemplos associados ao nodo filho v_j.

O grau de entropia é definido pela Equação 2.2.

$$entropia(t) = -\sum_{i=1}^{k} p(i|t) log_2 p(i|t) \tag{2.2}$$

em que $p(i|t)$ é a fração dos exemplos pertencentes à classe i, no nodo t, e k é o número de classes.

O critério de ganho seleciona como atributo teste aquele que maximiza o ganho de informação. O grande problema ao se utilizar o ganho de informação é que ele dá preferência a atributos com muitos valores possíveis (número de arestas). Um exemplo claro desse problema ocorreria ao utilizar um atributo totalmente irrelevante (por exemplo, um identificador único). Nesse caso, seria criado um nodo para cada valor possível, e o número de nodos seria igual ao número de identificadores. Cada um desses nodos teria apenas um exemplo, o qual pertence a uma única classe, ou seja, os exemplos seriam totalmente discriminados. Assim, o valor da entropia seria mínima porque, em cada nodo, todos os exemplos (no caso um só) pertencem à mesma classe. Essa divisão geraria um ganho máximo.

2.3.2 Gain Ratio

Para solucionar o problema do ganho de informação, foi proposto em Quinlan (1993) o *Gain Ratio*, que nada mais é que o ganho de informação relativo (ponderado) como

critério de avaliação. O *Gain Ratio* é definido pela Equação 2.3.

$$GainRatio(t) = \frac{ganho}{entropia(t)} \tag{2.3}$$

Pela Equação 2.3, é possível perceber que a função não é definida quando o denominador é igual a zero. Além disso, o *gain ratio* favorece atributos cujo denominador, ou seja, a entropia, possui valor pequeno. Em Quinlan (1988), é sugerido que o *gain ratio* seja realizado em duas etapas. Na primeira etapa, é calculado o ganho de informação para todos os atributos. Após isso, considerar apenas aqueles atributos que obtiveram um ganho de informação acima da média, e então escolher aquele que apresentar o melhor *gain ratio*. Dessa forma, Quinlan mostrou que o *gain ratio* supera o ganho de informação tanto em termos de acurácia como em termos de complexidade das árvores de decisão geradas.

2.3.3 Gini

Outra medida bastante conhecida é o *Gini*, utilizado no algoritmo CART (Breiman et al., 1984), que é baseado na medida de impureza chamada $gini_{index}$. Para um problema de k classes, o $gini_{index}$ é definido pela Equação 2.4.

$$gini_{index}(t) = 1 - \sum_{i=1}^{k} p(i|t)^2 \tag{2.4}$$

Assim como no cálculo do ganho de informação, basta calcular a diferença entre o $gini_{index}$ antes e após a divisão. Essa diferença, *Gini*, é representada pela Equação 2.5.

$$Gini = gini_{index}(pai) - \sum_{j=1}^{n} \frac{N(v_j)}{N} gini_{index}(v_j) \tag{2.5}$$

Assim, é selecionado o atributo que gerar um maior valor para *Gini*.

2.3.4 Outras medidas

As medidas apresentadas são as mais conhecidas e utilizadas na maioria dos trabalhos encontrados. Entretanto, a literatura não se limita a essas medidas. Em Rokach & Maimon (2008) podem ser encontradas outras medidas, tais como: *Distance Measure*, *Binary Criteria*, *Twoing Criterion*, *Orthogonal Criterion*, *Kolmogorov-Smirnov Criterion* e AUC *Splitting Criteria*.

2.4 Representação dos nodos

A forma de representação dos nodos pode influenciar de maneira decisiva no desempenho das árvores de decisão induzidas.

Um atributo é uma propriedade ou característica de um exemplo (Tan et al., 2005). Basicamente, existem pelo menos dois tipos de atributos: categórico e contínuo, também chamado de numérico. O atributo é categórico quando não existe ordem para seus valores (por exemplo, sexo: masculino e feminino). O atributo é contínuo quando existe uma ordem linear nos seus valores (por exemplo, $altura \in \Re$, um número real) (Monard & Baranauskas, 2003).

Dependendo do tipo de atributo, existem diferentes tipos de representação dos nodos para o particionamento dos dados. A seguir, são apresentadas algumas formas de representação considerando atributos categóricos e contínuos.

2.4.1 Atributos categóricos

Um ramo para cada valor do atributo

É a partição mais comum, na qual é criada uma aresta para cada valor do atributo usado como condição de teste. Embora esse tipo de partição permita extrair do atributo todo o seu conteúdo informativo, possui a desvantagem de tornar a árvore de decisão mais complexa. O algoritmo C4.5 (Quinlan, 1993) utiliza esse tipo de divisão para atributos categóricos.

A solução de Hunt

É a partição apresentada em E. Hunt & Stone (1966), também utilizada pelo algoritmo ID3, e sugere uma partição binária. Nesse caso, um dos valores é atribuído a uma das arestas e todos os outros valores à outra aresta. A desvantagem desse tipo de partição é não aproveitar todo o poder de discriminação do atributo em questão.

Atributos ordinais

Um atributo é ordenado quando há uma relação de ordem entre os seus possíveis valores. Por exemplo, tem-se um atributo *altura* que, se discretizado, pode possuir os valores *baixa*, *média* e *alta*. Com atributos desse tipo, é possível realizar uma partição binária do tipo $altura < média$, em que todos os exemplos cujo atributo *altura* têm valor *baixa* seguem por uma aresta e os outros seguem por outra aresta. Esse tipo de partição pode ser utilizado no algoritmo CART (Breiman et al., 1984).

Agrupamento de valores em dois conjuntos

De acordo com Breiman et al. (1984), a divisão binária também pode ser realizada de uma forma mais complexa, na qual cada um dos dois sub-conjuntos pode ser formado por exemplos com mais de um valor para o atributo utilizado como condição de teste. O grande problema desse tipo de partição é o elevado custo computacional para encontrar

a melhor divisão, pois o número de combinações possíveis é $2^{n-1} - 1$, onde n é o número de valores possíveis para o atributo em questão.

Agrupamento de valores em vários conjuntos

Visando permitir o agrupamento de valores em vários conjuntos com uma complexidade de cálculo razoável, o algoritmo C4.5 (Quinlan, 1993) permite encontrar uma solução de boa qualidade. Para isso, inicia criando uma aresta para cada valor do atributo em questão. Após, são testadas todas as combinações possíveis de dois valores e, caso nenhuma dessas combinações produza um ganho maior que a divisão anterior, o processo é interrompido e a divisão anterior é adotada como divisão final. Senão, é repetido o processo tendo como base a melhor das soluções anteriores. Nota-se que não se pode garantir que a divisão encontrada é a melhor possível, pois é verificado se houve melhoria apenas um passo à frente. Esse é o custo de se produzir um algoritmo mais simples.

2.4.2 Atributos contínuos

Os atributos contínuos permitem uma maior variedade de testes e, conseqüentemente, isso implica uma maior complexidade de cálculo. Segundo Fonseca (1994), alguns dos testes mais usados para partição de atributos contínuos são: testes simples ou pesquisa exaustiva, testes múltiplos (segmentação global e segmentação ao nível do nodo) e combinação linear de características.

O teste simples, também conhecido como pesquisa exaustiva, é o mais utilizado. Um dos algoritmos que o utiliza é o C4.5, e a divisão é sempre binária. Supondo um atributo contínuo x a ser utilizado como nodo teste, mesmo que seu domínio seja infinito, o número de exemplos num conjunto de treinamento T é finito e, portanto, o número de valores diferentes para esse atributo também é finito. Assim, os exemplos do conjunto T são ordenados de acordo com seus valores para o atributo x. Supondo que os diferentes valores de x sejam, em ordem crescente, $\{v_1, v_2, ..., v_m\}$, T é dividido em duas partes, são elas: T_1, cujos exemplos possuem valores $\{v_1, v_2, ..., v_i\}$ e T_2, com valores $\{v_{i+1}, v_{i+2}, ..., v_m\}$ para o atributo x. Para cada v_i, $i = 1, ..., (m-1)$, é calculado o ganho (independente do critério utilizado) para a respectiva divisão. Após avaliar todas as divisões possíveis, é escolhida aquela que fornecer o maior ganho. Por fim, é necessário definir o valor que será usado como limiar (valor usado para dividir os exemplos no nodo). Tendo posse do v_i que produziu o melhor ganho, o valor mais utilizado como limiar é $\frac{(v_i+v_{i+1})}{2}$, pois assim espera-se que a árvore resultante apresente melhores resultados para exemplos que não participaram do conjunto de treinamento.

2.5 Tratamento de valores desconhecidos

Para qualquer tipo de atributo, podem figurar valores desconhecidos, ou seja, exemplos com a ausência de um valor para um determinado atributo. Para caracterizar os valores desconhecidos, geralmente é utilizado um ponto de interrogação "?". Cabe destacar que um valor desconhecido é bem diferente do valor zero (atributos contínuos) ou de cadeias de caracteres vazias (atributos categóricos).

Para lidar com esses valores desconhecidos, a literatura dispõe de uma série de alternativas. A seguir, são apresentadas as que mais se destacam.

Substituição por valor

Uma das formas mais utilizadas para tratamento de valores desconhecidos é atribuir um valor para eles. Geralmente, utiliza-se a média e a moda do conjunto de treinamento para tratar valores que representam atributos numéricos e categóricos, respectivamente.

Tratamento por exclusão

Os valores desconhecidos em exemplos do conjunto de treinamento têm um tratamento bastante simples em que, para o cálculo da melhor divisão em cada nodo, são utilizados somente os exemplos que possuem valores definidos para o atributo em questão. Do ponto de vista da classificação de novos exemplos, no caso de um dado nodo testar um atributo cujo valor é desconhecido, será atribuído a esse exemplo o ramo mais freqüente desse nodo.

Introdução do valor desconhecido

Uma outra solução para o problema de valores desconhecidos é considerá-los como um novo valor, ou seja, criar uma aresta extra rotulada como "desconhecido". Assim, durante o treinamento todos os exemplos nos quais o valor do atributo testado é desconhecido são atribuídos a essa aresta, e o restante do processo segue igual à forma padrão.

Partições alternativas

Apresentado no algoritmo CART (Breiman et al., 1984), este método permite obter atributos de teste alternativos, em cada nodo, no caso do valor ser desconhecido para o atributo original. Segundo Fonseca (1994), uma partição alternativa é aquela que, com base em outro atributo, tenta "imitar" a partição normal, tendo como objetivo, na falta do atributo testado normalmente, efetuar uma divisão dos exemplos tão similar quanto possível. A semelhança entre as duas partições deverá ser não só na porcentagem de exemplos de cada classe que é destinada a cada sub-partição, mas também nos próprios exemplos que são colocados em cada uma delas. Dessa forma, será possível minimizar as conseqüências da classificação com base num atributo alternativo ao original.

2.6 Poda

Quando árvores de decisão são construídas, muitas das arestas ou sub-árvores podem refletir ruídos ou erros. Isso acarreta em um problema conhecido como *overfitting*, que significa um aprendizado muito específico do conjunto de treinamento, não permitindo ao modelo generalizar. Para detectar e excluir essas arestas e sub-árvores, são utilizados métodos de poda (*pruning*) da árvore, cujo objetivo é melhorar a taxa de acerto do modelo para novos exemplos, os quais não foram utilizados no conjunto de treinamento (Han, 2001). Conseqüentemente, a árvore podada se torna mais simples, facilitando a sua interpretabilidade por parte do usuário. Assim como o método de seleção, o método de poda também varia de acordo com os diferentes algoritmos de indução de árvores de decisão.

Existem diversas formas de realizar poda em uma árvore de decisão, e elas são classificadas como pré-poda ou pós-poda. O método pré-poda é realizado durante o processo de construção da árvore, em que o processo pode simplesmente parar de dividir o conjunto de elementos e transformar o nodo corrente em um nodo folha da árvore. O ganho de informação, por exemplo, pode ser utilizado como critério de poda. Caso todas as divisões possíveis utilizando um atributo A gerem ganhos menores que um limiar pré-estabelecido, então esse nodo vira folha, representando a classe mais freqüente no conjunto de exemplos. A dificuldade é encontrar um valor adequado para esse limiar, visto que um valor muito alto pode gerar uma árvore super-simplificada, enquanto que um valor muito baixo pode simplificar bem pouco a árvore. Já o pós-poda é realizado após a construção da árvore de decisão, removendo ramos completos, em que tudo que está abaixo de um nodo interno é excluído e esse nodo é transformado em folha, representando a classe mais freqüente no ramo. Para cada nodo interno da árvore, o algoritmo calcula a taxa de erro caso a sub-árvore abaixo desse nodo seja podada. Em seguida, é calculada a taxa de erro caso não haja a poda. Se a diferença entre essas duas taxas de erro for menor que um limiar pré-estabelecido, a árvore é podada, caso contrário não ocorre a poda. Esse processo se repete progressivamente, gerando um conjunto de árvores podadas. Por fim, para cada uma delas é calculada a acurácia na classificação de um conjunto de dados independente dos dados de treinamento (por exemplo, o conjunto de validação), e a árvore que obtiver a melhor acurácia será a escolhida.

Embora a poda seja um método bastante utilizado e eficaz na solução do problema de *overfitting*, deve-se ter cuidado para não podar demais a árvore. Quando isso ocorre, tem-se o problema conhecido como *underfitting*, em que o modelo de classificação não aprendeu o suficiente sobre os dados de treinamento.

Dentre os métodos de poda existentes, destacam-se: *Cost Complexity Pruning*, *Reduced Error Pruning*, *Minimum Error Pruning* (MEP), *Pessimistic Pruning*, *Error-Based Pruning* (EBP), *Minimum Description Length* (MDL) *Pruning*, *Mininum Message Length* (MML) *Pruning*, *Critical Value Pruning* (CVP), OPT e OPT-2. Maiores detalhes sobre

esses métodos podem ser encontrados em Rokach & Maimon (2008).

2.7 Avaliação de classificadores

A avaliação do desempenho de uma árvore de decisão, e de modelos de classificação em geral, é um aspecto fundamental em aprendizado de máquina. Como destacado anteriormente, um algoritmo de indução de árvores de decisão recebe como entrada um conjunto de treinamento para então gerar as árvores de decisão, as quais utilizam o conhecimento adquirido para prever os valores das classes de exemplos até então desconhecidos. Tanto as árvores de decisão geradas como o algoritmo de indução podem ser avaliados de acordo com algum critério. A avaliação é muito importante não somente para mensurar a qualidade das árvores de decisão, mas também para guiar o processo iterativo de KDD como um todo (Rokach & Maimon, 2008).

Há vários métodos para estimar o erro de generalização de um modelo durante o treinamento. A estimativa do erro ajuda o algoritmo de aprendizado a selecionar um modelo que não seja suscetível a *overfitting*, ou seja, quando o algoritmo de aprendizagem super-aprende os dados, gerando um modelo que não generaliza bem. Para que essa avaliação seja eficaz, é preciso cuidado na escolha das amostras que serão utilizadas tanto para a indução (treinamento) como para a avaliação (teste) dos classificadores.

O método de re-substituição utiliza o mesmo conjunto utilizado no treinamento para estimar as métricas de avaliação. Essa característica implica em resultados otimistas, visto que não garante que o modelo generalize bem quando novos exemplos são apresentados a ele (Monard & Baranauskas, 2003). sEm contraste, existem métodos de reamostragem que defendem a idéia de que os conjuntos de treinamento e teste devem ser disjuntos. Os métodos *hold-out*, *ramdom subsampling*, validação cruzada com k partições, *leave-one-out* e *bootstrap* são alguns exemplos mais utilizados para reamostragem.

O método mais simples de reamostragem é o *hold-out*, em que é definida uma porcentagem fixa para a divisão do conjunto de dados em treinamento e teste. Geralmente, é atribuído 2/3 (dois-terços) dos exemplos para o treinamento e 1/3 (um-terço) para o conjunto de teste. Esse método funciona bem quando há uma quantidade significativa de dados.

No entanto, quando a quantidade de dados não é suficientemente grande, outros métodos são sugeridos, tais como *random subsampling* e validação cruzada com k partições. Em *random subsampling*, os dados são aleatoriamente divididos várias vezes em conjuntos disjuntos de treinamento e teste. Os erros, bem como outras medidas convenientes, são obtidos para cada partição e então uma média é calculada. Na validação cruzada com k partições, os dados são divididos em k sub-conjuntos mutuamente exclusivos de tamanhos aproximadamente iguais. O algoritmo de indução é treinado e testado k vezes, sendo que em cada vez é testado com uma das partições e treinado com o restante. Ao final, é calculada a média dos k valores. De acordo com Tan et al. (2005),

o método de validação cruzada com k partições, com $k = 10$, é um bom estimador.

A estratificação é um processo geralmente aplicado em métodos de reamostragem, especialmente em *random subsampling* e validação cruzada com k partições. Esse processo garante que, em cada partição do conjunto (treinamento e teste), seja mantida a mesma distribuição de classes do conjunto total. A estratificação tende a reduzir a variância da estimativa de erro especialmente para conjuntos com muitas classes (Rokach & Maimon, 2008).

O método *leave-one-out* é um caso particular de validação cruzada com $k = n$, em que n é o número total de exemplos. Nesse caso, cada exemplo é utilizado uma vez para teste, sendo o restante $(n - 1)$ utilizado como treinamento. A grande desvantagem desse método é que ele é custoso computacionalmente e, portanto, é aconselhável apenas para pequenos conjuntos de dados.

O método *bootstrap* gera os conjuntos de treinamento e teste a partir de seleção aleatória dos exemplos do conjunto total. Essa reamostragem é realizada com reposição, ou seja, o mesmo exemplo pode figurar mais de uma vez no mesmo sub-conjunto (treinamento ou teste) e até mesmo nos dois sub-conjuntos. Esse processo é, geralmente, repetido várias vezes, gerando múltiplos conjuntos de treinamento e teste.

Uma vez escolhido o método de particionamento do conjunto de dados, o próximo passo é definir a medida de desempenho que será utilizada para avaliar o classificador. É possível também que mais de uma medida sejam utilizadas em paralelo. Essas medidas podem ser facilmente calculadas a partir de uma matriz que ilustra a qual classe cada exemplo pertence e também a qual classe ele foi classificado pelo classificador. Essa matriz é chamada de Matriz de Confusão, e um exemplo pode ser visto na Tabela 2.2, considerando um problema de duas classes.

Tabela 2.2: Exemplo de uma matriz de confusão para um problema de duas classes.

	Classe predita	
Classe real	Positiva	Negativa
Positiva	VP	FN
Negativa	FP	VN

Em problemas de classificação com apenas 2 (duas) classes, uma delas pode ser considerada como positiva e a outra como negativa. Essa idéia pode ser facilmente estendida para problemas com mais de 2 (duas) classes, em que cada uma delas é considerada como positiva e as outras como negativa. Nesse caso, haveriam c matrizes de confusão, sendo c igual ao número de classes. Considerando uma matriz de confusão (Tabela 2.2), os conceitos de VP, FP, FN e VN são descritos como segue:

Verdadeiros Positivos (VP): são os exemplos que pertencem à classe positiva e foram corretamente classificados como tal pelo classificador.

Falsos Positivos (FP): são os exemplos que pertencem à classe negativa, porém classificados incorretamente como positivas pelo classificador.

Verdadeiros Negativos (VN): são os exemplos que pertencem à classe negativa e foram corretamente classificados como negativos pelo classificador.

Falsos Negativos (FN): são os exemplos que pertencem à classe positiva, contudo classificados como negativos pelo classificador.

A partir dos valores de VP, FP, VN e FN, é possível calcular facilmente várias medidas de desempenho do classificador. A seguir, são apresentadas algumas medidas bastante utilizadas na literatura (Rokach & Maimon, 2008).

Acurácia

A acurácia estima a probabilidade do classificador acertar suas previsões. Ela é definida pela Equação 2.6.

$$Acurácia = \frac{|VP| + |VN|}{|VP| + |VN| + |FP| + |FN|} \tag{2.6}$$

Similarmente, o desempenho de um modelo de classificação pode ser expresso em termos de sua taxa de erro, que é o complemento da acurácia, e é definido pela Equação 2.7. Ela estima a probabilidade do classificador errar suas previsões.

$$Erro = \frac{|FP| + |FN|}{|VP| + |VN| + |FP| + |FN|} \tag{2.7}$$

Essas medidas são as mais utilizadas em problemas de classificação de dados. No entanto, dependendo do domínio de aplicação, pode não ser uma boa métrica, principalmente quando há desbalanceamento das classes, ou seja, quando há muitos exemplos de determinadas classes e poucos exemplos de outras classes.

Recall e Especificidade

De acordo com Rokach & Maimon (2008), a acurácia não é uma medida suficiente para avaliar desempenho de classificadores quando há desbalanceamento das classes. Quando o conjunto de dados contém muitos exemplos de uma mesma classe e poucos das outras, é fácil selecionar sempre a classe majoritária e obter uma boa taxa de acurácia. Portanto, nesses casos, Han (2001) sugere que medidas de sensibilidade e especificidade sejam utilizadas como alternativa.

A medida de sensibilidade também é conhecida como *recall*, Equação 2.8, e mede o quão bem um classificador classifica os exemplos positivos.

$$Recall = \frac{|VP|}{|VP| + |FN|} \tag{2.8}$$

A medida de especificidade na Equação 2.9 estima quão bem um classificador classifica como negativos os exemplos cujas classes são realmente negativas.

$$Especificidade = \frac{|VN|}{|VN| + |FP|} \tag{2.9}$$

Precisão

Outra medida de desempenho bastante conhecida é a precisão, a qual mensura quantos dos exemplos classificados como positivos são realmente pertencentes à classe positiva. Essa medida é bastante utilizada para detecção de *spams*, pois é preferível que um *spam* apareça na caixa de entrada do que o contrário, ou seja, um email importante vá parar na caixa de *spam*. A Equação 2.10 define o cálculo da precisão.

$$Precisão = \frac{|VP|}{|VP| + |FP|} \tag{2.10}$$

Medida-F

A medida-F é um *trade-off* entre as medidas de precisão e *recall*, uma vez que ao tentar melhorar uma dessas medidas, a outra é comprometida. A medida-F é calculada de acordo com a Equação 2.11. O valor padrão de β é 1.

$$Medida - F = \frac{(\beta^2 + 1) * Precisão * Recall}{\beta^2 * Precisão + Recall} \tag{2.11}$$

Outras medidas

Existem também outras abordagens para avaliação de classificadores. Alguns exemplos são: *ROC Curves*, *Hit Rate Curve*, *Qrecall*, *Lift Curve*, *Pearson Correlation Coefficient*, *Area Under Curve* (AUC), *Average Hit Rate*, *Average Qrecall* e *Potential Extract Measure* (PEM). Maiores detalhes sobre esses métodos podem ser encontrados em Rokach & Maimon (2008).

2.8 Algoritmos de indução de árvores de decisão

Nesta seção, são apresentados alguns algoritmos para indução de árvores de decisão. Além dos algoritmos mais conhecidos, tais como ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993) e CART (Breiman et al., 1984), foram selecionados na literatura outros artigos que propõem novos algoritmos para indução de árvores de decisão. Alguns deles, inclusive, não são baseados no algoritmo básico TDIDT. Os algoritmos selecionados são: NBTree (Kohavi, 1996), ADTree (Freund & Mason, 1999), LMT (Landwehr et al., 2005) e BFTree (Shi, 2007).

2.8.1 ID3

O ID3 (Quinlan, 1986) é o algoritmo pioneiro em indução de árvores de decisão. Ele é um algoritmo recursivo e baseado em busca gulosa, procurando, sobre um conjunto de atributos, aqueles que "melhor" dividem os exemplos, gerando sub-árvores.

A principal limitação do ID3 é que ele só lida com atributos categóricos, não sendo possível apresentar a ele conjuntos de dados com atributos contínuos. Nesse caso, os atributos contínuos devem ser previamente discretizados. Além dessa limitação, o ID3 também não apresenta nenhuma forma para tratar valores desconhecidos, ou seja, todos os exemplos do conjunto de treinamento devem ter valores conhecidos para todos os seus atributos. É de conhecimento geral que, na prática, os conjuntos de dados possuem muitos valores desconhecidos. Logo, para se utilizar o ID3, é necessário gastar um bom tempo com pré-processamento dos dados.

O ID3 utiliza o ganho de informação para selecionar a melhor divisão. No entanto, esse critério não considera o número de divisões (número de arestas), e isso pode acarretar em árvores mais complexas. Somado a isso, o ID3 também não apresentou nenhum método de pós-poda, o que poderia evitar esse problema de árvores mais complexas.

2.8.2 C4.5

O algoritmo C4.5 (Quinlan, 1993) é uma evolução do ID3 (Quinlan, 1986). As principais contribuições em relação ao ID3 são:

- lida tanto com atributos categóricos como também atributos contínuos. Para lidar com atributos contínuos, o algoritmo C4.5 define um limiar e então divide os exemplos de forma binária: aqueles cujo valor do atributo são maiores que o limiar e aqueles cujo valor do atributo são menores ou iguais ao limiar;

- trata valores desconhecidos. O algoritmo C4.5 permite que os valores desconhecidos para um determinado atributo sejam representados como '?', e o algoritmo trata esses valores de forma especial. Esses valores não são utilizados nos cálculos de ganho e de entropia;

- utiliza a medida chamada *gain ratio* para selecionar o atributo que melhor divide os exemplos. Essa medida se mostrou superior ao ganho de informação, gerando árvores mais precisas e menos complexas;

- lida com problemas em que os atributos possuem custos diferenciados;

- apresenta um método de pós-poda das árvores geradas. O algoritmo C4.5 faz uma busca na árvore, de baixo para cima, e transforma em nodos folha aqueles ramos que não apresentam nenhum ganho significativo.

O C4.5 é o algoritmo mais utilizado na literatura, justamente por ter mostrado ótimos resultados em problemas de classificação. A ferramenta de mineração de dados Weka (Witten & Frank, 1999) disponibiliza a implementação do algoritmo C4.5, porém o mesmo é chamado de J48 nessa ferramenta.

2.8.3 CART

O algoritmo CART (*Classification and Regression Trees*) foi proposto em Breiman et al. (1984) e consiste de uma técnica não paramétrica que induz tanto árvores de classificação quanto árvores de regressão, dependendo se o atributo classe é categórico (classificação) ou contínuo (regressão). Uma das principais virtudes do CART é a grande capacidade de pesquisa de relações entre os dados, mesmo quando elas não são evidentes, bem como a produção de resultados sob a forma de árvores de decisão de grande simplicidade e legibilidade (Fonseca, 1994).

As árvores geradas pelo algoritmo CART são sempre binárias, as quais podem ser percorridas da sua raiz até as folhas respondendo apenas a questões simples do tipo "sim" ou "não". Os nodos que correspondem a atributos categóricos são representados por agrupamento de valores em dois conjuntos. Da mesma forma que no algoritmo C4.5, o CART utiliza a técnica de pesquisa exaustiva para definir os limiares a serem utilizados nos nodos para dividir os atributos contínuos. Adicionalmente, o CART dispõe de um tratamento especial para atributos ordenados e também permite a utilização de combinações lineares entre atributos (agrupamento de valores em vários conjuntos).

Diferentemente das abordagens adotadas por outros algoritmos, os quais utilizam pré-poda, o CART expande a árvore exaustivamente, realizando pós-poda por meio da redução do fator custo-complexidade (Breiman et al., 1984). Segundo os autores, a técnica de poda utilizada é extremamente eficiente e produz árvores mais simples, precisas e com ótima capacidade de generalização.

2.8.4 NBTree

Em Kohavi (1996), é proposto o algoritmo NBTree, o qual induz um modelo de classificação híbrido, uma combinação de árvores de decisão com *naive-Bayes* (Rish, 2001). Os nodos contêm divisões considerando um único atributo, como nas árvores de decisão regulares, porém os nodos folha contêm classificadores *naive Bayes*. Essa abordagem mantém a boa interpretabilidade das duas técnicas e, ao mesmo tempo, fornece classificadores que superam ambas as técnicas, quando usadas independentemente, principalmente para classificar grandes conjuntos de dados.

Quando um atributo categórico é escolhido como nodo da árvore, é criada uma aresta para cada possível valor do atributo. No caso do atributo ser numérico, é realizada uma divisão binária de acordo com um limiar (valor que divide os exemplos). Esse valor é escolhido usando a técnica padrão para minimizar a entropia, como em algoritmos de

indução de árvores de decisão. A utilidade de um nodo é calculada ao discretizar os dados e computar a acurácia por validação cruzada com 5 partições usando *naive Bayes* no nodo. A utilidade de uma divisão (*split*) é a soma ponderada das utilidades dos nodos, onde o peso dado a cada nodo é proporcional ao número de exemplos contidos nesse nodo.

Não é informado no artigo como os valores desconhecidos são tratados pelo algoritmo NBTree. Para evitar *overfitting*, foi definido que uma divisão ocorre se houver pelo menos 30 exemplos no nodo e, após a divisão, se o erro relativo for reduzido em mais de 5%.

Os resultados obtidos foram considerados bastante satisfatórios pelo autor. NBTree induziu classificadores altamente precisos e, em muitos casos, superou significativamente ambas as técnicas que os constituem (C4.5 e *naive Bayes*). Uma vez que nenhum classificador superou os demais em todos os domínios, NBTree gerou classificadores com alta acurácia para dados reais e de larga escala. De fato, para três conjuntos de dados com mais de 10000 (dez mil) exemplos, NBTree superou os algoritmos C4.5 e *naive Bayes* em termos de acurácia. No entanto, o tempo de execução foi maior que ambas as outras técnicas, porém a dependência do número de exemplos para o cálculo da complexidade de criar as divisões foi a mesma, indicando que o tempo de execução possui boa escalabilidade (Kohavi, 1996).

2.8.5 ADTree

Yoav Freund e Llew Mason propõem em Freund & Mason (1999) o algoritmo ADTree (*Alternating Decision Tree*), o qual é uma generalização de árvores de decisão, alternadas entre árvores de decisão e *decision stumps*. Também é apresentado um algoritmo de aprendizado especial, o qual é baseado em *boosting* (Quinlan, 1996).

Os autores restringem o algoritmo apenas para problemas de duas classes. As árvores resultantes de ADTree possuem dois tipos de nodos: nodos de decisão e nodos de predição. Os nodos de decisão são idênticos aos encontrados nas árvores padrão, enquanto a cada nodo de predição é associado um número cujo valor é contínuo. Como em outras árvores de decisão, um exemplo percorre a árvore a partir do nodo raiz até uma das folhas. Entretanto, há uma diferença, pois a classificação que é associada com o caminho não é o nome de um nodo folha, mas sim a soma dos valores encontrados em cada nodo de predição visitado durante a trajetória.

O algoritmo de aprendizado do ADTree é bastante semelhante aos bem conhecidos C4.5 e CART, seguindo basicamente o TDIDT, adicionando uma regra de decisão ao encontrar a melhor divisão dos dados na árvore. Entretanto, algumas diferenças importantes são destacadas:

- Nodos de decisão podem ser adicionados em qualquer lugar da árvore, não somente nas folhas.
- Como critério de divisão dos dados, é utilizado um erro ponderado ao adicionar uma regra, ao invés de *Gini* ou ganho de informação.

Tanto para atributos numéricos como categóricos a divisão é binária. No primeiro caso, usa-se o operador de desigualdade, enquanto no segundo caso é utilizado o operador de igualdade. Em ambos os casos, as arestas possuem respostas positiva ou negativa à condição do nodo de decisão. Assim, a árvore de decisão resultante é sempre binária.

Embora o fato das árvores serem binárias possa sugerir que elas sejam maiores e, conseqüentemente, mais complexas, isso não ocorre nas árvores geradas por ADTree devido à sua estrutura diferenciada, pois cada nodo de predição percorrido dá a sua contribuição por meio do valor numérico associado a ele. Isso resulta em árvores menores e mais fáceis de serem interpretadas. Talvez por esse motivo não são encontradas no artigo informações sobre poda, pois ela pode ter sido considerada desnecessária.

O tratamento de valores desconhecidos é realizado pela própria estrutura da árvore. Caso um exemplo chegue num nodo que testa um atributo cujo valor é desconhecido para esse exemplo, o exemplo é classificado do mesmo jeito, pois não necessariamente deve-se chegar a um nodo folha, mas sim avaliar o sinal da soma de todos os nodos de predição visitados.

2.8.6 LMT

Métodos de indução de árvores de decisão e modelos lineares são técnicas conhecidas para tarefas de aprendizado supervisionado, tanto para predição de atributos numéricos como categóricos. Para predição de atributos numéricos, há trabalhos que combinam esses dois esquemas em modelos de árvores, ou seja, árvores que contêm função de regressão linear em suas folhas. Em Landwehr et al. (2005), é apresentado o algoritmo LMT, que adapta essa idéia para predição de atributos categóricos (classificação) usando regressão logística ao invés de regressão linear.

Os autores usam uma estratégia de construção dos modelos de regressão logística que seleciona os atributos relevantes de forma natural, e mostram como essa abordagem pode ser utilizada para induzir modelos de regressão logística nos nodos folhas. Como vantagem, é possível obter a estimativa das probabilidades das classes ao invés de somente uma classe.

A representação dos nodos é feita da mesma forma que no algoritmo C4.5, em que, para atributos numéricos, a divisão é binária e, para atributos categóricos, é criado um ramo para cada valor possível do atributo. O critério de seleção do atributo a compor o nodo é o Ganho de Informação.

Para lidar com valores desconhecidos de um determinado atributo, são calculadas a média, para atributos numéricos, e a moda, para atributos categóricos, dos valores conhecidos do conjunto de treinamento. Feito isso, os valores desconhecidos são preenchidos com os valores calculados. Os mesmos valores calculados são utilizados na classificação de novos exemplos.

Visando evitar *overfitting*, os autores utilizam o mesmo método de poda do algoritmo CART, o qual usa validação cruzada para obter resultados mais estáveis, gerando árvores menores e geralmente com maior acurácia.

As árvores geradas pelo algoritmo LMT foram comparadas às induzidas pelo C4.5. Quanto ao critério de acurácia, não houve significância estatística nas diferenças apresentadas para os conjuntos de dados utilizados. Quanto ao tamanho das árvores, o LMT gerou árvores bem menores que o C4.5 em todos os casos. Embora os autores utilizem isso como justificativa para maior interpretabilidade das árvores LMT, na realidade a estrutura desse tipo de árvore, principalmente pela complexidade das funções logísticas nos nodos filhos, inviabilizam o tamanho da árvore como medida de interpretabilidade dos modelos.

2.8.7 BFTree

Em Shi (2007), é proposto um algoritmo para indução de árvores de decisão binárias baseado na heurística *best-first*. Também são investigados dois novos métodos de poda, os quais determinam um tamanho apropriado para a árvore de decisão ao combinar o crescimento da árvore com a seleção, baseada em validação cruzada, do número de expansões que são realizadas. Um desses métodos segue a abordagem pré-poda enquanto o outro a pós-poda. Eles são chamados de *best-first-based pre-pruning* e *best-first-based post-pruning*, respectivamente. Ambos os métodos usam o mesmo mecanismo, portanto é possível compará-los. *Best-first-based pre-pruning* termina a divisão quando futuras divisões aumentam o erro da validação cruzada, enquanto o *best-first-based post-pruning*, a partir de uma árvore de decisão completa, descarta algumas expansões baseado no erro da validação cruzada. Assim, foi possível compará-los com outro método de poda baseado na validação cruzada, o *minimal cost-complexity pruning*, utilizado no algoritmo CART (Breiman et al., 1984). Os resultados mostraram que os dois métodos são competitivos com o apresentado no CART quanto ao critério chamado *one standard error rule*. Entretanto, *minimal cost-complexity pruning* é preferível aos outros métodos porque gera árvores de decisão mais compactas e com acurácias equivalentes.

O algoritmo BFTree segue a abordagem TDIDT. Em cada nodo interno da árvore é utilizado apenas um atributo como teste, e a escolha desse atributo pode ser realizada tanto pelo critério *Gini* como pelo critério de Ganho de Informação. Tanto para atributos numéricos como categóricos a divisão é binária, caracterizando também como binárias as árvores induzidas. A divisão dos atributos numéricos seguem a mesma técnica apresentada nos algoritmos C4.5 e CART. Quando o atributo é categórico, o critério utilizado é diferente para problemas de duas classes e problemas de múltiplas classes. Para o primeiro caso, a divisão é realizada por meio da mesma técnica apresentada no algoritmo CART, em que o método de busca reduz o espaço de busca de $2^{n-1} - 1$ para $n - 1$ (Breiman et al., 1984). Já para problemas de múltiplas classes, é utilizado um método proposto em Hong et al. (1999), em que é introduzido um método heurístico de busca que atinge um compromisso entre redução da impureza e velocidade da busca.

Para o tratamento de valores desconhecidos, o autor destaca que uma alternativa bastante simples para tratar o problema é considerar o valor desconhecido como um outro

possível valor do atributo, ou seja, criar uma aresta associada a esse valor. Nesse caso, um valor desconhecido poderia ser utilizado como um regra de decisão. Entretanto, o autor considera que, geralmente, valores desconhecidos não têm significado especial. Assim, em Shi (2007) foi usada uma solução apresentada em Witten & Frank (1999), em que os exemplos são divididos em partes ponderadas de acordo com um esquema numérico.

2.9 Considerações finais

Este capítulo apresentou uma revisão sobre os principais conceitos necessários para se construir um algoritmo de indução de árvores de decisão. Além disso, foram apresentados alguns trabalhos relacionados que propõem diferentes formas de indução de árvores de decisão. No próximo capítulo, será apresentada uma outra abordagem que pode ser utilizada para indução de árvores de decisão, que são os algoritmos evolutivos. A adoção dessa abordagem é bastante interessante para o desenvolvimento deste trabalho, e as vantagens de sua utilização serão destacadas.

Capítulo

3

Algoritmos Evolutivos

No capítulo anterior, foi realizada uma revisão dos conceitos sobre árvores de decisão bem como dos algoritmos tradicionais de indução. Além disso, foram apresentados os principais problemas oriundos dos algoritmos tradicionais de indução de árvores de decisão. Visando solucionar tais problemas, neste capítulo é apresentada uma revisão sobre algoritmos evolutivos, especialmente trabalhos que propõem a utilização de algoritmos genéticos e programação genética para indução de árvores de decisão.

3.1 Introdução

Algoritmos Evolutivos (AEs) são métodos estocásticos de busca baseados nos mecanismos de seleção natural e genética natural de Darwin (Darwin, 1859), que defende a idéia de que os indivíduos mais aptos sobrevivem. Esses métodos são capazes de evoluir soluções de um problema do mundo real, sendo considerados uma abordagem muito atrativa na busca de soluções sub-ótimas em problemas de otimização. Mesmo sendo aleatórios, os algoritmos evolutivos utilizam informações históricas (soluções anteriores) para especular os novos pontos de busca no espaço de soluções (Goldberg, 1989; Mitchell, 1998). As etapas envolvidas na execução de um algoritmo evolutivo padrão podem ser visualizadas na Figura 3.1.

Na natureza, indivíduos estão constantemente evoluindo, se adaptando cada vez mais ao ambiente em que vivem. Em AEs, cada indivíduo corresponde a uma solução candidata para um determinado problema. Cada indivíduo é avaliado por uma função de *fitness*, a qual mede o quanto o indivíduo está adaptado ao ambiente, ou seja, a qualidade da solução candidata. A cada geração (iteração), os melhores indivíduos têm maior probabilidade de serem selecionados para se reproduzirem. Os indivíduos selecionados passam por

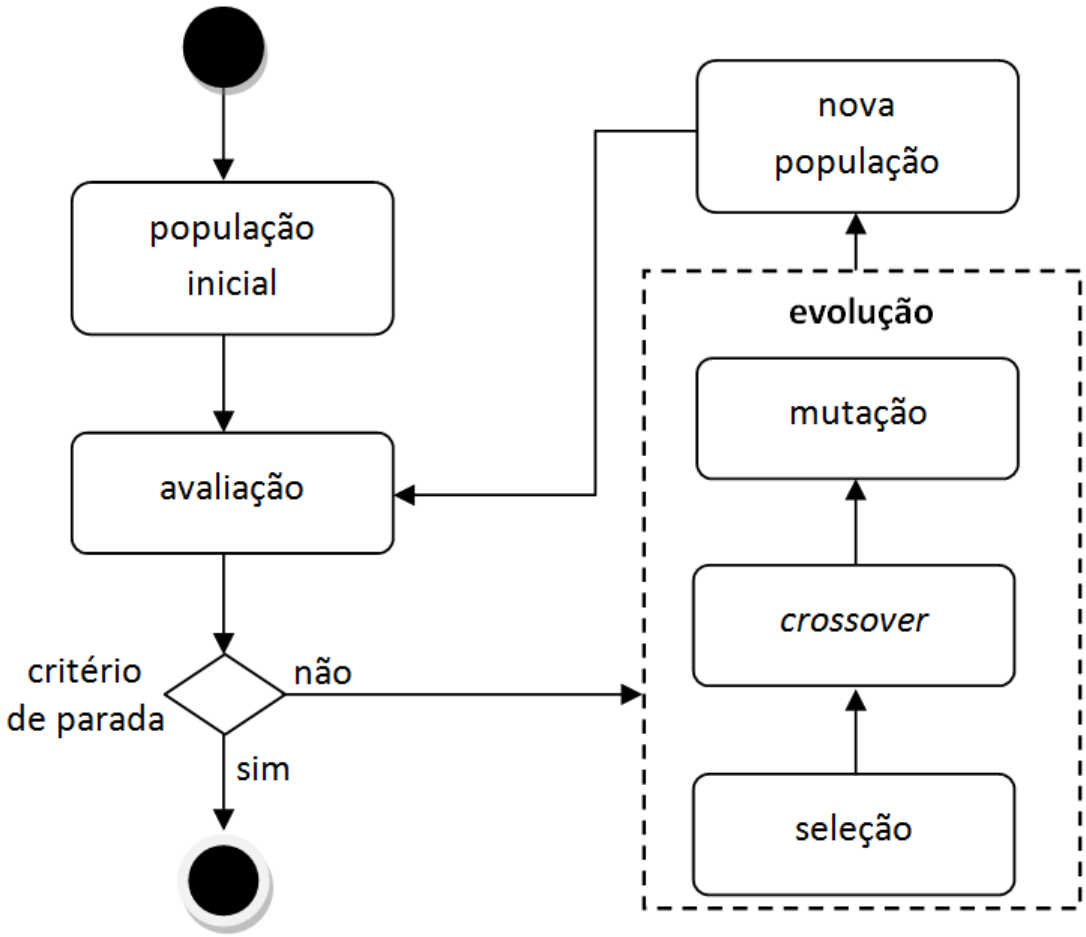


Figura 3.1: Diagrama de atividades utilizado para representar o princípio básico de funcionamento de um algoritmo evolutivo.

operações inspiradas na genética natural, como *crossover* (em que partes do material genético de dois indivíduos são trocadas) e mutação (em que parte do material genético de um indivíduo é substituída por material genético gerado aleatoriamente), produzindo novos indivíduos (filhos) que irão substituir os atuais (pais) e formar uma nova geração da população. Esse processo é iterativamente repetido até que um critério de parada seja satisfeito, como por exemplo até que um número máximo de gerações tenha sido atingido ou até que uma solução satisfatória tenha sido encontrada.

Existem vários tipos de AEs e, de acordo com Freitas (2008), os dois mais utilizados em mineração de dados são Algoritmos Genéticos (AG) e Programação Genética (PG). Em um alto nível de abstração, tanto AG como PG podem ser descritos pelo diagrama ilustrado na Figura 3.1. Embora compartilhem o mesmo diagrama, há algumas diferenças importantes entre eles. A mais importante envolve a natureza fundamental da solução que eles representam. Mais precisamente, em AGs cada indivíduo representa uma solução para uma instância particular de um problema a ser solucionado. Por outro lado, em PG uma solução candidata deve representar uma solução genérica (um programa ou um algoritmo) para o tipo de problema a ser solucionado, no sentido de que o programa deve ser genérico o suficiente para ser aplicado a qualquer instância do tipo de problema

em questão. Por esse motivo, a representação dos indivíduos de um AG tende a ser mais simples que a representação dos indivíduos de uma PG. Geralmente, em AGs os indivíduos são representados por *strings* de tamanho fixo, enquanto que em PGs são representados por árvores de tamanho variável.

AEs podem ser utilizados para solucionar muitos tipos de problemas, e para isso é necessário especificar cuidadosamente qual tipo de solução candidata um indivíduo representa e como a qualidade dessa solução pode ser avaliada. A forma de representação, que define o espaço de busca, e a função de *fitness*, bem como outras questões de algoritmos evolutivos são tratadas nas seções seguintes.

3.2 Representação

Segundo Eiben & Smith (2003), nesta etapa é onde deve ser criado um *link* entre o "mundo real" e o "mundo evolutivo", ou seja, definir a ponte entre o problema original e o espaço de busca a ser explorado pelo algoritmo evolutivo. Objetos formando possíveis soluções dentro do contexto do problema original são chamados de fenótipos, enquanto que sua codificação, ou seja, os indivíduos do AE, são chamados de genótipos. É importante destacar que o espaço de busca dos fenótipos pode ser bem diferente do espaço de busca dos genótipos. Uma solução real (fenótipo) para o problema original deve ser obtida por meio da decodificação do melhor genótipo encontrado pelo algoritmo evolutivo após a sua execução. Definir uma forma de representação adequada é uma tarefa bastante difícil e pode ser decisiva para o sucesso do algoritmo evolutivo. Geralmente, uma boa codificação vem com bastante prática e bom conhecimento do domínio de aplicação (Mitchell, 1998).

A forma de representação é característica de cada problema em questão. A representação binária é a mais conhecida, sendo muito usada em AGs. As formas mais comuns de representação são apresentadas a seguir.

3.2.1 Representação binária

A codificação binária é uma forma de codificação bastante utilizada na literatura por diversas razões. A principal delas é o fato de diversas teorias de convergência dos AGs serem baseadas na pressuposição de que a forma de representação é essa. Além disso, operações como *crossover* e mutação são mais fáceis de serem aplicadas quando se utiliza esse tipo de codificação. No entanto, a desvantagem é quando o problema de otimização é muito complexo, sendo necessário *strings* muito grandes. A Figura 3.2 ilustra um exemplo de indivíduo nesse formato, em que pode ser observado que o *string* é composto por *bits* (0's e 1's).

0	1	1	0	0	0	1	0	0

Figura 3.2: Indivíduo codificado como um *string* binário.

3.2.2 Representação inteira

Em muitas situações, a representação binária não é a mais adequada para codificar a solução de um problema. Um exemplo típico seria o problema de encontrar os valores ótimos para um conjunto de variáveis cujos domínios sejam os inteiros. Seria muito mais natural representar a solução como um *string* de números inteiros, sendo o tamanho do *string* igual ao número de variáveis e cada posição (gene do indivíduo) representando o valor da variável em questão. A Figura 3.3 ilustra um possível indivíduo para a solução do problema exemplificado.

7	1	3	9	1	2	1	5	8

Figura 3.3: Indivíduo codificado como um *string* de inteiros.

3.2.3 Representação por valores reais

Pela mesma razão da representação anterior, em diversas aplicações é mais natural utilizar valores reais para formar os indivíduos. Um exemplo bem típico deste tipo de codificação é quando se quer otimizar os pesos de uma rede neural. Este tipo de representação pode ser visualizado na Figura 3.4.

0.2	1.5	0.7	3.7	0.5	0.9	2.5	5.5	0.1

Figura 3.4: Indivíduo codificado como um *string* de valores reais.

3.2.4 Representação por árvores

A representação de indivíduos por árvores se torna interessante por ser bastante flexível e, teoricamente, abranger um espaço de busca maior, pois as árvores podem sofrer variações de seus tamanhos após as operações de *crossover* e mutação.

A codificação por árvores é útil para evoluir programas ou estruturas que possam ser codificados como árvores. A linguagem de programação LISP é usada freqüentemente para essa finalidade, visto que os programas em LISP podem ser facilmente analisados gramaticalmente como uma árvore. A Figura 3.5 ilustra um exemplo de representação por árvores.

Por ser uma representação flexível, uma grande desvantagem é que, com a evolução da população, as estruturas das árvores podem se tornar muito grandes e a evolução passar a tomar caminhos incontroláveis. Além disso, árvores com estruturas muito grandes passam a ser difíceis de entender.

Neste ponto, é importante discutir uma questão de terminologia. Em PG, cada indivíduo da população é um programa (contendo dados e operadores/funções aplicadas

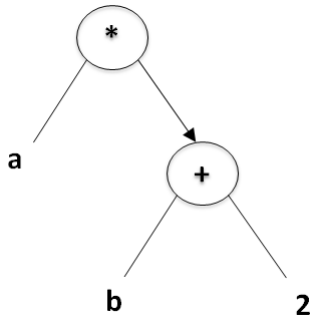


Figura 3.5: Indivíduo codificado como uma árvore.

aos dados), e esse indivíduo geralmente é representado como uma estrutura de árvore, mas não necessariamente deve ser essa a sua representação. Por essa razão, muitos autores confundem a classificação de seus algoritmos evolutivos e os classificam como PG quando simplesmente a forma de representação é árvore. Entretanto, o que realmente caracteriza uma PG é o que é representado, no caso um algoritmo ou um programa de computador, e não a forma de representação (Basgalupp et al., 2009b).

3.3 População inicial

A geração da população inicial é o processo no qual são criados os primeiros indivíduos. O número de indivíduos da população inicial é um parâmetro fornecido pelo usuário. É desejável que os indivíduos sejam os mais diferentes possíveis entre si, de modo que estejam espalhados no espaço de busca das soluções para o problema em questão, aumentando, assim, a diversidade da população.

O método mais comum de geração de população inicial é o método no qual os indivíduos têm seus genes preenchidos com valores aleatórios. Supondo que o cromossomo seja um *string* binário, um exemplo de preenchimento aleatório dos genes é assumir uma probabilidade p para cada gene receber o valor 1 e $(1 - p)$ para receber o valor 0.

Espera-se que os indivíduos que compõem a população inicial representem soluções factíveis para o problema em questão. Para garantir a factibilidade dessas soluções, utiliza-se, neste ponto, conhecimento sobre o problema. Um exemplo desse conhecimento é o domínio dos valores preenchidos em um *string* de valores reais, pois, dependendo do problema de otimização, não é qualquer valor que pode ser atribuído a qualquer gene do indivíduo.

3.4 Função de fitness

Nesta etapa, é medido o grau de aptidão dos indivíduos da população utilizando a função de *fitness*. A função de *fitness* é, junto com a etapa de codificação, uma das partes do algoritmo evolutivo que está diretamente relacionada com o domínio do problema em questão. Na função de fitness, um ou mais critérios (objetivos) podem ser utilizados para

otimização. A seguir, são apresentadas as formas de se resolver problemas de um único objetivo e também problemas multi-objetivos.

3.4.1 Um único objetivo

Em diversas aplicações, o problema a ser resolvido é natural de minimização ou maximização, onde o objetivo é maximizar um lucro $u(x)$ ou minimizar um custo $g(x)$. Nesses casos, torna-se intuitivo utilizar diretamente $u(x)$ ou $g(x)$ como função de *fitness*. Entretanto, em algumas situações não é desejável que a função de *fitness* assuma valores negativos. Um exemplo é quando se utiliza o método da roleta para a seleção. Assim, torna-se necessário realizar um mapeamento entre o valor obtido (ex: custo ou lucro) e o valor de *fitness*, garantindo que a função de *fitness* nunca assuma valores negativos.

É muito comum transformar problemas de minimização em problemas de maximização por questões de simplificação. Por exemplo, quando há um problema de minimização de custo, onde $g(x)$ é o custo associado ao indivíduo x, pode-se usar como *fitness* a Equação 3.1.

$$f(x) = C_{max} - g(x) \tag{3.1}$$

em que $f(x)$ representa o *fitness* do indivíduo x, $g(x)$ é o custo associado ao indivíduo x e C_{max} uma constante definida *a priori*. Há várias maneiras de escolher um valor para C_{max}. Por exemplo, C_{max} pode receber o maior valor observado de g, o maior valor de g na população corrente ou o maior valor de g nas últimas k gerações, garantindo que $f(x)$ nunca seja negativo para qualquer valor de x.

Quando o problema é de maximização, o mapeamento é mais direto. Entretanto, alguns valores podem ser negativos e o mapeamento deve resolver essa situação. Uma alternativa pode ser expressa pela Equação 3.2.

$$f(x) = u(x) + C_{min} \tag{3.2}$$

em que $f(x)$ representa o *fitness* do indivíduo x, $u(x)$ é o valor de lucro associado ao indivíduo x e C_{min} uma constante definida *a priori*. A idéia utilizada para determinar C_{min} é a mesma para atribuir um valor à C_{max}. Porém, ao invés de escolher os maiores, são escolhidos os menores valores de u.

3.4.2 Multi-objetivos

Uma questão crucial em mineração de dados é como avaliar a qualidade de um modelo candidato (ex: modelo de classificação como um conjunto de regras ou uma árvore de decisão), e diversos critérios podem, simultaneamente, ser levados em consideração para essa avaliação. Assim, quando deseja-se otimizar mais de um critério, surgem os problemas multi-objetivos, os quais são muito comuns em mineração de dados. De acordo com Freitas

(2004), há três abordagens para lidar com problemas multi-objetivos: (i) a abordagem convencional de fórmula-ponderada, em que um problema multi-objetivo é transformado em um problema de um único objetivo; (ii) a abordagem lexicográfica; e (iii) a abordagem de Pareto. Essas abordagens são apresentadas a seguir.

Abordagem de fórmula-ponderada

Na literatura de mineração de dados, esta é a abordagem mais utilizada para lidar com problemas multi-objetivos. A transformação de um problema multi-objetivo em um único objetivo é feita ao atribuir um peso numérico a cada objetivo (critério de avaliação) e então combinar os valores dos critérios ponderados por meio de operações como adição ou multiplicação. Supondo o problema de indução de árvores de decisão, um exemplo de função para avaliar a qualidade (*fitness*) de um indivíduo (árvore de decisão) é apresentado pela Equação 3.3.

$$fitness(i) = w_1 \times A(i) + w_2 \times TS(i) \tag{3.3}$$

em que w_1 e w_2 são os coeficientes de ponderação representando a importância de cada critério, nesse caso acurácia ($A(i)$) e tamanho da árvore ($TS(i)$), respectivamente, e i é o indivíduo (árvore de decisão) em questão.

A grande vantagem dessa abordagem é ser conceitualmente simples e fácil de usar (Deb & Kalyanmoy, 2001; Freitas, 2004). Esses fatores possivelmente explicam a popularidade dessa abordagem. Entretanto, há vários fatores negativos, os quais são discutidos a seguir.

O problema mais comum da abordagem de fórmula-ponderada é a forma *ad-hoc* com que os pesos são determinados, uma vez que os valores dos pesos são baseados em uma intuição vaga por parte do usuário a respeito da importância dos critérios utilizados ou então em uma experimentação variando esses pesos. Esse problema é conhecido por "números mágicos", os quais são, geralmente, justificados na literatura por afirmações vagas do tipo "os valores dos pesos foram determinados empiricamente" (Freitas, 2004). Além disso, uma vez que os pesos foram determinados e a função de *fitness* é definida, o algoritmo evolutivo irá buscar pela melhor solução considerando apenas esses pesos, não dando chance de encontrar soluções que poderiam ser exploradas com pesos diferentes. Esse problema é conhecido como "oportunidades perdidas".

Outro problema particular da abordagem de fórmula-ponderada envolve a soma/subtração (ao invés de multiplicação/divisão) de termos representando diferentes critérios, os quais podem ter escalas muito diferentes em suas unidades de medida. Otimizar a acurácia e a complexidade (número de nodos) de uma árvore de decisão é um exemplo típico desse problema. Para resolver esse problema, geralmente são aplicadas técnicas de normalização, as quais poderiam ser aplicadas na medida de complexidade, considerando o exemplo anterior, tornando seus valores na escala $[0, 1]$, da mesma forma que a acurácia. No entanto, existem várias técnicas de normalização e a escolha da melhor envolveria um conhecimento prévio do domínio de aplicação ou uma determinação

empírica com diferentes técnicas de normalização. Isso caracterizaria uma abordagem *ad-hoc* ao invés de uma abordagem baseada em princípios (Freitas, 2004).

Por fim, existe outro problema associado à abordagem de fórmula ponderada que é conhecido como "combinar maçãs com laranjas". Considerando o mesmo exemplo anterior para otimização da acurácia e complexidade (número de nodos) de árvores de decisão, é fácil enxergar que essas duas medidas não são comensuráveis para avaliação da qualidade de um modelo. É importante notar que, neste ponto, está sendo considerado que a normalização do número de nodos da árvore já foi realizada e, conseqüentemente, não está sendo levado em consideração o problema das diferentes escalas das medidas. O problema, neste ponto, é que geralmente não faz sentido nenhum adicionar/subtrair dois critérios que têm unidades de medida diferentes, além do significado do resultado de multiplicação/divisão desses critérios ser questionável. A solução mais coerente é reconhecer que esses critérios são bem diferentes e tratá-los separadamente e de uma outra forma, ou seja, não combiná-los na mesma fórmula (Freitas, 2004).

Embora seja muito provável que haja uma combinação de pesos que gere soluções ótimas para o problema de otimização em questão, encontrar esses pesos é um outro problema de otimização bastante complexo e, principalmente, muito demorado. Logo, é computacionalmente inviável encontrar esses pesos ótimos.

Abordagem de Pareto

Ao invés de transformar um problema multi-objetivo em um problema de um único objetivo, a idéia da abordagem de Pareto é a de usar um algoritmo multi-objetivo para solucionar o problema multi-objetivo original. Para isso, é utilizada a dominância de Pareto (Coello et al., 2006), a qual é definida como segue.

Uma solução s_1 é dita dominar uma solução s_2 se e somente se s_1 é estritamente melhor que s_2 em pelo menos um dos critérios a serem otimizados e s_1 não é pior que s_2 em todos os outros critérios. Considerando que todos os critérios c_i, $i = 1, ..., k$, devem ser minimizados, a solução s_1 domina s_2 se e somente se $\exists c_i$ tal que $s_1(c_i) < s_2(c_i)$ e $\forall c_i$, $i = 1, ..., k$, $s_1(c_i) \leq s_2(c_i)$, em que $s_1(c_i)$ e $s_2(c_i)$ denotam a qualidade das soluções s_1 e s_2, respectivamente, em relação ao critério c_i, e k é o número de critérios a serem otimizados. A solução s_i é dita não-dominada se e somente se não há uma solução s_j que domine s_i.

É importante destacar que na abordagem de Pareto não há uma combinação de diferentes critérios em uma única fórmula, pois todos os critérios são tratados separadamente. Conseqüentemente, essa abordagem não sofre do problema oriundo de medidas com escalas diferentes nem do problema de "combinar maçãs com laranjas". Além disso, ao contrário das outras abordagens, a abordagem de Pareto retorna ao usuário um conjunto de soluções, as não-dominadas, ao invés de uma única solução. Esses fatores implicam maior complexidade dos algoritmos de otimização multi-objetivos baseados em Pareto. Alguns exemplos de aplicação dessa abordagem são discutidos em Pappa et al. (2002). A Figura 3.6 ilustra um exemplo da curva de Pareto para um problema de dois

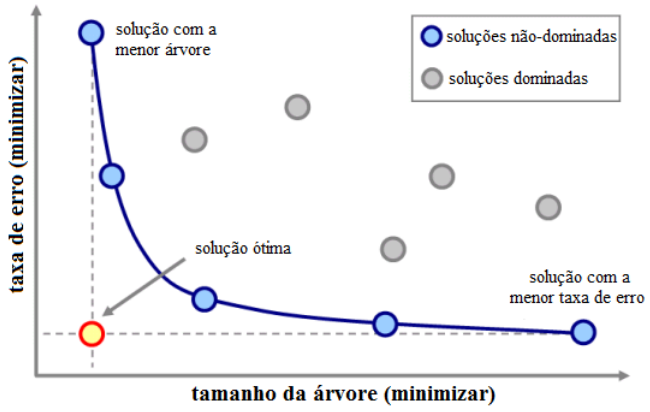


Figura 3.6: Curva de Pareto para o problema de minimizar a taxa de erro e o tamanho de uma árvore de decisão.

objetivos (minimizar a taxa de erro e minimizar o tamanho de uma árvore de decisão).

Foi destacado na seção anterior o problema em definir os pesos para os critérios a serem otimizados quando se opta pela utilização da abordagem de fórmula-ponderada. Nesse ponto, há uma interação com o usuário, para o qual, em grande parte das vezes, é atribuída a tarefa de informar os valores desses pesos baseando-se em seu conhecimento no domínio de aplicação. Com a utilização da abordagem de Pareto, há também a interação com o usuário, porém ela ocorre no final da execução do algoritmo. Como já mencionado, a abordagem de Pareto não retorna uma única solução, mas sim um conjunto de soluções não-dominadas. Assim, o usuário pode usar o seu conhecimento sobre o domínio e escolher uma dentre as soluções não-dominadas, baseado nesse conhecimento. Logo, embora introduza uma subjetividade (por parte do usuário) para a escolha final da melhor solução, acredita-se que é mais fácil e intuitivo para o usuário fazer essa escolha, baseada em seus conhecimentos, do que definir os pesos para os critérios a serem combinados em uma fórmula matemática ponderada.

Quando se tem algum conhecimento prévio do problema em questão, uso da abordagem de Pareto pode apresentar algumas desvantagens. A principal delas é a dificuldade em escolher a melhor solução a ser utilizada na prática, visto que, muitas vezes, há a necessidade por parte do usuário de obter apenas uma única solução. Considerando o contexto de evolução de árvores de decisão, seria possível combinar as árvores resultantes (as árvores não-dominadas), caracterizando um *ensemble* de árvores, e teoricamente obter boas acurácias. No entanto, em virtude da grande quantidade de árvores de decisão, as quais são em geral diferentes e inconsistentes, a interpretabilidade seria sacrificada, o que não seria bom para o usuário caso esse critério fosse importante para ele.

Outra característica 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, por exemplo, é consideravelmente mais importante que o tamanho da árvore, mas a abordagem de Pareto assume que ambos os critérios são igualmente relevantes (Freitas, 2004).

Abordagem Lexicográfica

A idéia básica desta abordagem não é atribuir pesos, mas sim diferentes prioridades a objetivos distintos, e então focar na otimização dos objetivos de acordo com as suas respectivas prioridades. Considere que x e y são indivíduos, e a e b dois critérios (objetivos) de avaliação. Além disso, considere também que a tem prioridade sobre b e que t_a e t_b são os limiares de tolerância associados com a e b, respectivamente. A abordagem lexicográfica trabalha de acordo com a seguinte análise: se $|a_x - a_y| > t_a$ então é possível estabelecer uma escolha pelo melhor indivíduo considerando apenas o critério a. Caso contrário, a medida b deve ser avaliada. Caso $|b_x - b_y| > t_b$, então o melhor indivíduo pode ser escolhido considerando o critério b. Se ainda assim os indivíduos continuarem "empatados", então utiliza-se apenas o valor absoluto de a para determinar o melhor indivíduo. Dependendo do critério utilizado, o melhor valor tanto para a como para b pode ser o maior ou o menor deles. Por exemplo, no caso dos indivíduos representarem árvores de decisão, a e b podem representar, respectivamente, acurácia e tamanho da árvore. Assim, dar-se-ia prioridade para um indivíduo com maior acurácia (maior valor para a). Porém, em caso de empate, optar-se-ia por uma árvore menor (menor valor de b), o que facilitaria a sua interpretabilidade. Considere, por exemplo, duas árvores de decisão t_1 e t_2, sendo que t_1 tem 90% de acurácia e 20 nodos, enquanto t_2 tem 70% de acurácia e 15 nodos. Muitos pesquisadores em mineração de dados e usuários dariam preferência pela árvore t_1 sobre a t_2. Entretanto, a abordagem de Pareto considera que essas duas árvores são equivalentes, pois nenhuma domina a outra.

É viável considerar que muitos possam argumentar que definir limiares é tão problemático quanto definir os pesos de uma abordagem de fórmula-ponderada. Realmente, essa não é uma tarefa trivial. Uma abordagem bastante comum é utilizar métodos baseados em estatística para definir esses limiares, como por exemplo o desvio-padrão, o qual permite rejeitar uma hipótese nula de diferença insignificante entre dois valores com um certo grau de confiança. Embora ainda continue sendo necessário que o usuário defina um parâmetro, o grau de confiança, é mais aceitável esperar do usuário que ele defina algo como "95% de confiança estatística" do que esperar que ele atribua bons pesos a uma fórmula-ponderada.

Outra vantagem sobre a abordagem de fórmula-ponderada é que a abordagem lexicográfica trata cada critério separadamente, reconhecendo que cada um deles mede um aspecto diferente de qualidade de uma solução candidata. Conseqüentemente, a abordagem lexicográfica não sofre dos problemas mencionados anteriormente, tais como o problema dos "números mágicos", o problema de combinar critérios de escalas totalmente diferentes, e o problema de "combinar maçãs com laranjas" (Freitas, 2004).

A capacidade da abordagem lexicográfica de atribuir diferentes prioridades a diferentes critérios a serem otimizados é o que a diferencia das outras abordagens, pois em muitos domínios de aplicação essa capacidade é indispensável. No contexto de evolução de árvores de decisão, e de classificadores em geral, está claro que a acurácia é um critério que deve ser

prioritário quando comparado ao número de nodos. Nesse contexto, seria dada preferência às árvores menores somente em caso de diferença não significativa na acurácia.

3.5 Seleção

É no estágio de seleção que são escolhidos os indivíduos que formarão a próxima geração da população. Nesse ponto, faz-se uso do grau de aptidão (*fitness*) dos indivíduos, e os mais aptos possuem probabilidade maior de serem escolhidos como reprodutores. É importante destacar que a escolha é probabilística, não sendo garantido, portanto, que o melhor indivíduo (o mais apto) será um dos selecionados para formar a próxima geração da população.

O elitismo foi introduzido por Kenneth De Jong (Jong, 1975) e pode ser utilizado por vários métodos de seleção. Essa propriedade garante que um número x de melhores indivíduos seja mantido na próxima geração da população. Esses melhores indivíduos poderiam ser perdidos após as operações de *crossover* e mutação. Segundo Mitchell (1998), muitos pesquisadores têm encontrado uma melhoria significativa na performance dos AEs quando utilizam o elitismo. Entretanto, é bom tomar bastante cuidado com o valor de x, pois, se muito grande, pode causar o domínio prematuro dos indivíduos mais aptos desde as gerações iniciais, implicando uma rápida perda de diversidade na população.

Os métodos de seleção mais populares, os quais são discutidos em Mitchell (1998), são apresentados a seguir.

Roleta

Neste método de seleção, cada indivíduo possui um intervalo de números de uma roleta. Esse intervalo de números é proporcional ao valor de *fitness* do indivíduo. A roleta é girada n vezes, em que n é o número de reprodutores. A cada giro da roleta, um número z é marcado, sendo que cada número da roleta deve ter igual probabilidade de ser marcado. O indivíduo selecionado é aquele que possuir o número z dentro de seu intervalo de números. No final do processo, são selecionados n indivíduos reprodutores.

Tabela 3.1: Representação dos valores de *fitness* pra cada indivíduo de uma população.

Indivíduo	a	b	c	d	e	f	g	h	i	j
Valor de *fitness*	7	6	1	10	4	8	3	3	6	9

Considerando a Tabela 3.1, a qual representa uma população de 10 indivíduos com seus respectivos valores de *fitness*, a roleta com os intervalos de números destinados a cada indivíduo pode ser visualizada na Figura 3.7.

A grande desvantagem deste método é que um pequeno grupo de indivíduos, os mais aptos, possuem probabilidades de seleção bem maiores que os outros indivíduos. Isso pode

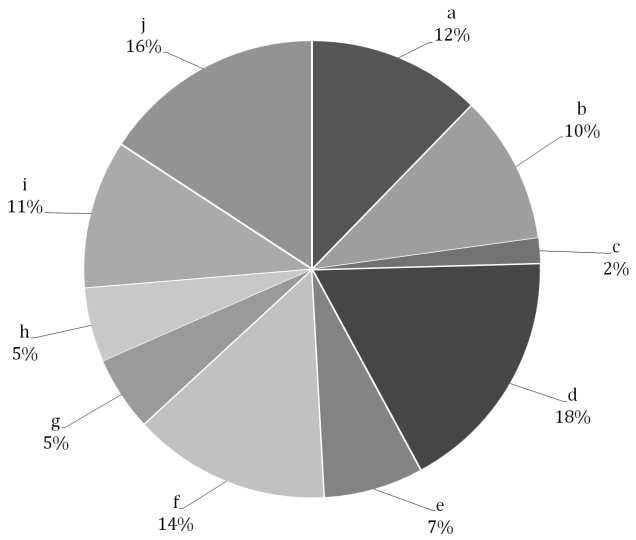


Figura 3.7: Roleta representando a proporção do intervalo de números reservado para cada um dos 10 (dez) indivíduos da população.

aumentar a pressão seletiva, não permitindo maior diversidade da população nas gerações iniciais e, conseqüentemente, convergindo para ótimos locais.

Ranking

O método *ranking* previne a convergência prematura para ótimos locais no espaço de soluções do problema. Nesse caso, os indivíduos da população são ordenados de acordo com o seu *fitness* e a probabilidade de escolha é atribuída conforme a posição do indivíduo no *ranking*. Quando se utiliza esse método, o problema de grandes diferenças entre as chances de seleção dos indivíduos é eliminado, uma vez que a relação das probabilidades de escolha entre dois indivíduos de posições i e $i+1$ no *ranking* é independente das diferenças absolutas de seus valores de *fitness*, dependendo apenas das posições no *ranking*.

O *ranking* evita dar probabilidade maior de seleção para um grupo pequeno de indivíduos altamente aptos, reduzindo, assim, a pressão da seleção quando a variação do *fitness* é muito elevada.

Se, por um lado, não levar em conta a informação do valor absoluto do *fitness* é vantajoso para evitar uma convergência prematura, por outro é desvantajoso, visto que, em alguns casos, pode ser importante conhecer a diferença entre os valores de aptidão de dois indivíduos consecutivos no *ranking*, principalmente nas gerações finais da população, quando espera-se que estejam as melhores soluções para o problema.

Torneio

Neste método, t indivíduos são escolhidos aleatoriamente na população para participarem de um torneio. Quanto maior o tamanho de t, maior a chance dos indivíduos mais aptos serem selecionados, aumentando, assim, a pressão seletiva. Há também um parâmetro k ($0 \leq k \leq 1$) que indica a probabilidade do melhor indivíduo, dentre os

participantes do torneio, ser selecionado. Geralmente $k = 1$, caracterizando o chamado torneio determinístico (Eiben & Smith, 2003). Após uma rodada do torneio, todos os participantes, inclusive o vencedor, retornam à população original e podem novamente ser selecionados para os próximos torneios. Esse processo se repete até que o número desejado de reprodutores seja alcançado.

A seleção por torneio é, provavelmente, a mais utilizada na literatura de algoritmos evolutivos. Isso se explica, principalmente, por sua extrema simplicidade e pelo fato de que a pressão seletiva pode ser facilmente controlada ao variar o tamanho t do torneio. Outra vantagem deste método é que, assim como o *ranking*, não há a necessidade de normalizar o *fitness* dos indivíduos, visto que o fato de um indivíduo ser duas ou três vezes mais apto que outro não é o que determina a decisão da escolha, mas sim qual deles é o melhor.

3.6 Operadores genéticos

3.6.1 Crossover

Após serem selecionados os pares de indivíduos (pais), eles podem reproduzir. Cada par de pais gera um par de filhos. Cada filho é composto por uma combinação dos genes dos pais. Quais genes dos pais passarão para cada um dos filhos é determinado pelo método de *crossover* utilizado. Cabe salientar que o *crossover* acontece de acordo com uma probabilidade, que em geral é alta, a qual é um parâmetro pré-determinado pelo usuário. Caso não ocorra o *crossover*, os filhos são exatamente cópias de seus pais.

Os tipos mais comuns de *crossover* são apresentados a seguir.

Crossover de 1 ponto

Este tipo de cruzamento é composto por um ponto de *crossover* em uma posição p, em que o $filho_1$ recebe os genes do pai até o ponto p, e os genes da mãe do ponto p em diante. Já o $filho_2$ recebe os genes da mãe até o ponto p e os genes do pai do ponto p em diante. Esse processo pode ser visualizado na Figura 3.8, no qual os dois indivíduos de cima representam os pais enquanto os dois abaixo representam os filhos.

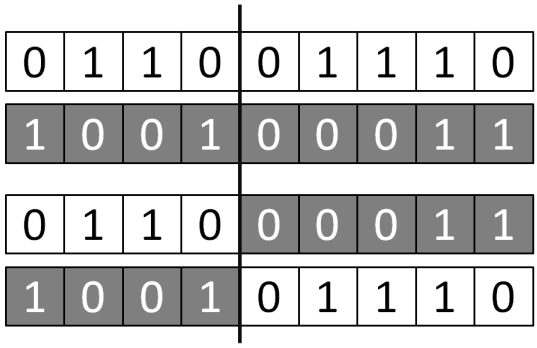


Figura 3.8: Um exemplo de *crossover* de 1 ponto.

Crossover Multi-pontos

Esse cruzamento é semelhante ao *crossover* de 1 ponto, porém pode haver n pontos de *crossover*. A Figura 3.9 ilustra os filhos gerados para um exemplo com $n = 2$.

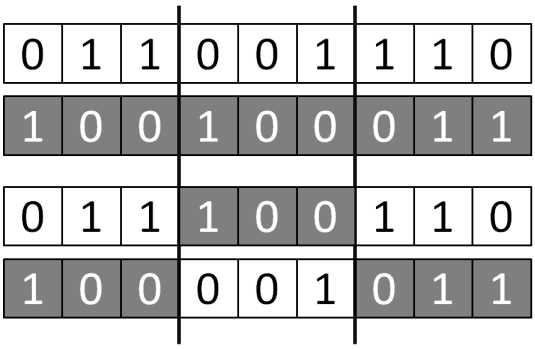


Figura 3.9: Um exemplo de *crossover* multi-pontos.

Crossover Uniforme

No crossover uniforme, os genes a serem trocados são escolhidos aleatoriamente em cada execução. A Figura 3.10 ilustra o exemplo de uma execução desse tipo de cruzamento.

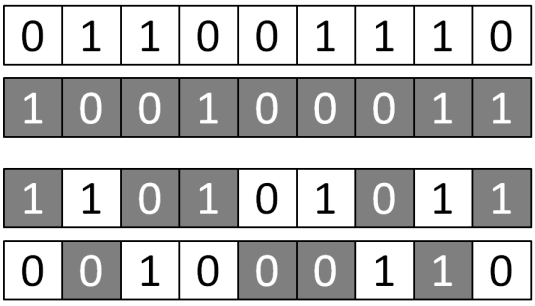


Figura 3.10: Um exemplo de *crossover* uniforme.

Crossover de árvores

Um ponto de *crossover* é selecionado em cada um dos pais, os separando em duas partes, as quais compões os filhos. Um exemplo desse tipo de *crossover* pode ser visualizado na Figura 3.11.

3.6.2 Mutação

Após todo o processo de *crossover*, antes da nova população ser definida, os indivíduos ainda podem passar pelo processo de mutação. A mutação é uma alteração aleatória no valor dos genes de um indivíduo e ocorre com uma probabilidade pré-determinada, a qual é fornecida pelo usuário, que em geral é baixa. Pode-se efetuar a mutação de um gene trocando seu valor por outro qualquer, somando ou subtraindo um valor

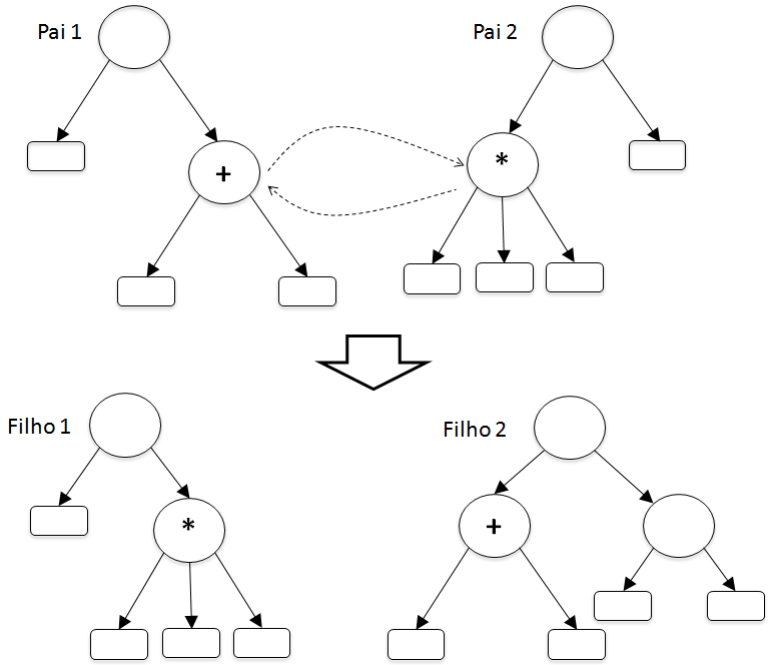


Figura 3.11: Um exemplo de *crossover* entre dois indivíduos representados por árvores.

aleatoriamente escolhido, trocando os valores dos genes de posições aleatórias, dentre outras opções. A mutação destina-se a especular novos pontos no espaço de busca de soluções, explorando novas possibilidades, como também a evitar que aconteça a chamada "convergência prematura", evento no qual o algoritmo converge para um ótimo local no espaço de soluções do problema, paralisando o processo evolutivo. A Figura 3.12 ilustra uma possível mutação para um indivíduo com representação binária, em que três de seus genes são alterados.

0	1	1	0	0	1	1	1	0
0	0	1	0	1	1	1	0	0

Figura 3.12: Um exemplo de mutação em um indivíduo com representação binária.

3.7 Indução de árvores de decisão

Algoritmos Evolutivos já foram explorados no contexto de indução de árvores de decisão. Trabalhos como Cantu-Paz & 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, uma combinação linear de atributos, ao invés de um único atributo, para particionar o conjunto de exemplos. Encontrar a melhor árvore de decisão oblíqua é um problema NP-completo, o que motivou os autores de Cantu-Paz & 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 AEs, a PG tem sido amplamente utilizada como método de indução de árvores de decisão (Koza, 1991; Zhao, 2007; Estrada-Gil et al., 2007; Bot & Langdon, 2000; Loveard & Ciesielski, 2001, 2002; Shirasaka et al., 1998; Zhao & Shirasaka, 1999; Tür & 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. Zhao (2007) considerou 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 & 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 & Ciesielski (2001), os autores propuseram diferentes alternativas para representar um problema de classificação multi-classes por meio de PG e, após, estenderam seu trabalho (Loveard & Ciesielski, 2002) para lidar com atributos categóricos. Em Shirasaka et al. (1998) e Zhao & Shirasaka (1999), os autores propuseram o projeto de árvores de classificação binária usando PG, e em Tür & Güvenir (1996) é descrito um algoritmo de PG para indução de árvores considerando somente atributos binários.

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

Considerando os AGs, não há muitos trabalhos relacionados que os classificam como tal para indução de árvores de decisão. Em Carvalho & 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 teve como objetivo gerar uma árvore ótima, mas sim gerar uma árvore usando o C4.5 e, então, substituir algumas regras extraídas das árvores (as regras de pequenos disjuntos) por regras evoluídas pelo AG.

A indução de AD por meio de AG é apresentado em Kretowski & Grzes (2005), Z. Bandar & McLean (1999), Llora & Garrell (2001), Fu et al. (2003a), Fu et al. (2003b), Fu et al. (2006) e Gray & Fan (2008). Em Kretowski & Grzes (2005), a função de *fitness* utiliza dois critérios: Q_{RCLass}, qualidade de classificação estimada no conjunto de treinamento, 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 & McLean (1999), cada árvore de decisão tem um número fixo de nodos que são representados como um vetor estático, e somente divisões binárias desses nodos são permitidas. Essa abordagem impõe a restrição de que atributos categóricos 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 necessariamente é verdade porque as divisões binárias podem gerar árvores com mais níveis de profundidade que, conseqüentemente, resultarão em regras mais difíceis de serem interpretadas.

Em Llora & Garrell (2001), é proposto o algoritmo GALE para indução de árvores de decisão ortogonais, oblíquas e multivariadas. As árvores ortogonais se caracterizam por seus nodos internos testarem um único atributo, usualmente um teste do tipo $a_i < \alpha$, sendo a_i um atributo do problema e α uma constante numérica. As árvores oblíquas sugerem usar testes mais elaborados nos nodos internos, algo do tipo $\sum_{i=1}^{d} w_i * a_i + w_{d+1}$, onde $\overline{w} = \langle w_1, w_2, ..., w_{d+1} \rangle$ é um vetor de coeficientes. Já as árvores multivariadas (Llora & Garrell, 2000) são caracterizadas por seus nodos internos que contêm um protótipo δ (um exemplo definido artificialmente) (Wettschereck, 1994; Domingos, 1995) e um limiar de ativação Θ. O nodo é ativado se a equação $\sqrt{\sum_{i=1}^{d}(\delta_i - e_i)^2} \leq \Theta$ é satisfeita, sendo e_i o valor de a_i no exemplo e a ser classificado. Como função de *fitness*, o GALE utiliza a acurácia do conjunto de treinamento. A grande limitação do GALE é que só trabalha com atributos numéricos, sendo necessário um pré-processamento adequado para atributos categóricos.

Fu et al. (Fu et al., 2003a,b, 2006) propuseram o GAIT, um algoritmo genético para indução de árvores de decisão binárias. Esse algoritmo recebe como entrada árvores de decisão geradas pelo algoritmo C4.5 com a idéia de evoluí-las. O *fitness* é calculado por um único objetivo, acurácia do conjunto de validação (*scoring*, como é chamado pelos autores). O operador de *crossover* utilizado é bastante comum, em que árvores trocam sub-árvores entre si. Na mutação, dois nodos (de sub-árvores diferentes) são selecionados

aleatoriamente e suas sub-árvores são trocadas. O TARGET também implementa um método de pós-poda por *default*, o qual é baseado na acurácia do conjunto de validação. De acordo com os autores, os resultados obtidos pelo GAIT superaram o C4.5. No entanto, não foi possível replicar os experimentos porque o algoritmo não foi disponibilizado pelos autores.

Gray & Fan (2008) propuseram o TARGET, um algoritmo genético para indução de árvores de decisão binárias cujos nodos podem testar até três variáveis ao mesmo tempo. Embora não seja um algoritmo multi-objetivo, pois utiliza apenas a acurácia de treinamento como *fitness*, o TARGET utiliza a medida BIC (Schwarz, 1978) para penalizar árvores complexas e manter as árvores mais simples. TARGET implementa dois tipos de *crossover* (troca de nodos e troca de sub-árvores) e quatro tipos de mutação (conjunto de divisão, regra de divisão, troca de nodos, troca de sub-árvores), além de implementar o elitismo, em que os cinco melhores indivíduos são mantidos para a próxima geração. Os autores concluem que o TARGET superou o CART em termos de acurácia. Entretanto, da mesma forma que o GAIT, os autores de TARGET não disponibilizaram o algoritmo e não foi possível replicar os experimentos.

Sem considerar a questão de terminologia, em nenhum dos trabalhos supracitados foi utilizada a abordagem lexicográfica para otimização de múltiplos objetivos. A Tabela 3.2 resume os trabalhos relacionados de acordo com a característica dos respectivos algoritmos propostos.

Tabela 3.2: Características dos algoritmos propostos nos trabalhos relacionados.

	Codificação	**# Arestas**	**Nodos**	**Seleção**	***Fitness***	**Poda**
Shirasaka et al. (1998)	árvore	binária	univariada	roleta	simples	−−
Z. Bandar & McLean (1999)	*string*	binária	univariada	não informado	simples	pós
Llora & Garrell (2001)	árvore	múltiplas	univariada or multivariada	roleta	simples	pós
Kretowski (2004)	*string*	binária	multivariada	roleta (*linear scaling*)	fórmula-ponderada	pós
Kretowski & Grzes (2005)	árvore	binária	multivariada	*linear ranking*	fórmula-ponderada	pós
Zhao (2007)	árvore	múltiplas	multivariada	torneio	pareto	−−
Estrada-Gil et al. (2007)	árvore	binária	univariada	torneio	fórmula-ponderada	−−

3.8 Considerações finais

Este capítulo apresentou uma revisão sobre algoritmos evolutivos, em especial quanto às diferentes abordagens para lidar com problemas multi-objetivos. Por meio de uma revisão sobre os trabalhos relacionados, especificamente para indução de árvores de decisão, foi observado que os algoritmos evolutivos têm tido bastante sucesso nesse tipo de aplicação, principalmente quando há a necessidade da área de aplicação em obter classificadores não só com boa acurácia, mas também com uma boa interpretabilidade, caracterizando um problema multi-objetivo. Dentre as abordagens multi-objetivas apresentadas, destacou-se a lexicográfica que, embora até onde se saiba não tenha sido utilizada nesse contexto, se mostra uma abordagem bastante promissora para ser adotada

na indução de árvores de decisão. Nesse sentido, decidiu-se adotar essa abordagem no contexto de indução de árvores de decisão, e essa proposta é apresentada no capítulo seguinte.

Capítulo

4

Algoritmo LEGAL-Tree

Este capítulo apresenta o algoritmo proposto neste trabalho, o LEGAL-Tree (*LExicographical Genetic Algorithm for Learning decision Trees*), o qual é um método evolutivo multi-objetivo baseado na abordagem lexicográfica para indução de árvores de decisão. A seguir, é feita uma descrição sobre cada detalhe de implementação desse algoritmo.

4.1 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 l arestas, em que l 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 limiar, o qual é determinado automaticamente pelo algoritmo.

A Figura 4.1 ilustra um exemplo desses tipos de representação.

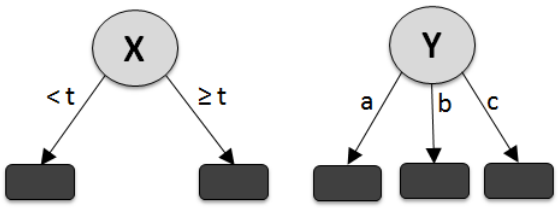


Figura 4.1: Exemplo de representação de nodos categórico (Y) e numérico (X).

4.2 Gerando a população inicial

A idéia inicial foi gerar indivíduos de forma totalmente aleatória, selecionando atributos para formarem os nodos não-terminais e dividindo o conjunto de treinamento apropriadamente. Sem considerar valores desconhecidos, os nodos categóricos dividiriam o conjunto de treinamento de acordo com os respectivos valores das arestas, pois há uma para cada categoria possível. Já os nodos numéricos dividiriam os dados de acordo com um limiar gerado aleatoriamente (ex: valor do atributo para um exemplo qualquer do conjunto de treinamento). O processo seria repetido recursivamente até que não houvesse mais nodos não-terminais, e estes seriam rotulados com valores de classe, também aleatórios.

Entretanto, foi decidido não utilizar essa abordagem porque, mesmo que a construção 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.

Para a geração da população inicial, LEGAL-Tree dispõe de duas opções. A primeira faz o uso dos *decisions stumps* (Basgalupp et al., 2009b,a) para geração das árvores. A segunda abordagem é uma técnica mais avançada, em que é usado um algoritmo de indução de árvores de decisão, BeamClassifier, para gerar as árvores inicias. Esses dois métodos são apresentados a seguir.

4.2.1 Decision stumps

A primeira estratégia de incorporar conhecimento específico no AG é definida pelo pseudo-código ilustrado no Algoritmo 1. No final da execução, a lista $dsList$ será composta pelos *decision stumps* gerados.

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 & Mason, 1999). Neste trabalho, 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. Quando o nodo representa um atributo numérico, é utilizado o ganho de informação (Equação 2.1) para determinar o valor do limiar utilizado para a

Algoritmo 1 Geração dos *Decision Stumps*

1: seja *data* o conjunto de treinamento
2: seja *nAtt* o número de atributos de *data*
3: divida *data* em 10 partes disjuntas
4: seja *dsList* a lista de *decision stumps*
5: dado que y_j representa a $j - ésima$ parte de *data*
6: **para** $y_j = 1$ até 10 **faça**
7: dado que a_i é o $i - ésimo$ atributo de *data*
8: **para** $a_i = 1$ até *nAtt* **faça**
9: **se** a_i é numérico **então**
10: limiar = infoGain(y_j,a_i)
11: dsList.add(new numericDS(a_i,limiar))
12: **senão**
13: dsList.add(new categoricalDS(a_i))
14: **fim se**
15: **fim para**
16: **fim para**

divisão. O pseudo-código ilustrado no Algoritmo 1 irá basicamente gerar um conjunto de *decision stumps* (na verdade, $10 \times n$ *decision stumps*, em que n é o número de atributos preditivos do conjunto de treinamento). Ao dividir o conjunto de treinamento 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 o mais diversificada possível. Tal processo de geração da população inicial baseado em *decision stumps* não deixa de ser aleatório, porém evita geração de soluções de baixa qualidade e não factíveis, que atrasariam 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 4.2 ilustra esse processo, 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 por causa da aplicação dos operadores genéticos que afetarão suas estruturas.

As duas principais vantagens dessa abordagem para a geração da população inicial de árvores de decisão são que: (a) ao invés de aleatória, a escolha dos limiares para atributos numéricos em nodos não-terminais é baseada no ganho de informação, automaticamente

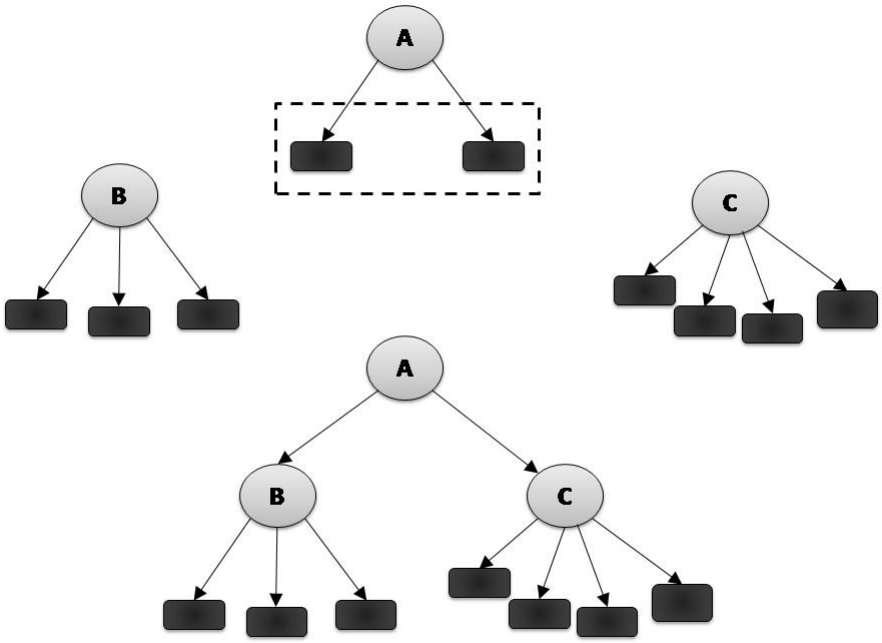


Figura 4.2: Combinando três *decision stumps* (A, B e C) para construção de um indivíduo de profundidade 2.

detectando correlações nos dados; e (b) a classe associada com o nodo folha é sempre a mais freqüente nos exemplos cobertos pelo nodo, ao invés de uma classe escolhida aleatoriamente. Ambas as vantagens são resultados 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 eficácia, como discutido previamente.

Com o intuito de garantir que somente soluções factíveis sejam geradas, LEGAL-Tree evita que o mesmo atributo categórico representado por um nodo x seja utilizado em outro nodo das sub-árvores de x. Caso haja repetição de nodos numéricos, os limiares são ajustados, utilizando o ganho de informação, de forma a evitar casos indesejáveis (ex: após seguir um possível caminho dado por um nodo x em que o valor deve ser menor que um limiar t, não teria sentido reavaliar o mesmo atributo em um nodo filho y considerando valores maiores que t, pois nenhum exemplo seria coberto por tal condição).

4.2.2 BeamClassifier

Com o objetivo de reduzir os problemas oriundos da busca gulosa usada em muitos algoritmos de indução de árvores de decisão (C4.5, CART, etc.), é proposto neste trabalho um novo algoritmo de indução baseado em *beam search* (Russell & Norvig, 2003), chamado BeamClassifier. De acordo com Russell & Norvig (2003), o algoritmo *beam search* mantém w estados ao invés de apenas um. Ele inicia com w estados gerados aleatoriamente. A cada passo, todos os sucessores de todos os w estados são gerados. Se é encontrada uma solução satisfatória, o algoritmo pára. Caso contrário, são selecionados os w melhores

sucessores da lista completa e então o processo é repetido. A estratégia gulosa é um caso particular de *beam search* com largura $w = 1$. Ao utilizar diferentes valores para w, é possível que outros pontos do espaço de busca sejam explorados, uma vez que w possibilidades são exploradas ao invés de apenas uma.

Neste trabalho, propõe-se utilizar o BeamClassifier para gerar as árvores da população inicial do LEGAL-Tree. Inicia-se com n árvores de decisão (nodos raiz), em que n é o número de atributos da base de dados em questão, e cada nodo raiz representa um desses atributos. A seguir, o algoritmo seleciona as melhores w árvores de acordo com um dado critério (apresentados posteriormente), e cada uma delas é expandida em até n caminhos diferentes, isto é, considerando cada um dos atributos da base de dados. O número de expansões é menor que n quando há redundância (ver Seção 4.8).

O algoritmo implementa a expansão como segue. Para cada árvore, ocorre n expansões, em que n é o número de atributos da base de dados. Para cada expansão, o algoritmo realiza uma busca pré-fixada, expandindo recursivamente (para cada atributo) o nodo raiz e então seus filhos da esquerda para a direita. Assim, um passo do algoritmo é realizado e restam $n \times w$ árvores, das quais são selecionadas as w melhores para a próxima expansão. Esse processo é repetido até que um critério de parada seja satisfeito.

Embora o algoritmo BeamClassifier siga uma busca local para explorar o domínio de soluções, ele usa também mais de uma alternativa na busca, evitando muitos casos de ótimos locais. Para calcular o ganho de expansão, BeamClassifier pode utilizar tanto o ganho de informação (Equação 2.1) como o *Gini* (Equação 2.5), e essa opção é um parâmetro definido pelo usuário. O princípio de funcionamento de BeamClassifier é ilustrado no Algoritmo 2.

Algoritmo 2 Algoritmo BeamClassifier

```
seja data o conjunto de treinamento
seja h a lista de todas as árvores intermediárias
seja e a lista das árvores expandidas
seja bestW a lista das w melhores árvores
seja expansible um valor booleano
expansible = true
e = createInicialTrees(data)
h.add(e)
bestW = theBestWTrees(e)
expansible = isExpansible(bestW)
enquanto expansible faça
  e = expand(bestW);
  bestW = theBestWTrees(e);
  h.add(e);
  expansible = isExpansible(bestW);
fim enquanto
h.add(bestW)
```

Na Figura 4.3 pode ser visto um exemplo de uma execução (fictícia) de BeamClassifier considerando a base de dados *PlayTennis* (Quinlan, 1986), com $w = 2$. O algoritmo

inicia gerando uma árvore para cada atributo (4 árvores iniciais) e então seleciona as duas ($w = 2$) melhores de acordo com um critério (acurácia para o conjunto de validação). As duas árvores selecionadas são, então, expandidas ao considerar todas as possibilidades de atributos para cada nodo folha. Logo, o próximo passo é selecionar novamente as duas melhores árvores considerando todas as geradas. Nesse exemplo, 15 árvores foram geradas durante a expansão e 17 árvores são consideradas para a seleção das duas melhores árvores (as 15 que foram geradas por meio das expansões mais as duas previamente selecionadas, as quais geraram essas 15). O passo de expansão ocorre somente quando há um ganho significativo no particionamento dos dados. O algoritmo termina sua execução quando a expansão das melhores árvores não resultar em ganho significativo (determinado pelo método $isExpansible()$ no Algoritmo 2).

Todas as árvores que foram avaliadas e expandidas durante a execução são usadas para comporem a população inicial do LEGAL-Tree. Por exemplo, todas as árvores ilustradas na Figura 4.3, incluindo as omitidas pelas reticências, são usadas para formar a população inicial de árvores, isto é, cada uma dessas árvores é um indivíduo da população.

Caso o número de árvores geradas por esse processo seja maior que o tamanho da população (p), então somente as melhores p árvores (considerando a função de *fitness*) são selecionadas para formar a população inicial. Similarmente, se o número de árvores geradas durante a execução de BeamClassifier é menor que p, então o restante da população é preenchida com *decision stumps* selecionados aleatoriamente.

Ao gerar a população inicial por meio do algoritmo BeamClassifier, espera-se atingir um alto nível de diversidade entre as árvores, uma vez que os indivíduos terão diferentes tamanhos e estruturas. Ou seja, não serão exclusivamente árvores completas como os indivíduos gerados pelo método anterior (*decision stumps*). Além disso, esse método também faz uso do conhecimento específico da tarefa de classificação, diminuindo o número de gerações necessárias para atingir soluções de boa qualidade.

4.3 Tratamento de valores desconhecidos

Para o tratamento de valores desconhecidos, são utilizadas estratégias muito simples:

- Atributos categóricos: se o nodo a ser analisado representa um atributo categórico, os valores desconhecidos são substituídos pela moda de todos os valores do atributo no conjunto de treinamento;
- Atributos numéricos: se o nodo a ser analisado representa um atributo numérico, os valores desconhecidos são substituídos pela média aritmética de todos os valores do atributo no conjunto de treinamento.

Foi analisada também a estratégia de criar uma aresta adicional representando o valor desconhecido. Entretanto, os resultados obtidos não foram satisfatórios. Como era esperado, esse procedimento aumentou o tamanho das árvores geradas por LEGAL-Tree.

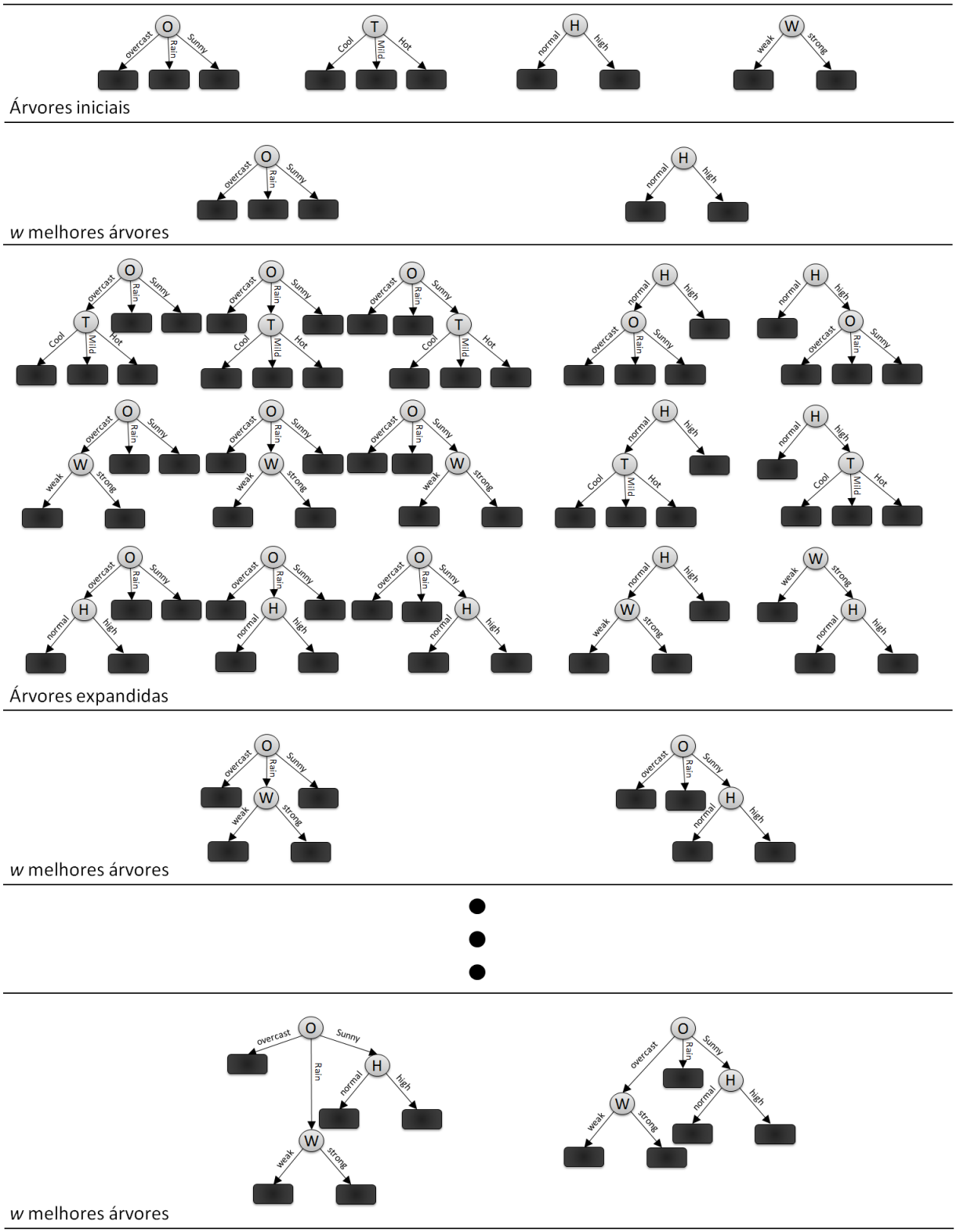


Figura 4.3: Exemplo de execução de BeamClassifier.

Além disso, essas árvores não apresentaram ganho em acurácia, e em muitos casos ainda houve perda.

4.4 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).

Conhecimentos descobertos em mineração de dados devem ser não somente precisos mas também interpretáveis pelo usuário (Witten & Frank, 1999), em que interpretabilidade pode ser estimada pelo tamanho do classificador - assume-se que classificadores menores são preferíveis quando outros critérios permanecem iguais. Isso é justificado por *Occam's razor* (Domingos, 1999), que essencialmente diz que, dentre múltiplas hipóteses que são equivalentemente consistentes com os dados, deve ser escolhida aquela mais simples. Nesse contexto, a avaliação do *fitness* de LEGAL-Tree é feita por uma abordagem multi-objetiva lexicográfica, a qual consiste basicamente de atribuir diferentes prioridades a diferentes objetivos e, então, focar na otimização desses objetivos de acordo com suas respectivas prioridades (Freitas, 2008). Neste trabalho, são considerados acurácia preditiva e tamanho da árvore como objetivos, as quais são medidas bastante utilizadas para avaliação de árvores de decisão. Entretanto, esse conjunto de objetivos pode ser facilmente estendido para um número o de objetivos de acordo com o domínio de aplicação a ser considerado, tais como profundidade da árvore, custo dos atributos utilizados, *precision*, *recall*, medida-F, 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 maçãs com laranjas" (combinar 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, pois comprometem a interpretabilidade da solução. 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, por exemplo, é consideravelmente mais importante que o tamanho

da árvore, mas a abordagem de Pareto supõe que ambos os objetivos são igualmente relevantes. Esses problemas foram discutidos em mais detalhes na Seção 3.4.

A abordagem lexicográfica não sofre dos problemas supracitados, além de ser conceitualmente simples, fácil de usar, de implementar e de estender para objetivos adicionais. Adicionalmente, até onde se sabe, a abordagem lexicográfica ainda não havia sido utilizada para indução de árvores de decisão até que Basgalupp et. al. a propôs em Basgalupp et al. (2009a,b).

LEGAL-Tree permite ao usuário definir os limiares, o que é particularmente útil para os casos em que o usuário tem um conhecimento prévio do domínio, mas também permite a definição automática dos limiares baseada nos resultados fornecidos pelo teste-t pareado corrigido (Nadeau & Bengio, 2003). Nesse caso, a única informação que o usuário deve fornecer é o nível de significância (por exemplo, 90% ou 95%) e o teste-t pareado corrigido indica se a diferença das acurácias desses indivíduos é estatisticamente significativa ou não.

Considerando o conjunto de validação (o mesmo poderia ser aplicado para o conjunto de treinamento), são comparados os valores observados das classes com os valores preditos pelo classificador. Para cada exemplo do conjunto de validação, é possível determinar se o classificador acertou (1) ou se errou (0) a classificação. Assim, é gerado um vetor binário que, se calculada a média de seus valores, é obtida a acurácia. Assim como é utilizado para comparação de médias, o teste-t pode ser utilizado para comparar a acurácia de dois indivíduos (classificadores) baseado em seus respectivos vetores binários. O resultado desse teste definirá se os dois indivíduos são ou não significativamente diferentes.

Como não é possível gerar um vetor binário (acerto ou erro) ao considerar outras medidas (ex: tamanho da árvore), pois não existe um gabarito, o procedimento é estritamente diferente: se a menor árvore é menor que uma porcentagem (ex: 80%) do tamanho da maior árvore, a diferença é considerada significativa. Caso contrário, os valores absolutos das medidas mais prioritárias são analisados para indicar o melhor indivíduo. Assim, basta o usuário definir o valor dessa porcentagem de tolerância.

O Algoritmo 3 ilustra como a avaliação lexicográfica (com a definição prévia dos limiares) pode ser utilizada para comparar dois indivíduos (x e y). Quando houver empate em todos os critérios, a escolha pode ser aleatória. A adaptação para a abordagem utilizando teste estatístico pode ser feita de forma bem simples.

4.5 Seleção

Para selecionar os indivíduos que formarão as próximas gerações da população, LEGAL-Tree usa o método do torneio, um método popular e bastante efetivo (ver Seção 3.5). É também implementada a seleção por elitismo, a qual mantém os melhores indivíduos nas próximas gerações. Tanto o número de participantes do torneio como o número de indivíduos que fazem parte da elite são parâmetros que podem ser definidos

Algoritmo 3 Função de *fitness* lexicográfica

sejam x e y dois indivíduos (árvores de decisão)
seja T o conjunto de treinamento
seja V o conjunto de validação
seja $A(z, S)$ a acurácia do indivíduo z para o conjunto de dados S
seja $TS(z)$ o número de nodos do indivíduo (árvore) z
seja ATT o limiar para a acurácia de treinamento
seja ATV o limiar para a acurácia de validação
seja TST o limiar para o número de nodos
se $|A(x, V) - A(y, V)| > ATV$ **então**
  se $A(x, V) > A(y, V)$ **então**
    o indivíduo x é o melhor
  senão
    o indivíduo y é o melhor
  senão se $|A(x, T) - A(y, T)| > ATT$ **então**
    se $A(x, T) > A(y, T)$ **então**
      o indivíduo x é o melhor
    senão
      o indivíduo y é o melhor
    senão se $|TS(x) - TS(y)| > TST$ **então**
      se $TS(x) < TS(y)$ **então**
        o indivíduo x é o melhor
      senão
        o indivíduo y é o melhor
      senão se $A(x, V) > A(y, V)$ **então**
        o indivíduo x é o melhor
      senão se $A(x, V) < A(y, V)$ **então**
        o indivíduo y é o melhor
      senão se $A(x, T) > A(y, T)$ **então**
        o indivíduo x é o melhor
      senão se $A(x, T) < A(y, T)$ **então**
        o indivíduo y é o melhor
      senão se $TS(x) > TS(y)$ **então**
        o indivíduo x é o melhor
      senão se $TS(x) < TS(y)$ **então**
        o indivíduo y é o melhor
      senão
        os indivíduos x e y são iguais
      fim se
    fim se
  fim se
fim se

pelo usuário.

Em cada rodada do torneio, os participantes são comparados dois a dois por meio da função de *fitness* lexicográfica, e aquele com o maior número de vitórias é selecionado para fazer parte dos reprodutores. Quando há um empate em número de vitórias, o indivíduo com o valor absoluto mais alto para o critério mais prioritário (ex: acurácia para o conjunto de validação) é selecionado. Esse processo é repetido até que sejam selecionados todos os indivíduos que participarão da etapa de reprodução.

4.6 Crossover

LEGAL-Tree implementa a operação de *crossover* como segue. Primeiro, dois indivíduos são escolhidos aleatoriamente dentre os selecionados (vencedores dos torneios). Para cada indivíduo, um número é gerado aleatoriamente entre 1 (nodo raiz) e n (número total de nodos da árvore em questão), indicando nodos específicos de cada árvore. Para localizar cada um desses nodos, LEGAL-Tree realiza uma busca pré-fixada, visitando recursivamente o nodo raiz e, em seguida, 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 os números sorteados para ambas as árvores (indivíduos), LEGAL-Tree troca as respectivas sub-árvores representadas por esses nodos, os quais representam os nodos raiz de cada sub-árvore. Assim, dois novos indivíduos são gerados.

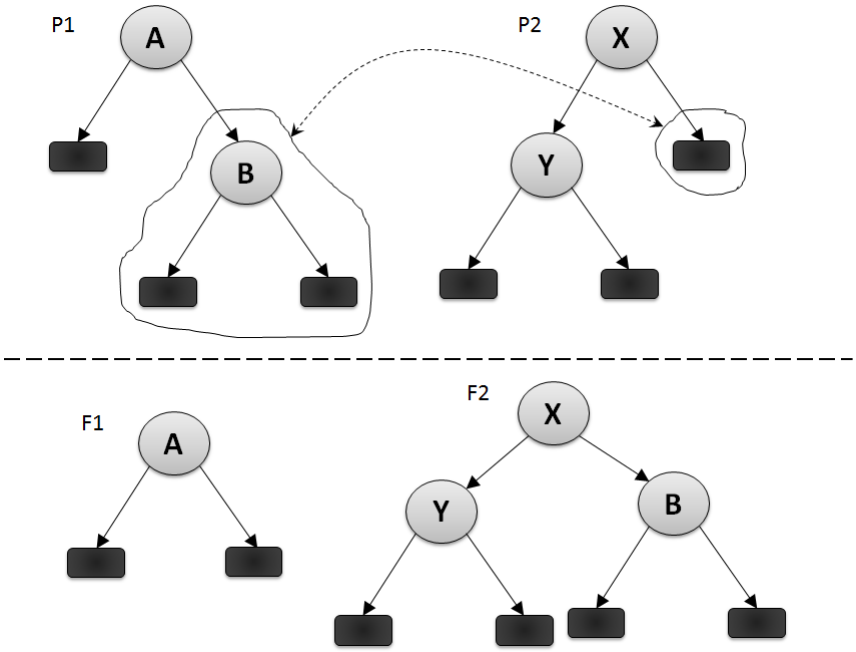


Figura 4.4: *Crossover* entre dois indivíduos, $P1$ e $P2$, formando dois novos indivíduos, $F1$ e $F2$.

Observe na Figura 4.4 os dois indivíduos ($P1$ e $P2$) selecionados para reprodução. Para $P1$, o nodo sorteado foi o B, enquanto que para o $P2$ foi sorteado o nodo folha cercado. Após identificar esses dois nodos pelo método de busca supracitado, a operação *crossover* irá criar dois indivíduos filhos ($F1$ e $F2$), que serão resultados da estrutura de um dos pais com uma parte do outro. $F1$ mantém a estrutura de $P1$, mas herda o nodo folha (e tudo o que estivesse abaixo caso fosse um nodo interno) de $P2$. Similarmente, $F2$ mantém a estrutura de $P2$ e herda o nodo B (junto com seus filhos) de $P1$.

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 *crossover* 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 (limiares que resultam em intervalos irregulares ao utilizar nodos numéricos). A Seção 4.8 mostra detalhes de como LEGAL-Tree trata dessas questões.

4.7 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 4.5 ilustra ambas as estratégias apresentadas, as quais são denominadas "Mutação de redução" e "Mutação de expansão", respectivamente.

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.

4.8 Factibilidade das soluções

Após as operações de *crossover* 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 *crossover*. 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 limiares 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

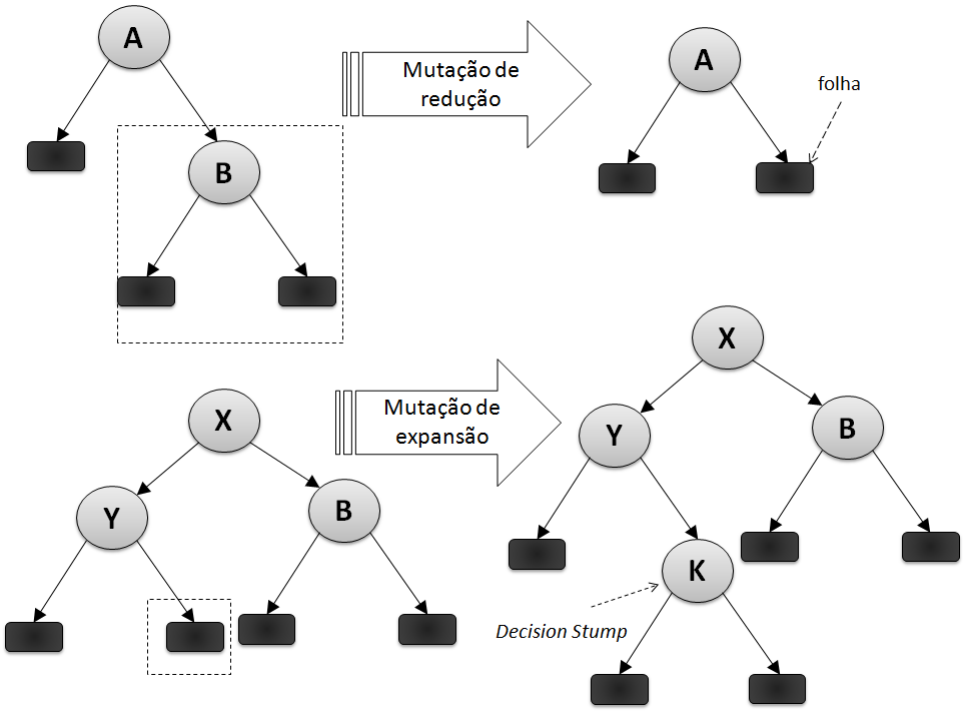


Figura 4.5: Duas estratégias,"Mutação de redução" e "Mutação de expansão", para mutação dos indivíduos.

caso, LEGAL-Tree substitui toda a sub-árvore por um nodo folha representando a classe mais freqüente (Filtro Categórico). Por outro lado, se o nodo é numérico e seu limiar é inconsistente com algum nodo ancestral, então o limiar é ajustado para evitar caminhos que não contemplem nenhum exemplo (Filtro Numérico). A Figura 4.6 ilustra exemplos desses dois problemas e como eles são solucionados pelos filtros.

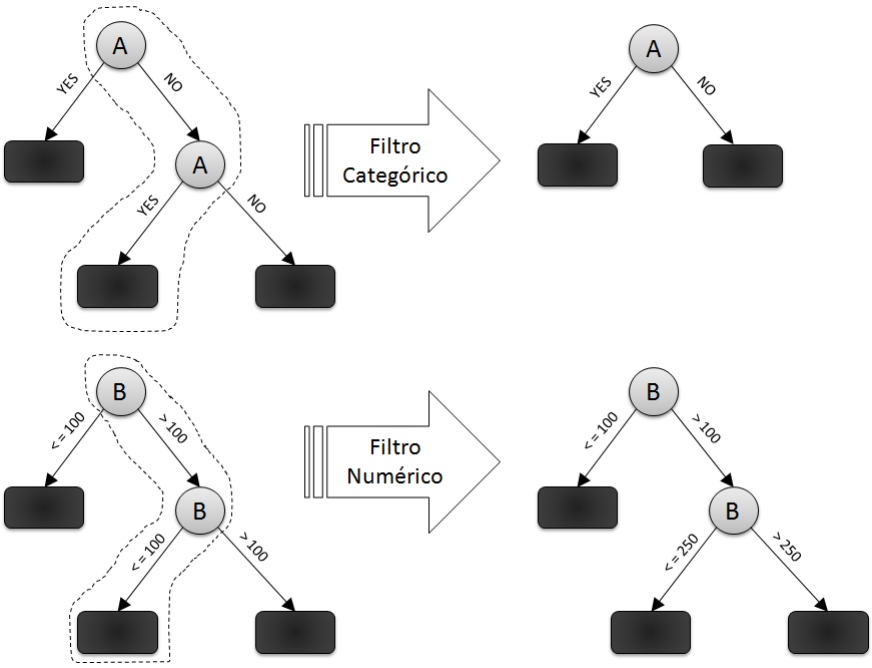


Figura 4.6: Filtros para eliminar redundâncias (Filtro Categórico) e inconsistências (Filtro Numérico).

Esses problemas são tratados por LEGAL-Tree por meio de filtros aplicados após as operações de *crossover* e mutação. Esses filtros são similares aos utilizados na geração da população inicial, quando são evitados esses problemas durante a combinação dos *decision stumps* para a geração das árvores de decisão.

Outro problema que pode ocorrer após as operações de *crossover* e mutação é quando um nodo folha deixa de representar a classe mais freqüente para os exemplos cobertos por ele. Essa ocorrência 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).

4.9 Considerações finais

Neste capítulo, foi apresentado um novo algoritmo para indução de árvores de decisão chamado LEGAL-Tree. O algoritmo LEGAL-Tree apresenta algumas diferenças em relação aos 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 propõe 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 encontrados na utilização de fórmula-ponderada e Pareto. Até o momento, LEGAL-Tree considera dois critérios a serem otimizados: acurácia e tamanho da árvore (a qual representa uma medida de complexidade). No próximo capítulo, é realizada uma análise experimental em que LEGAL-Tree é comparado com outros algoritmos (tradicionais e evolutivos) de indução de árvores de decisão.

Capítulo

5

Análise Experimental do Algoritmo LEGAL-Tree

Neste capítulo são apresentados os experimentos realizados aplicando o algoritmo LEGAL-Tree em bases de dados para diversos problemas de classificação. Com o objetivo de fazer uma análise comparativa, outros algoritmos de indução de Árvore de Decisão (AD) também são aplicados, tais como J48, SimpleCart, GALE e BeamClassifier. Os resultados dessa análise também são discutidos neste capítulo.

5.1 Plano experimental

Para a avaliação dos algoritmos de indução de árvores de decisão utilizados neste trabalho, foram utilizadas bases de dados de diversos problemas de classificação. Essas bases de dados foram coletadas do repositório de dados de aprendizado de máquina UCI (Asuncion & Newman, 2007), incluindo anneal, audiology, autos, balance-scale, colic, credit-g, diabetes, glass, heart-c, heart-statlog, hepatitis, ionosphere, iris, labor, lymph e primary-tumor. Na Tabela 5.1 são apresentadas informações como número de exemplos, número de atributos categóricos (entre parênteses aparece o número de atributos binários), número de atributos numéricos, número de classes e acurácia *default* (porcentagem da classe majoritária) para cada uma das bases de dados.

Nos experimentos realizados, além do algoritmo proposto neste trabalho, o LEGAL-Tree, foram utilizados também outros algoritmos de indução de árvores de decisão. São eles: o algoritmo J48, o qual é a implementação do tradicional algoritmo C4.5 (Quinlan, 1993) disponível na ferramenta Weka (Witten & Frank, 1999), o algoritmo SimpleCart, implementação do CART (Breiman et al., 1984) também disponível no

Tabela 5.1: Sumarização das bases de dados utilizadas nos experimentos.

	# exemplos	atributos numéricos	atributos categóricos	# classes	(%) acurácia *default*
anneal	898	8	32 (19)	6	76
audiology	226	0	69 (61)	24	25
autos	205	15	10 (4)	7	33
balance-scale	625	4	0	3	46
colic	368	7	15 (2)	2	63
credit-g	1000	7	14 (2)	2	70
diabetes	768	8	0	2	65
glass	214	9	0	6	36
heart-c	303	6	7 (3)	5	54
heart-statlog	270	13	0	2	56
hepatitis	155	6	13 (13)	2	79
ionosphere	351	34	0	2	64
iris	150	4	0	3	33
labor	57	8	8 (3)	2	65
lymph	148	3	15 (9)	4	55
primary-tumor	339	0	17 (14)	22	25

Weka, o algoritmo BeamClassifier, proposto neste trabalho como método de geração da população inicial, e o algoritmo evolutivo GALE (Llora & Garrell, 2001). Além do GALE, também foram solicitados aos respectivos autores outros algoritmos evolutivos para indução de árvores de decisão, tais como o GAIT (Fu et al., 2006) e o TARGET (Gray & Fan, 2008). Porém, esses autores não responderam ou não disponibilizaram os respectivos códigos.

O método escolhido para a avaliação foi a validação cruzada com 10 partições, que é amplamente utilizado e recomendado para avaliação de algoritmos de AM (Tan et al., 2005). É importante mencionar que todos os algoritmos utilizaram as mesmas divisões (conjuntos de treinamento e teste) das bases de dados, garantindo uma comparação justa dos resultados obtidos. No caso dos algoritmos LEGAL-Tree e BeamClassifier, metade do conjunto total de treinamento foi utilizado como conjunto de validação. Para cada algoritmo, foram coletadas as seguintes medidas: acurácia (treinamento, validação e teste para LEGAL-Tree e BeamClassifier, treinamento e teste para os algoritmos J48 e SimpleCart, e apenas acurácia de teste para o algoritmo GALE, uma vez que apenas essa é disponibilizada pelo algoritmo), tamanho (número de nodos) e profundidade das árvores.

Como os algoritmos LEGAL-Tree e GALE são estocásticos, eles foram executados 30 vezes para cada uma das 10 divisões realizadas pela validação cruzada. Após a execução sobre as 16 bases de dados apresentadas na Tabela 5.1, foram calculadas as médias e os respectivos desvios-padrão das 30 execuções de cada partição, e então a média das 10 partições. No caso dos outros algoritmos (J48, SimpleCart e BeamClassifier), foram

calculadas as médias e desvios-padrão para cada uma das 10 partições, pois os algoritmos são determinísticos e não há a necessidade de serem executados mais de uma vez.

Os experimentos foram realizados em um *cluster*, chamado *pi-cluster*, o qual foi disponibilizado pela Universidade de Kent, na Inglaterra. O *pi-cluster* consiste de 31 processadores Pentium-4 com 3.2 GHz. Cada nodo possui 1 GB de RAM e 40 GB de HD. Como o *cluster* estava disponível para muitos usuários, os quais o utilizavam de forma contínua e simultânea, os tempos de execução dos algoritmos não foram levados em consideração na análise comparativa, pois os experimentos não foram realizados nas mesmas condições para permitir o uso desse critério.

Os algoritmos J48, SimpleCart e GALE foram executados com os valores *default* para seus respectivos parâmetros. Os parâmetros de LEGAL-Tree foram determinados de acordo com o conhecimento prévio do autor, e podem ser vistos na Tabela 5.2.

Tabela 5.2: Parâmetros utilizados por LEGAL-Tree nos experimentos

Parâmetro	Valor
Número de indivíduos da população	200
Percentagem (sobre o número máximo de gerações) que termina o algoritmo caso não haja melhoria.	10%
Número máximo de gerações.	200
Percentagem (sobre o número total) dos indivíduos que participarão do torneio.	5%
Percentagem (sobre o número total de indivíduos) de elitismo, em que os melhores são mantidos para a próxima geração (elitismo)	5%
Taxa de Crossover	90%
Taxa de Mutação	5%

Compartilhando a configuração padrão para os parâmetros apresentados na Tabela 5.2, foram executadas três variações do algoritmo LEGAL-Tree: LEGAL-DS, LEGAL-BC e LEGAL-ST. Em todas as versões, foram utilizados os mesmos critérios para a avaliação lexicográfica: acurácia para o conjunto de validação, acurácia para o conjunto de treinamento e tamanho da árvore, nessa ordem de prioridade. A acurácia para o conjunto de treinamento foi introduzida após ser constatado que ao utilizar apenas a acurácia para o conjunto de validação, ocorria o problema de *underfitting* (Basgalupp et al., 2009b). Em todas as execuções, o algoritmo BeamClassifier foi configurado para utilizar o ganho de informação para calcular o ganho de expansão (Seção 4.2.2).

Em LEGAL-DS, é utilizada a técnica dos *decision stumps* para a geração da população inicial, enquanto que em LEGAL-BC a população inicial é gerada pelo algoritmo BeamClassifier. Em ambas as versões, são utilizados os limiares de 2%, 2% e 3 nodos para acurácia de validação, acurácia de treinamento e tamanho da árvore, respectivamente. Em LEGAL-DS, foi usado o valor 4 como profundidade máxima das árvores geradas na população inicial. Já em LEGAL-BC, foi utilizado $w = 3$, em que w é a largura do *Beam Search* (mais detalhes na Seção 4.2.2).

A versão LEGAL-ST adiciona ao LEGAL-BC a automatização dos limiares das acurácias por meio do teste-t pareado corrigido com $\alpha = 0.05$. Para o tamanho da árvore de decisão, a diferença é considerada significativa quando a menor árvore é menor que 80% da maior árvore (mais detalhes na Seção 4.4).

Para comparar os resultados obtidos pelos algoritmos supracitados, foram utilizados diferentes métodos estatísticos.

Para verificar a significância estatística das diferenças dos resultados observados por cada algoritmo, par a par, foi utilizado o test-t pareado corrigido (Nadeau & Bengio, 2003). Os dados utilizados consistem das acurácias para cada uma das 10 partições de teste, além dos tamanhos e profundidades das árvores de decisão geradas. O test-t pareado corrigido foi utilizado com nível de significância $\alpha = 0.05$ e 9 graus de liberdade.

Para avaliar se houve diferença significativa entre todos os algoritmos, ao mesmo tempo, foi utilizado o teste de Friedman (Friedman, 1940), um teste não-paramétrico baseado em *ranking*. Foi utilizada a implementação disponível no *software* estatístico *R*. Esse teste é fortemente recomendado em Demšar (2006) para comparação de múltiplos classificadores. Entretanto, Demšar (2006) recomenda o teste quando n e k são suficientemente grandes ($n > 10$ e $k > 5$), em que n é o número de bases de dados e k o número de classificadores. Essa restrição se aplica no contexto deste trabalho, pois são comparados 7 algoritmos em 16 bases de dados. Uma vez que o teste de Friedman é aplicado, ele apenas indica se há ou não uma diferença significativa entre todos os algoritmos sendo considerados. Se confirmada essa diferença, utiliza-se o teste de Nemenyi (Nemenyi, 1983) para verificação de quais algoritmos são melhores que os outros, par a par. Tanto o teste de Friedman quanto de Nemenyi foram aplicados com $\alpha = 0.10$ e 6 graus de liberdade. Optou-se por utilizar α maior que o teste-t porque os testes de Friedman e Nemenyi são mais conservadores.

5.2 Análise dos resultados

Os resultados obtidos para os algoritmos LEGAL-DS, LEGAL-BC, LEGAL-ST, GALE, J48, SimpleCart e BeamClassifier (BC) são apresentados nas Tabelas 5.3, 5.4, 5.5, 5.6,5.7, 5.8 e 5.9, respectivamente.

A Tabela 5.10 apresenta a acurácia preditiva, com os respectivos desvios-padrão, para todos os algoritmos do experimento. Para cada base de dados, o melhor resultado está em negrito. Considerando os valores absolutos e os desvios-padrão (para desempate), está bem claro que há o domínio de SimpleCart (CART), J48 e LEGAL-Tree em relação aos outros métodos. Tanto o J48 quanto o SimpleCart apresentaram os melhores resultados em 5 das 16 bases de dados. Já o algoritmo LEGAL-Tree apresentou o melhor resultado em 6 bases de dados, sendo 3 por LEGAL-DS e 3 por LEGAL-BC. O algoritmo GALE foi superior em apenas uma base de dados (balance-scale), mesmo assim essa vitória foi divida com o SimpleCart, que obteve o mesmo resultado para essa base. Os algoritmos LEGAL-ST e

Tabela 5.3: Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo LEGAL-Tree com a configuração LEGAL-DS.

	Acurácia de Treinamento	Acurácia de Validação	Acurácia de Teste	Tamanho da Árvore	Profundidade da Árvore
anneal	0.95 (0.03)	0.93 (0.03)	0.94 (0.03)	21.14 (4.51)	3.46 (0.81)
audiology	0.82 (0.04)	0.66 (0.03)	0.69 (0.05)	28.19 (7.92)	5.37 (1.05)
autos	0.95 (0.02)	0.80 (0.05)	0.73 (0.08)	82.46 (19.49)	5.58 (1.2)
balance-scale	0.86 (0.03)	0.84 (0.02)	0.78 (0.03)	41.43 (18.47)	7.42 (1.73)
colic	0.89 (0.01)	0.83 (0.01)	0.85 (0.01)	7.16 (2.03)	2.32 (0.49)
credit-g	0.77 (0.03)	0.79 (0.02)	0.71 (0.03)	43.63 (24.13)	5.17 (1.53)
diabetes	0.83 (0.03)	0.75 (0.02)	0.74 (0.03)	23.70 (19)	5.25 (2.45)
glass	0.95 (0.03)	0.82 (0.03)	0.69 (0.08)	48.48 (9.2)	7.77 (1.32)
heart-c	0.91 (0.03)	0.85 (0.02)	0.78 (0.05)	31.24 (11.26)	4.90 (1.03)
heart-statlog	0.89 (0.02)	0.85 (0.02)	0.80 (0.05)	17.33 (7)	4.44 (1.14)
hepatitis	0.94 (0.02)	0.89 (0.02)	0.82 (0.07)	16.48 (4.82)	4.51 (0.92)
ionosphere	0.93 (0.02)	0.94 (0.01)	0.90 (0.04)	12.25 (4.84)	4.95 (1.6)
iris	0.99 (0.01)	0.96 (0.01)	0.95 (0.01)	6.64 (1.05)	2.78 (0.43)
labor	0.95 (0.04)	0.97 (0.02)	0.82 (0.12)	14.42 (4.77)	3.95 (0.87)
lymph	0.94 (0.03)	0.83 (0.03)	0.78 (0.06)	22.73 (7.04)	4.15 (0.86)
primary-tumor	0.57 (0.06)	0.46 (0.03)	0.39 (0.05)	50.02 (18.52)	7.96 (1.28)

Tabela 5.4: Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo LEGAL-Tree com a configuração LEGAL-BC.

	Acurácia de Treinamento	Acurácia de Validação	Acurácia de Teste	Tamanho da Árvore	Profundidade da Árvore
anneal	0.98 (0.01)	0.96 (0.01)	0.97 (0.01)	22.20 (1.83)	4.40 (0.5)
audiology	0.89 (0.02)	0.73 (0.01)	0.72 (0.02)	39.88 (5.08)	6.70 (0.45)
autos	0.95 (0.02)	0.86 (0.01)	0.76 (0.04)	78.47 (10.71)	5.39 (0.52)
balance-scale	0.85 (0.02)	0.82 (0.01)	0.77 (0.03)	33.37 (11.97)	6.18 (1.24)
colic	0.89 (0)	0.83 (0)	0.86 (0)	6.12 (0.27)	2.05 (0.09)
credit-g	0.76 (0.02)	0.78 (0.01)	0.72 (0.02)	39.24 (15.2)	4.51 (0.89)
diabetes	0.85 (0.02)	0.75 (0.01)	0.76 (0.02)	22.28 (10.89)	5.64 (1.53)
glass	0.95 (0.02)	0.81 (0.02)	0.70 (0.05)	43.49 (5.48)	8.14 (1.1)
heart-c	0.87 (0.02)	0.86 (0.01)	0.78 (0.02)	18.60 (5.37)	4.35 (0.69)
heart-statlog	0.89 (0.02)	0.84 (0.02)	0.78 (0.04)	14.77 (3.85)	4.18 (0.8)
hepatitis	0.93 (0.01)	0.83 (0.01)	0.79 (0.03)	9.06 (2.63)	2.88 (0.66)
ionosphere	0.94 (0.01)	0.95 (0)	0.90 (0.01)	12.31 (2.47)	5.19 (0.75)
iris	0.99 (0)	0.95 (0)	0.95 (0.01)	6.96 (0.41)	2.97 (0.18)
labor	0.90 (0.04)	0.94 (0.01)	0.80 (0.07)	9.12 (2.51)	2.89 (0.7)
lymph	0.93 (0.02)	0.82 (0.02)	0.76 (0.04)	18.35 (4.35)	3.56 (0.55)
primary-tumor	0.55 (0.05)	0.45 (0.02)	0.40 (0.04)	41.90 (15.52)	7.47 (1.21)

Tabela 5.5: Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo LEGAL-Tree com a configuração LEGAL-ST.

	Acurácia de Treinamento	Acurácia de Validação	Acurácia de Teste	Tamanho da Árvore	Profundidade da Árvore
anneal	0.99 (0)	0.98 (0.01)	0.98 (0.02)	38.60 (4.77)	6.29 (0.94)
audiology	0.85 (0.03)	0.71 (0.03)	0.69 (0.08)	40.46 (5.06)	6.99 (1.05)
autos	0.83 (0.03)	0.76 (0.02)	0.66 (0.11)	57.62 (8.78)	4.48 (0.46)
balance-scale	0.81 (0.02)	0.81 (0.02)	0.76 (0.04)	28.07 (4.68)	6.86 (0.55)
colic	0.89 (0.01)	0.83 (0.01)	0.85 (0.05)	11.96 (4.75)	2.81 (0.35)
credit-g	0.74 (0.03)	0.78 (0.01)	0.72 (0.03)	46.22 (19.63)	4.90 (1.42)
diabetes	0.84 (0.02)	0.75 (0.01)	0.75 (0.04)	22.17 (3.4)	6.05 (1.04)
glass	0.84 (0.05)	0.77 (0.06)	0.68 (0.08)	28.63 (7.35)	6.37 (0.73)
heart-c	0.82 (0.04)	0.86 (0.02)	0.79 (0.1)	13.44 (7.46)	3.59 (1.79)
heart-statlog	0.86 (0.02)	0.83 (0.02)	0.77 (0.08)	15.41 (3)	4.81 (0.76)
hepatitis	0.92 (0.03)	0.83 (0.02)	0.79 (0.07)	9.29 (5.28)	2.97 (1.27)
ionosphere	0.95 (0.01)	0.95 (0.01)	0.89 (0.06)	18.33 (3.51)	6.76 (1.1)
iris	0.99 (0.01)	0.96 (0.01)	0.95 (0.05)	8.02 (0.91)	3.17 (0.19)
labor	0.86 (0.04)	0.94 (0.03)	0.78 (0.15)	7.19 (2.8)	2.43 (0.76)
lymph	0.89 (0.02)	0.81 (0.01)	0.76 (0.11)	17.71 (3.03)	3.11 (0.34)
primary-tumor	0.38 (0.02)	0.35 (0.01)	0.34 (0.03)	11.37 (1.59)	3.42 (0.52)

Tabela 5.6: Acurácia, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo GALE.

	Acurácia de Teste	Tamanho da Árvore	Profundidade da Árvore
anneal	0.84 (0.03)	5.42 (1.93)	1.14 (0.86)
audiology	0.44 (0.09)	7.17 (3.36)	1.61 (1.21)
autos	0.51 (0.07)	12.73 (4.7)	3.11 (1.17)
balance-scale	0.79 (0.04)	29.93 (10.78)	4.61 (1.22)
colic	0.82 (0.02)	4.00 (1.1)	0.50 (0.53)
credit-g	0.71 (0.03)	22.09 (13.11)	4.01 (1.54)
diabetes	0.74 (0.02)	11.41 (6.96)	2.52 (1.47)
glass	0.59 (0.09)	14.05 (5.26)	3.08 (1.03)
heart-c	0.77 (0.06)	12.81 (6.41)	2.70 (1.17)
heart-statlog	0.79 (0.05)	21.78 (9.63)	3.98 (1.25)
hepatitis	0.80 (0.05)	6.71 (3.03)	1.53 (1.03)
ionosphere	0.89 (0.03)	12.31 (5.23)	3.54 (1.27)
iris	0.95 (0.03)	6.43 (1.73)	1.58 (0.63)
labor	0.68 (0.19)	2.91 (1.3)	0.15 (0.43)
lymph	0.77 (0.07)	12.95 (6.17)	2.94 (1.18)
primary-tumor	0.31 (0.04)	8.48 (4.65)	1.83 (1.11)

Tabela 5.7: Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo J48.

	Acurácia de Treinamento	Acurácia de Teste	Tamanho da Árvore	Profundidade da Árvore
anneal	1.00	0.98	50.10	5.80
	(0)	(0.01)	(5.3)	(0.4)
audiology	0.90	0.78	48.90	9.50
	(0.01)	(0.08)	(3.65)	(0.81)
autos	0.94	0.82	62.00	6.00
	(0.01)	(0.07)	(6.72)	(1)
balance-scale	0.90	0.77	78.20	8.00
	(0.01)	(0.04)	(11.39)	(0.77)
colic	0.86	0.85	8.10	1.60
	(0.01)	(0.05)	(2.02)	(0.49)
credit-g	0.85	0.71	117.40	8.10
	(0.02)	(0.03)	(28.2)	(1.81)
diabetes	0.84	0.74	37.40	7.00
	(0.02)	(0.05)	(12.39)	(1.26)
glass	0.92	0.67	44.20	7.90
	(0.01)	(0.08)	(5.23)	(0.83)
heart-c	0.92	0.78	43.30	5.10
	(0.02)	(0.08)	(11.33)	(1.04)
heart-statlog	0.92	0.77	33.80	5.10
	(0.01)	(0.09)	(3.71)	(0.54)
hepatitis	0.92	0.84	17.80	4.60
	(0.02)	(0.07)	(4.4)	(0.92)
ionosphere	0.99	0.91	27.40	7.90
	(0.01)	(0.03)	(3.56)	(1.51)
iris	0.98	0.96	8.40	2.70
	(0)	(0.05)	(0.92)	(0.46)
labor	0.90	0.74	5.90	1.10
	(0.05)	(0.21)	(2.34)	(0.83)
lymph	0.93	0.77	28.30	4.70
	(0.01)	(0.1)	(3.49)	(0.46)
primary-tumor	0.61	0.40	83.70	8.70
	(0.01)	(0.05)	(6.65)	(0.78)

Tabela 5.8: Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo SimpleCart.

	Acurácia de Treinamento	Acurácia de Teste	Tamanho da Árvore	Profundidade da Árvore
anneal	1.00 (0)	0.99 (0.01)	21.40 (1.74)	5.90 (0.3)
audiology	0.84 (0.05)	0.73 (0.09)	34.40 (9.76)	6.30 (1.49)
autos	0.93 (0.02)	0.75 (0.1)	44.60 (6.31)	6.40 (1.02)
balance-scale	0.89 (0.03)	0.79 (0.04)	69.20 (32.4)	6.50 (0.92)
colic	0.87 (0.02)	0.86 (0.05)	7.40 (4.54)	1.70 (1.27)
credit-g	0.80 (0.03)	0.74 (0.04)	28.00 (15.34)	6.80 (3.37)
diabetes	0.79 (0.02)	0.76 (0.04)	12.00 (7.11)	3.40 (2.15)
glass	0.78 (0.05)	0.72 (0.08)	18.40 (10.12)	4.50 (1.2)
heart-c	0.86 (0.02)	0.82 (0.08)	12.40 (3.69)	2.40 (0.66)
heart-statlog	0.86 (0.03)	0.79 (0.07)	11.80 (2.99)	2.60 (0.8)
hepatitis	0.84 (0.06)	0.78 (0.04)	4.80 (6.23)	0.80 (1.66)
ionosphere	0.94 (0.03)	0.89 (0.05)	12.60 (7.58)	3.00 (1.73)
iris	0.97 (0.01)	0.95 (0.05)	7.60 (2.01)	2.30 (1)
labor	0.93 (0.04)	0.76 (0.19)	8.80 (4.33)	2.00 (0.89)
lymph	0.87 (0.05)	0.76 (0.08)	13.00 (7.21)	2.70 (1.35)
primary-tumor	0.52 (0.04)	0.39 (0.06)	31.20 (9.48)	7.80 (1.17)

Tabela 5.9: Acurácias, tamanho e profundidade, e os respectivos desvios-padrão, das árvores de decisão geradas pelo algoritmo BeamClassifier.

	Acurácia de Treinamento	Acurácia de Validação	Acurácia de Teste	Tamanho da Árvore	Profundidade da Árvore
anneal	0.97 (0.01)	0.96 (0.01)	0.96 (0.02)	66.40 (12.29)	7.00 (1.05)
audiology	0.79 (0.02)	0.68 (0.02)	0.65 (0.08)	37.80 (4.13)	6.30 (0.82)
autos	0.79 (0.04)	0.69 (0.04)	0.64 (0.1)	52.80 (8.28)	4.80 (1.55)
balance-scale	0.78 (0.03)	0.78 (0.02)	0.74 (0.06)	15.80 (3.55)	6.00 (0.94)
colic	0.88 (0.02)	0.85 (0.01)	0.83 (0.05)	36.60 (5.15)	4.10 (0.74)
credit-g	0.75 (0.02)	0.79 (0.01)	0.70 (0.04)	111.30 (57.93)	6.00 (1.05)
diabetes	0.81 (0.02)	0.72 (0.02)	0.74 (0.05)	13.00 (3.53)	4.20 (1.4)
glass	0.81 (0.02)	0.77 (0.03)	0.68 (0.07)	24.20 (3.43)	5.80 (0.92)
heart-c	0.83 (0.02)	0.81 (0.02)	0.78 (0.08)	27.90 (5.45)	4.90 (0.57)
heart-statlog	0.86 (0.02)	0.82 (0.02)	0.73 (0.07)	13.60 (4.53)	4.10 (0.74)
hepatitis	0.89 (0.03)	0.86 (0.01)	0.77 (0.09)	14.80 (3.05)	4.60 (0.52)
ionosphere	0.93 (0.01)	0.94 (0.01)	0.90 (0.05)	12.00 (4.14)	4.90 (1.52)
iris	0.99 (0.01)	0.95 (0.01)	0.95 (0.05)	6.60 (0.84)	2.80 (0.42)
labor	0.88 (0.05)	0.95 (0.02)	0.80 (0.19)	5.60 (0.97)	2.00 (0)
lymph	0.87 (0.03)	0.81 (0.02)	0.77 (0.11)	21.60 (4.17)	3.60 (0.97)
primary-tumor	0.51 (0.02)	0.49 (0.01)	0.39 (0.05)	55.60 (5.04)	9.60 (0.97)

BeamClassifier não venceram em nenhum caso, porém seus resultados ficaram em torno da média dos outros algoritmos para a maioria das bases de dados. A mesma afirmação não pode ser feita para o algoritmo GALE que, embora tenha obtido o melhor resultado para uma base de dados, ficou abaixo da média para a maioria das bases de dados, especialmente em anneal, audiology, autos, glass, labor e primary-tumor.

Tabela 5.10: Médias das acurácias preditivas (%) e os respectivos desvios-padrão.

	LEGAL-DS	LEGAL-BC	LEGAL-ST	GALE	J48	CART	BeamClassifier
anneal	0.94 (0.03)	0.97 (0.01)	0.98 (0.02)	0.84 (0.03)	0.98 (0.01)	**0.99 (0.01)**	0.96 (0.02)
audiology	0.69 (0.05)	0.72 (0.02)	0.69 (0.08)	0.44 (0.09)	**0.78 (0.08)**	0.73 (0.09)	0.65 (0.08)
autos	0.73 (0.08)	0.76 (0.04)	0.66 (0.11)	0.51 (0.07)	**0.82 (0.07)**	0.75 (0.1)	0.64 (0.1)
balance-scale	0.78 (0.03)	0.77 (0.03)	0.76 (0.04)	**0.79 (0.04)**	0.77 (0.04)	**0.79 (0.04)**	0.74 (0.06)
colic	0.85 (0.01)	**0.86 (0.00)**	0.85 (0.05)	0.82 (0.02)	0.85 (0.05)	0.86 (0.05)	0.83 (0.05)
credit-g	0.71 (0.03)	0.72 (0.02)	0.72 (0.03)	0.71 (0.03)	0.71 (0.03)	**0.74 (0.04)**	0.70 (0.04)
diabetes	0.74 (0.03)	**0.76 (0.02)**	0.75 (0.04)	0.74 (0.02)	0.74 (0.05)	0.76 (0.04)	0.74 (0.05)
glass	0.69 (0.08)	0.70 (0.05)	0.68 (0.08)	0.59 (0.09)	0.67 (0.08)	**0.72 (0.08)**	0.68 (0.07)
heart-c	0.78 (0.05)	0.78 (0.03)	0.78 (0.09)	0.77 (0.06)	0.78 (0.08)	**0.82 (0.08)**	0.78 (0.08)
heart-statlog	**0.80 (0.05)**	0.78 (0.04)	0.77 (0.08)	0.79 (0.05)	0.77 (0.09)	0.79 (0.07)	0.73 (0.07)
hepatitis	0.82 (0.07)	0.79 (0.03)	0.79 (0.07)	0.80 (0.05)	**0.84 (0.07)**	0.78 (0.04)	0.77 (0.09)
ionosphere	0.90 (0.04)	0.90 (0.01)	0.89 (0.06)	0.89 (0.03)	**0.91 (0.03)**	0.89 (0.05)	0.90 (0.05)
iris	0.95 (0.01)	0.95 (0.01)	0.95 (0.05)	0.95 (0.03)	**0.96 (0.05)**	0.95 (0.05)	0.95 (0.05)
labor	**0.82 (0.12)**	0.80 (0.07)	0.78 (0.15)	0.68 (0.19)	0.74 (0.21)	0.76 (0.19)	0.80 (0.19)
lymph	**0.78 (0.06)**	0.76 (0.04)	0.76 (0.11)	0.77 (0.07)	0.77 (0.1)	0.76 (0.08)	0.77 (0.11)
primary-tumor	0.39 (0.05)	**0.40 (0.04)**	0.34 (0.03)	0.31 (0.04)	0.40 (0.05)	0.39 (0.06)	0.39 (0.05)

As Tabelas 5.11 e 5.12 mostram os casos em que as versões de LEGAL-Tree foram significativamente superiores e inferiores, respectivamente, aos outros algoritmos de acordo com o teste-t pareado corrigido. Cada célula $C_{i,j}$ representa a comparação entre o algoritmo da linha i com o algoritmo da coluna j. Cada célula possui uma sigla referenciando a base de dados em que o algoritmo i é significativamente diferente do algoritmo j: anneal (A_n), audiology (A_d), autos (A_t), balance-scale (B), colic (C_o), credit-g (C_g), diabetes (D), glass (G), heart-c (H_c), heart-statlog (H_s), hepatitis (H_e), ionosphere (I_o), iris (I_r), labor (L_a), lymph (L_y) e primary-tumor (P). Um hífen é usado quando a diferença não é significativa.

Em uma primeira análise sobre a Tabela 5.11, fica bastante evidente a superioridade

de LEGAL-Tree em relação ao algoritmo GALE. Essa constatação independe da versão de LEGAL-Tree. No entanto, não há nenhum caso em que LEGAL-Tree foi significativamente superior aos algoritmos J48 e SimpleCart. Em poucos casos LEGAL-Tree superou significativamente o BeamClassifier, caracterizando um empate técnico entre esses algoritmos em termos de acurácia. Na Tabela 5.12, em poucas situações os algoritmos J48 e SimpleCart superaram significativamente LEGAL-Tree em termos de acurácia. Com LEGAL-Tree sendo comparado ao J48, isso ocorreu em 3 bases de dados (anneal, audiology e autos) quando comparado com a versão LEGAL-DS, em apenas 1 base (anneal) quando comparado com a versão LEGAL-BC e novamente em 3 casos (audiology, autos e primary-tumor) quando comparado com a versão LEGAL-ST. Da mesma forma, com LEGAL-Tree sendo comparado ao SimpleCart, a versão LEGAL-DS foi superada em apenas 2 bases de dados (anneal e credit-g), a versão LEGAL-BC foi superada em somente 1 base de dados (anneal) e a versão LEGAL-ST foi significativamente superada por SimpleCart em apenas 1 base de dados, a autos. Vale destacar que, exceto a base de dados credit-g, em que apenas a versão LEGAL-DS foi superada por SimpleCart, as outras 4 bases de dados (anneal, audiology, autos e primary-tumor), nas quais LEGAL-Tree foi superado por J48 e SimpleCart, são as que possuem o maior número de classes (6, 24, 7 e 22, respectivamente) dentre as bases analisadas. Isso pode ser considerado uma evidência de que LEGAL-Tree não se comporta tão bem quanto os algoritmos J48 e SimpleCart, em termos de acurácia, quando lhe é apresentada como entrada uma base de dados cujo número de classes é grande. Em relação às diferenças significativas entre LEGAL-Tree e BeamClassifier, não foi observado nenhum padrão em relação às características das bases de dados.

Tabela 5.11: Considerando a acurácia para o conjunto de teste, são apresentadas as bases de dados em que LEGAL-DS, LEGAL-BC e LEGAL-ST são significativamente melhores que as outras abordagens de acordo com o teste-t.

	GALE	J48	CART	BC
LEGAL-DS	$A_n A_d A_t - C_o - -G$	$- - - - - - - - -$	$- - - - - - - - -$	$- - - - B_a - - -$
	$- - - - - L_a - P$	$- - - - - - - - -$	$- - - - - - - - -$	$- - H_s H_e - - - -$
LEGAL-BC	$A_n A_d A_t - C_o - -G$	$- - - - - - - - -$	$- - - - - - - - -$	$-A_d - - - -C_g-$
	$- - - - - L_a - P$	$- - - - - - - - -$	$- - - - - - - - -$	$- - - - - - - -$
LEGAL-ST	$A_n A_d A_t - - - -G$	$- - - - - - - - -$	$- - - - - - - - -$	$- - - - - - - -$
	$- - - - - L_a - P$	$- - - - - - - - -$	$- - - - - - - - -$	$- - - - - - - -$

O teste de Friedman registrou diferença significativa entre os métodos com $p = 0.0005534$. Assim, foi executado o teste de Nemenyi para comparar os algoritmos par a par. Esse teste registrou apenas quatro diferenças significativas em termos de acurácia, que são as vitórias de LEGAL-BC, J48 e SimpleCart sobre GALE, além da vitória de SimpleCart sobre BeamClassifer. Isso confirma o que havia sido evidenciado anteriormente, a superioridade desses algoritmos em relação ao GALE em termos de acurácia. O destaque, aqui, fica para o algoritmo BeamClassifier que, embora tenha

Tabela 5.12: Considerando a acurácia para o conjunto de teste, são apresentadas as bases de dados em que LEGAL-DS, LEGAL-BC e LEGAL-ST são significativamente piores que as outras abordagens de acordo com o teste-t.

	GALE	J48	CART	BC
LEGAL-DS	$--------$	$A_nA_dA_t-----$	A_n----C_g--	$--------$
	$--------$	$--------$	$--------$	$--------$
LEGAL-BC	$--------$	$A_n-------$	$A_n-------$	$--------$
	$--------$	$--------$	$--------$	$--------$
LEGAL-ST	$--------$	$-A_dA_t-----$	$--A_t-----$	$--------$
	$--------$	$--------P$	$--------$	$--------$

sido superado pelo SimpleCart, não foi inferior ao J48, um algoritmo já consolidado na literatura.

A Tabela 5.13 apresenta o tamanho das árvores, com os respectivos desvios-padrão, para todos os algoritmos do experimento. É possível notar que o algoritmo GALE gerou árvores menores em 11 das 16 bases de dados, perdendo apenas nas bases de dados balance-scale, heart-c, heart-statlog, hepatitis e ionosphere. No entanto, como já foi observado, o GALE apresentou acurácias significativamente menores e, portanto, foi considerado que ele está eliminado da "disputa", uma vez que a avaliação em termos de interpretabilidade só é importante quando não há perda de acurácia. Por essa razão, os valores em negrito na Tabela 5.13 registram os melhores resultados, em termos de tamanho de árvore, desconsiderando o GALE.

A primeira observação que se pode fazer ao analisar a Tabela 5.13 é que o algoritmo J48 não venceu em nenhum caso. No entanto, o algoritmo SimpleCart gerou as menores árvores em 9 bases de dados, enquanto que o algoritmo BeamClassifier foi superior em 4 casos. Já o algoritmo LEGAL-Tree gerou as menores árvores em apenas 3 bases da dados, uma para cada versão (LEGAL-DS, LEGAL-BC e LEGAL-ST). Em relação aos valores absolutos de tamanho da árvore, não foi observado nenhum padrão nas bases de dados para constatar se algum algoritmo se comporta melhor em determinados tipos de bases de dados.

Em termos de tamanho das árvores de decisão geradas, as Tabelas 5.14 e 5.15 apresentam os casos em que as versões de LEGAL-Tree foram significativamente superiores e inferiores, respectivamente, aos outros algoritmos de acordo com o teste-t pareado corrigido.

A Tabela 5.14 mostra claramente a superioridade de LEGAL-Tree, independente da versão, sobre o algoritmo J48. Tanto LEGAL-BC quanto LEGAL-ST geraram árvores significativamente menores que o J48 em 12 bases de dados, ou seja, 75% dos casos. LEGAL-DS superou significativamente o J48 em 11 casos. Quando comparado ao BeamClassifier, LEGAL-Tree foi superior em poucos casos. LEGAL-DS foi significativamente superior a BeamClassifier em 4 bases de dados (anneal, audiology, colic e primary-tumor), enquanto que LEGAL-BC e LEGAL-ST geraram árvores

Tabela 5.13: Médias dos tamanhos das árvores (número de nodos) e os respectivos desvios-padrão.

	LEGAL-DS	LEGAL-BC	LEGAL-ST	GALE	J48	CART	BeamClassifier
anneal	21.14	22.20	38.60	5.42	50.10	**21.40**	66.40
	(4.51)	(1.83)	(4.77)	(1.93)	(5.30)	**(1.74)**	(12.29)
audiology	**28.19**	39.88	40.46	7.17	48.90	34.40	37.80
	(7.92)	(5.08)	(5.06)	(3.36)	(3.65)	(9.76)	(4.13)
autos	82.46	78.47	57.62	12.73	62.00	**44.60**	52.80
	(19.49)	(10.71)	(8.78)	(4.70)	(6.72)	**(6.31)**	(8.28)
balance-scale	41.43	33.37	28.07	29.93	78.20	69.20	**15.80**
	(18.47)	(11.97)	(4.68)	(10.78)	(11.39)	(32.4)	**(3.55)**
colic	7.16	**6.12**	11.96	4.00	8.10	7.40	36.60
	(2.03)	**(0.27)**	(4.75)	(1.1)	(2.02)	(4.54)	(5.15)
credit-g	43.63	39.24	46.22	22.09	117.40	**28.00**	111.30
	(24.13)	(15.2)	(19.63)	(13.11)	(28.20)	**(15.34)**	(57.93)
diabetes	23.70	22.28	22.17	11.41	37.40	**12.00**	13.00
	(19.00)	(10.89)	(3.40)	(6.96)	(12.39)	**(7.11)**	(3.53)
glass	48.48	43.49	28.63	14.05	44.20	**18.40**	24.20
	(9.20)	(5.48)	(7.35)	(5.26)	(5.23)	**(10.12)**	(3.43)
heart-c	31.24	19.96	17.98	12.81	43.30	**12.40**	27.90
	(11.26)	(6.48)	(1.65)	(6.41)	(11.33)	**(3.69)**	(5.45)
heart-statlog	17.33	14.77	15.41	21.78	33.80	**11.80**	13.60
	(7.00)	(3.85)	(3.00)	(9.63)	(3.71)	**(2.99)**	(4.53)
hepatitis	16.48	9.06	9.29	6.71	17.80	**4.80**	14.80
	(4.82)	(2.63)	(5.28)	(3.03)	(4.40)	**(6.23)**	(3.05)
ionosphere	12.25	12.31	18.33	12.31	27.40	12.60	**12.00**
	(4.84)	(2.47)	(3.51)	(5.23)	(3.56)	(7.58)	**(4.14)**
iris	6.64	6.96	8.02	6.43	8.40	7.60	**6.60**
	(1.05)	(0.41)	(0.91)	(1.73)	(0.92)	(2.01)	**(0.84)**
labor	14.42	9.12	7.19	2.91	5.90	8.80	**5.60**
	(4.77)	(2.51)	(2.8)	(1.30)	(2.34)	(4.33)	**(0.97)**
lymph	22.73	18.35	17.71	12.95	28.30	**13.00**	21.60
	(7.04)	(4.35)	(3.03)	(6.17)	(3.49)	**(7.21)**	(4.17)
primary-tumor	50.02	41.90	**11.37**	8.48	83.70	31.20	55.60
	(18.52)	(15.52)	**(1.59)**	(4.65)	(6.65)	(9.48)	(5.04)

significativamente menores que BeamClassifier em 6 bases de dados (anneal, colic, credit-g, heart-c, hepatitis, e primary-tumor). Em comparação ao SimpleCart, LEGAL-Tree gerou árvores significativamente menores apenas em duas bases de dados: balance-scale (LEGAL-BC e LEGAL-ST) e primary-tumor (LEGAL-ST).

Por outro lado, ao observar a Tabela 5.15 é possível notar que em raros casos o J48 superou LEGAL-Tree. Isso ocorreu em apenas 2 bases de dados (autos e labor) quando comparado a LEGAL-DS, apenas 1 base (autos) quando comparado a LEGAL-BC e em nenhum caso quando comparado a LEGAL-ST. Entretanto, BeamClassifier superou significativamente LEGAL-Tree em alguns casos. Em relação ao LEGAL-DS, BeamClassifier gerou árvores significativamente menores em 5 bases de dados (autos, balance-scale, diabetes, glass e labor). Quando comparado ao LEGAL-BC, o algoritmo BeamClassifier foi significativamente superior em 5 bases de dados (autos, balance-scale, diabetes, glass e iris). Já em relação ao LEGAL-ST, BeamClassifier gerou árvores significativamente menores em apenas 3 situações (balance-scale, diabetes e ionosphere). Uma observação muito clara nessas tabelas é a "superioridade" do GALE em relação ao LEGAL-Tree em termos de tamanho das árvores. No entanto, como já mencionado, essa constatação não é relevante uma vez que LEGAL-Tree superou significativamente o GALE em termos de acurácia, critério mais prioritário. Por fim, quando LEGAL-Tree é comparado ao algoritmo SimpleCart, este gera árvores significativamente menores em 10 casos quando comparado à versão LEGAL-DS e em 6 casos quando comparado com a versão LEGAL-ST. Nesse caso, é aceitável preferir o SimpleCart a essas versões de LEGAL-Tree, principalmente à versão LEGAL-DS. No entanto, quando o algoritmo SimpleCart é comparado com a versão LEGAL-BC, o primeiro gera árvores significativamente menores apenas em 3 bases de dados (autos, glass e heart-c). Nesse caso, de acordo com o teste-t pareado corrigido, considera-se que LEGAL-BC é equivalente ao SimpleCart, uma vez que o superou em uma base de dados e foi superado em três. Considerando essas 3 bases de dados em que SimpleCart foi significativamente melhor que LEGAL-BC em termos de tamanho de árvore, surge outra evidência de que LEGAL-Tree pode, até o momento, não gerar bons resultados quando trabalha com bases de dados com várias classes, uma vez que autos possui 7 classes, glass possui 6 e heart-c tem 5 classes.

O teste de Friedman indicou diferença significativa entre os métodos utilizados, com $p = 1.005 \times 10^{-7}$. De acordo com o teste de Nemenyi, as seguintes diferenças foram confirmadas. Primeiro, que GALE gerou árvores menores que LEGAL-DS, LEGAL-BC, LEGAL-ST, J48 e BeamClassifier. Segundo, que LEGAL-BC, LEGAL-ST, SimpleCart e BeamClassifier geraram árvores significativamente menores que o J48. Supreendentemente, de acordo com Nemenyi, o GALE não superou significativamente o SimpleCart, em termos de tamanho das árvores, mesmo com uma visível superioridade de acordo com os valores absolutos. O destaque dos resultados apresentados pelo teste de Nemenyi fica para a LEGAL-Tree (exceto a versão LEGAL-DS) , BeamClassifier e SimpleCart, pois os três superaram J48 em termos de tamanho de árvore e, como

não perderam em acurácia, é possível concluir que, de forma geral, eles apresentaram resultados superiores ao algoritmo J48.

No entanto, de acordo com o teste de Friedman em termos de acurácia, o algoritmo SimpleCart superou o algoritmo BeamClassifier. Logo, pode-se concluir que as versões LEGAL-BC e LEGAL-ST, além do algoritmo SimpleCart, são as que apresentaram os melhores resultados em um contexto geral. Em especial, destaca-se LEGAL-BC e SimpleCart, uma vez que esses dois algoritmos apresentaram ganho significativo em termos de acurácia quando comparados ao algoritmo GALE.

Tabela 5.14: Considerando o tamanho das árvores, são apresentadas as bases de dados em que LEGAL-DS, LEGAL-BC e LEGAL-ST são significativamente melhores que as outras abordagens de acordo com o teste-t.

	GALE	J48	CART	BC
LEGAL-DS	$--------$	$A_nA_d-B_a-C_gD-$	$--------$	$A_nA_d--C_o---$
	$-H_s------$	$H_cH_s-I_oI_r-L_yP$	$--------$	$-------P$
LEGAL-BC	$--------$	$A_nA_d-B_a-C_gD-$	$---B----$	$A_n---C_oC_g--$
	$-H_s------$	$H_cH_sH_eI_oI_r-L_yP$	$--------$	H_c-H_e----P
LEGAL-ST	$--------$	$A_nA_d-B_a-C_gDG$	$---B----$	$A_n---C_oC_g--$
	$-H_s------$	$H_cH_sH_eI_o--L_yP$	$-------P$	H_c-H_e----P

Tabela 5.15: Considerando o tamanho das árvores, são apresentadas as bases de dados em que LEGAL-DS, LEGAL-BC e LEGAL-ST são significativamente piores que as outras abordagens de acordo com o teste-t.

	GALE	J48	CART	BC
LEGAL-DS	$A_nA_dA_tB_aC_oC_gDG$	$--A_t-----$	$--A_t--C_gDG$	$--A_tB_a--DG$
	$H_c-H_e--L_aL_yP$	$-----L_a--$	$H_cH_sH_e--L_aL_yP$	$-----L_a--$
LEGAL-BC	$A_nA_dA_t-C_oC_gDG$	$--A_t-----$	$--A_t----G$	$--A_tB_a--DG$
	$H_c----L_aL_yP$	$--------$	$H_c-------$	$-----I_r--$
LEGAL-ST	$A_nA_dA_t-C_oC_gDG$	$--------$	$A_n-A_t--C_gD-$	$---B_a--D-$
	$H_c--I_oI_rL_aL_yP$	$--------$	$H_cH_s------$	$---I_o----$

Embora tenham sido coletados e apresentados os resultados em termos de profundidade das árvores de decisão geradas, não será feita uma discussão desse critério para não tornar o texto repetitivo. Isso porque os resultados referentes às profundidades das árvores são equivalentes aos resultados referentes aos tamanhos das árvores, uma vez que essas medidas são diretamente proporcionais.

Para que fique mais claro quais algoritmos se comportam melhor para bases de dados específicas, as Figuras 5.1 e 5.2 ilustram os gráficos de barras para as acurácias preditivas. Analogamente, as Figuras 5.3 e 5.4 apresentam os gráficos de barras para o tamanho das árvores de decisão geradas pelos algoritmos para cada base de dados analisada.

5.3 Considerações finais

Este capítulo apresentou uma análise experimental do algoritmo LEGAL-Tree. Os resultados obtidos foram comparados com os algoritmos tradicionais para indução de árvores de decisão, o J48 e o SimpleCart, com o algoritmo evolutivo GALE (Llora & Garrell, 2001) e com o algoritmo BeamClassifier, o qual foi proposto também neste trabalho.

Os resultados mostraram que LEGAL-Tree pode gerar resultados equivalentes ao SimpleCart e superiores ao J48, especialmente quando se trata de bases de dados com poucas classes. A superioridade de LEGAL-Tree sobre o J48 se dá, especialmente, em termos de tamanho das árvores geradas, uma vez que gera árvores significativamente menores e, portanto, mais compreensíveis que as geradas pelo J48, e sem perda de acurácia. Em relação ao GALE, foi constatado que, embora tenha gerado árvores de decisão pequenas, suas respectivas acurácias foram significativamente inferiores que aqueles fornecidos pelos outros métodos.

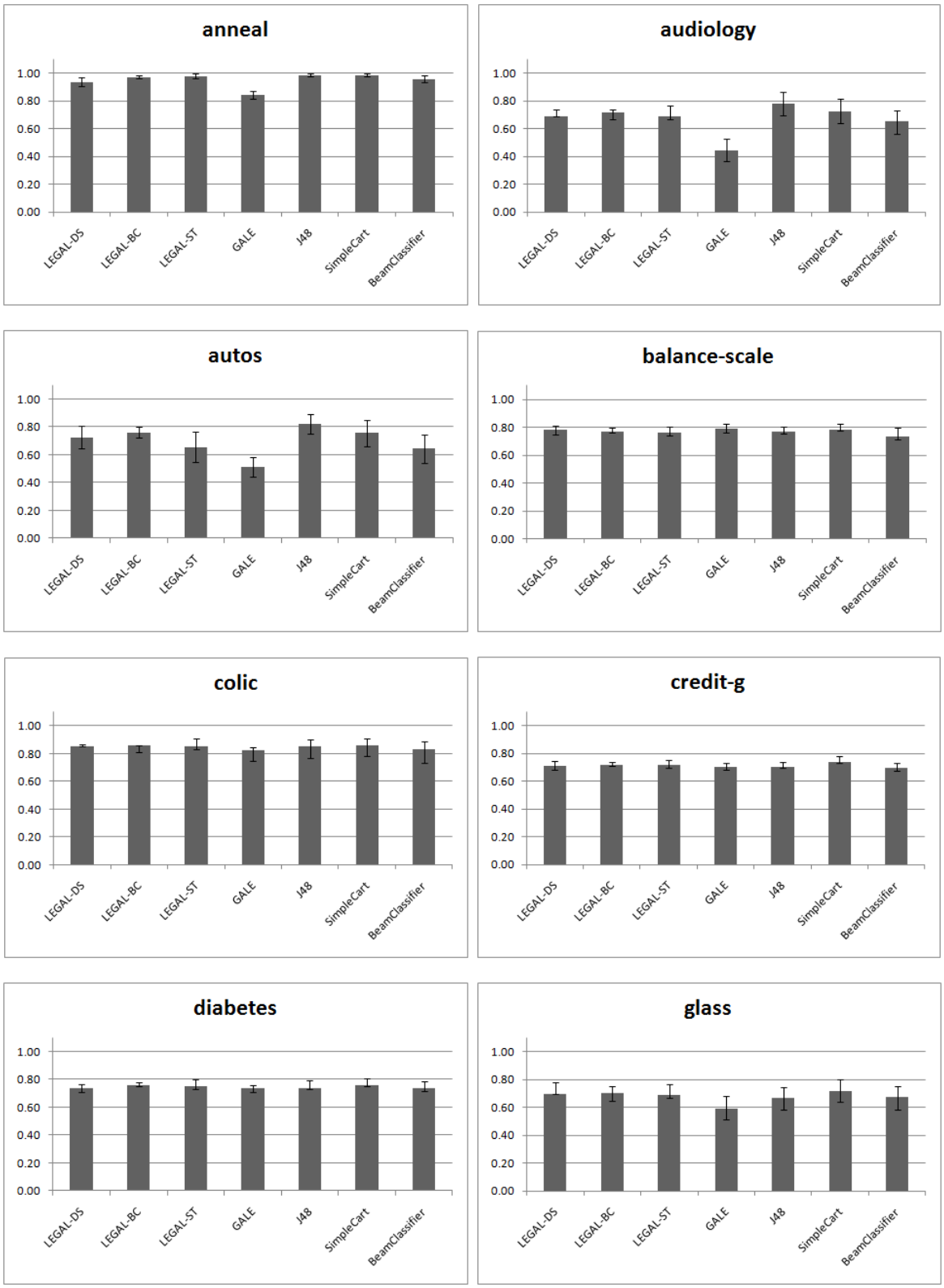


Figura 5.1: Gráfico de barras das acurácias das árvores de decisão geradas

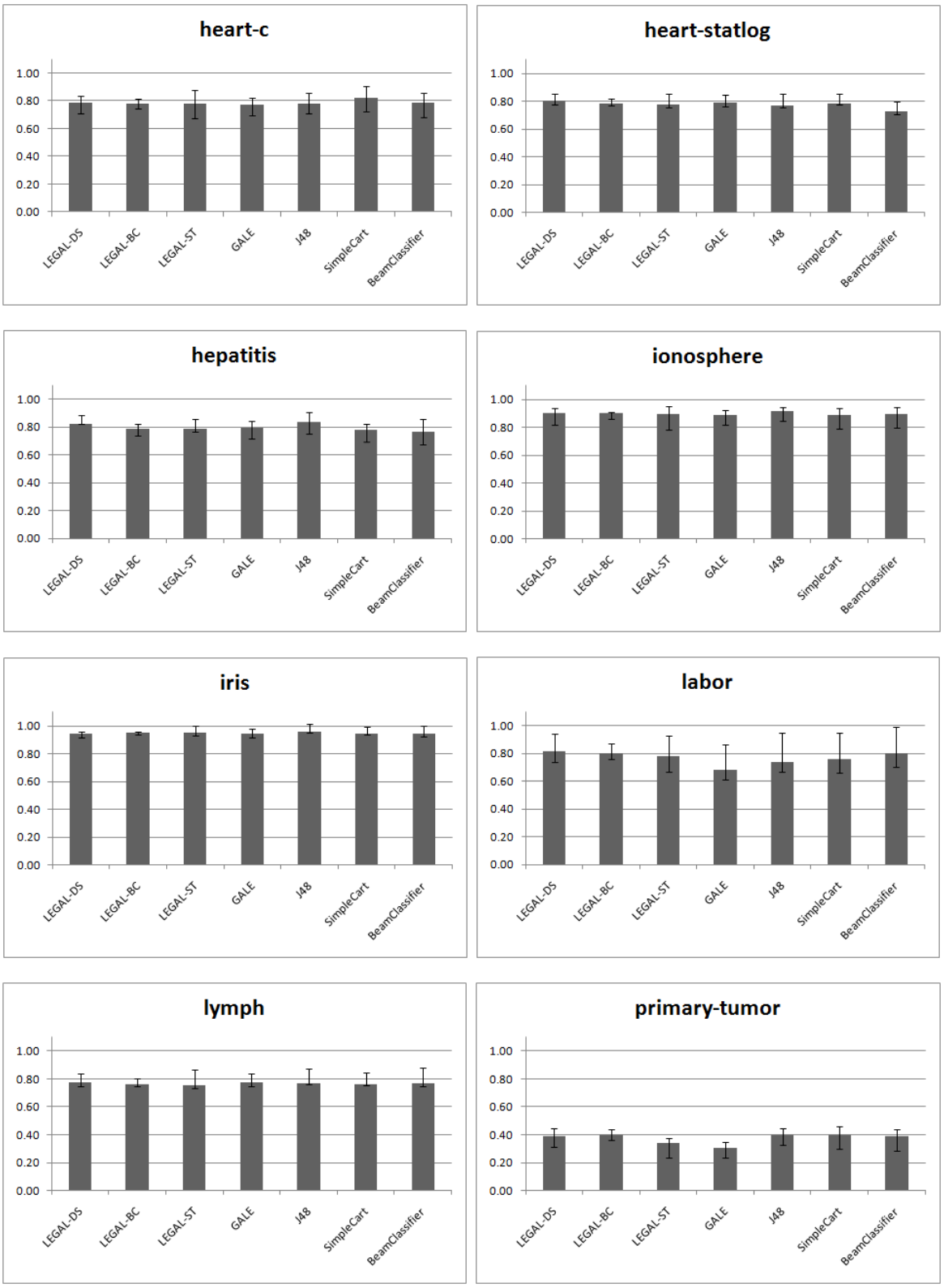


Figura 5.2: Gráfico de barras das acurácias das árvores de decisão geradas

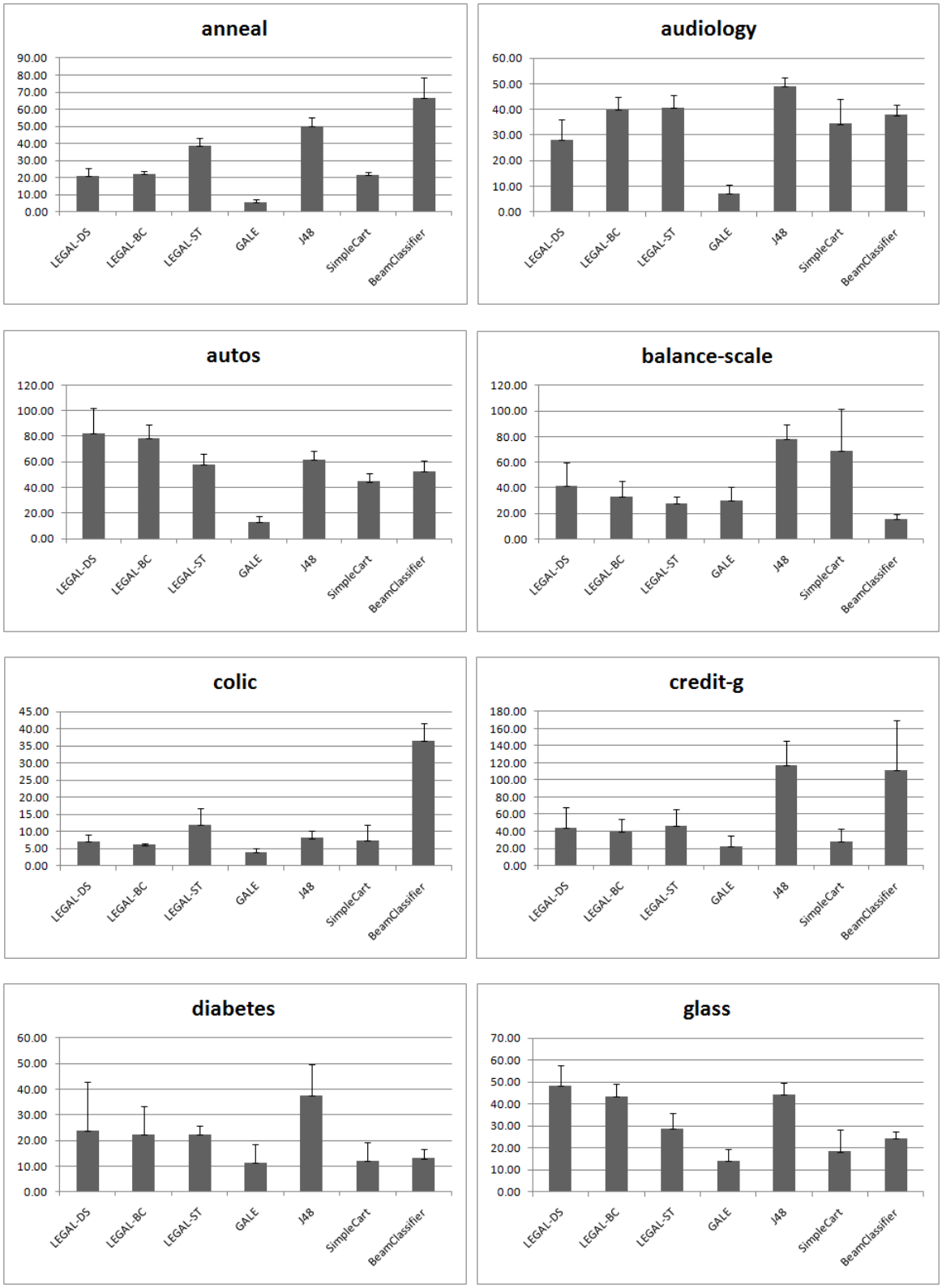


Figura 5.3: Gráfico de barras dos tamanhos das árvores de decisão geradas

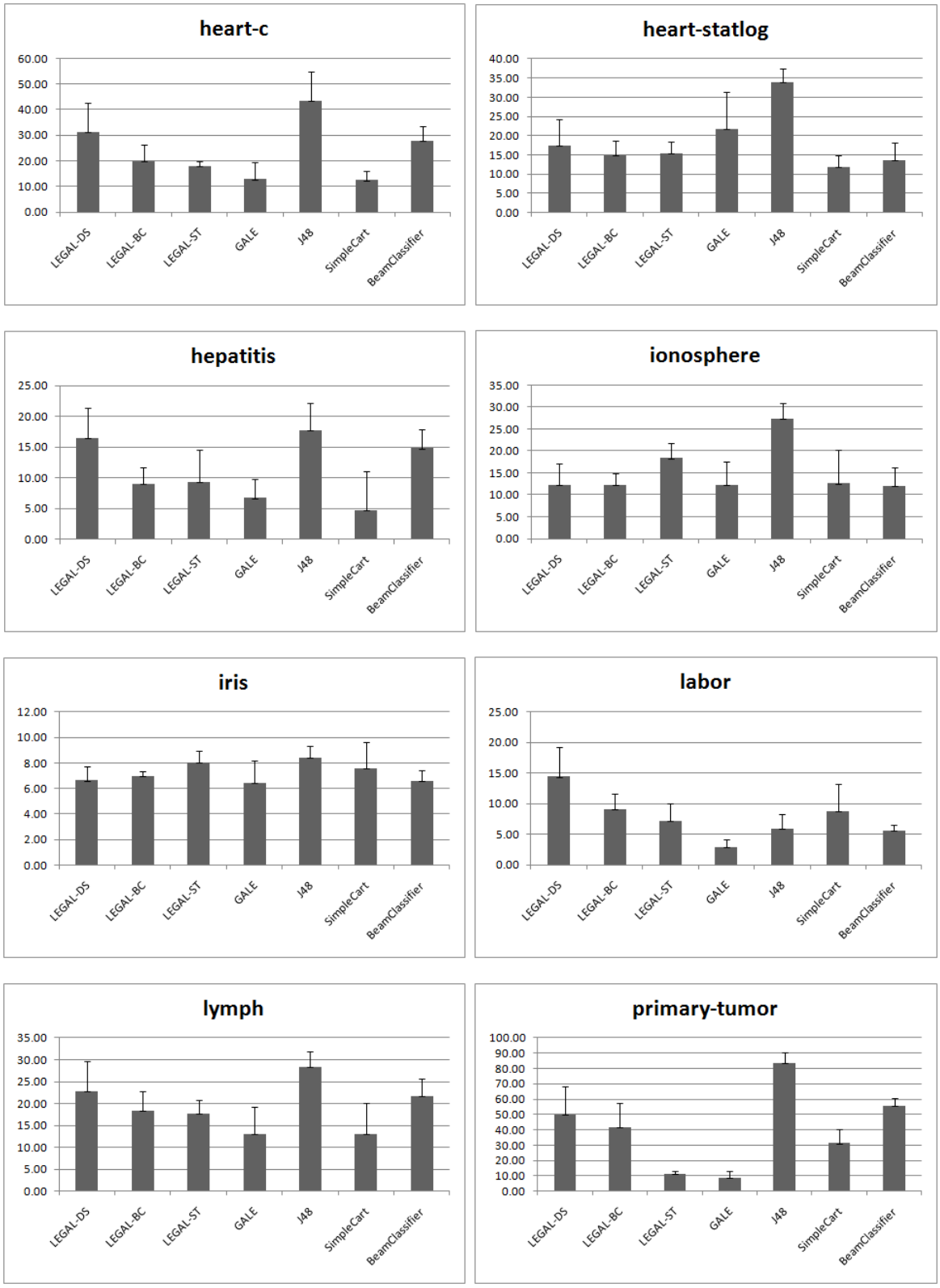


Figura 5.4: Gráfico de barras dos tamanhos das árvores de decisão geradas

Capítulo

6

Conclusão

Árvores de decisão têm sido amplamente utilizadas para indução de modelos que são fáceis de interpretar, principalmente porque esses modelos se assemelham ao raciocínio humano. Algoritmos de indução de AD geralmente usam uma estratégia gulosa, *top down* e com particionamento recursivo para a construção das árvores. Esses fatores implicam em convergência para ótimos locais ao invés de globais e na contribuição para *overffiting*, diminuindo a capacidade de generalização dos modelos. Com isso em mente, algoritmos evolutivos foram propostos na literatura, especialmente com a terminologia de programação genética. Embora essa não seja a nomeclatura apropriada (como discutido na Seção 3.7), esses trabalhos têm basicamente o mesmo objetivo do presente trabalho: evoluir soluções candidatas representadas por árvores de decisão com o intuito de convergir para soluções pelo menos próximas à ótima.

Este trabalho propôs um algoritmo evolutivo, mais especificamente um algoritmo genético chamado LEGAL-Tree. Este algoritmo difere essencialmente das propostas anteriores em um aspecto importante: a função de *fitness*. Enquanto as outras abordagem trabalham com apenas um objetivo ou então multi-objetivos baseados na fórmula ponderada ou dominância de Pareto, este trabalho propôs a utilização da abordagem lexicográfica, em que vários critérios podem ser considerados de acordo com suas respectivas prioridades para o domínio do problema em questão. Essa abordagem é relativamente simples de implementar e controlar, e também não sofre dos problemas relacionados às abordagens de fórmula ponderada e dominância de Pareto (Seção 4.4).

A função de *fitness* de LEGAL-Tree considera as duas medidas mais comuns para avaliação das árvores de decisão: acurácia e tamanho da árvore. Os resultados obtidos sobre 16 bases de dados de diversos problemas de classificação mostraram que LEGAL-Tree teve um desempenho equivalente ao desempenho do algoritmo SimpleCart

(implementação do CART (Breiman et al., 1984)) e superou o tradicional algoritmo J48 (implementação do C4.5 (Quinlan, 1993)), além de ter superado também o algoritmo evolutivo GALE (Llora & Garrell, 2001). A principal contribuição de LEGAL-Tree não foi gerar árvores com maior acurácia preditiva, mas sim gerar árvores menores e, portanto, mais compreensíveis ao usuário do que as outras abordagens, mantendo a acurácia preditiva equivalente. Isso mostra que LEGAL-Tree obteve sucesso na otimização lexicográfica de seus objetivos, uma vez que a idéia é justamente dar preferência às árvores menores (em termos de número de nodos) quando há equivalência de acurácia.

Esta é a primeira vez, até onde se sabe, que a abordagem lexicográfica é usada para indução evolutiva de árvores de decisão. Outras abordagens, como a de fórmula ponderada, têm sido utilizadas há bastante tempo e, conseqüentemente, têm tido uma boa quantidade de conhecimento prévio para dar suporte a elas. Por essa razão, essas abordagens mais tradicionais apresentam menos dificuldade na configuração de seus parâmetros. Mesmo assim, LEGAL-Tree apresentou resultados equivalentes à abordagem de fórmula ponderada em Basgalupp et al. (2009b), além de superar outros algoritmos, tanto tradicional (J48) como evolutivo (GALE). Esses resultados são encorajadores para o prosseguimento da utilização dessa abordagem.

6.1 Trabalhos futuros

A seguir, são apresentadas algumas idéia de trabalhos futuros, bem como melhorias que podem ser feitas em LEGAL-Tree.

A tarefa de definir os melhores parâmetros de algoritmos evolutivos em geral pode ser considerado também um problema de otimização. Logo, é possível utilizar um outro algoritmo evolutivo para otimizar os parâmetros de LEGAL-Tree de uma forma adequada para domínios de aplicação específicos. Dessa forma, seria possível encontrar valores para os parâmetros de LEGAL-Tree que gerem bons resultados para dados de expressão gênica, por exemplo.

Seria interessante desenvolver outros métodos de *crossover* e mutação, além de outros métodos para tratamento de valores desconhecidos. Embora uma das mutações implementadas em LEGAL-Tree possa ser considerada como um método de poda, pois reduz o tamanho da árvore, acredita-se que a implementação de um método de poda mais elaborado possa melhorar bastante os resultados de LEGAL-Tree. Essa é uma das prioridades para evolução desse algoritmo.

Pretende-se incorporar outras medidas na função de *fitness*, tais como medida-F, *precision* e AUC. Tais medidas permitem que seja feita uma análise mais minuciosa em determinados domínios de aplicação, especialmente em problemas de 2 classes em que há o interesse em falsos positivos e falsos negativos. Um exemplo desse tipo de aplicação seria a detecção de *spam*, em que não basta um classificador com alta taxa de acerto que, em certas situações, detecta um email importante como *spam* e o envia para a caixa de

spam. Isso caracteriza um falso positivo, o que deve ser evitado nessa situação, mesmo às custas da taxa de acerto.

Finalmente, pretende-se utilizar programação genética (PG) para evoluir algoritmos de indução de árvores de decisão. A idéia é evoluir algoritmos genéricos para indução de árvores de decisão, ou seja, o resultado final será um algoritmo de indução de árvores de decisão, e não apenas uma árvore de decisão. Nesse sentido, o algoritmo de PG que será desenvolvido pode ser considerado um algoritmo de *meta-learning*, pois o objetivo é aprender um algoritmo de aprendizagem. Dessa forma, se torna possível criar automaticamente algoritmos de indução de árvores de decisão customizados para as necessidades do usuário. Essa customização pode ser realizada com dois focos. O primeiro seria em relação a determinados domínios de aplicação, enquanto que o segundo foco seria relacionado às características de bases de dados. Essa idéia está altamente relacionada à função de *fitness*, pois é possível usar diferentes conjuntos de bases de dados para gerar os resultados das medidas utilizadas para computar o *fitness* dos indivíduos, sendo que cada um desses conjuntos pode representar um domínio de aplicação específico ou bases de dados com características semelhantes. Por exemplo, pode-se selecionar bases de dados da área de bioinformática, utilizá-las como entrada dos algoritmos de indução de árvores de decisão (indivíduos da PG) e utilizar as medidas obtidas das árvores de decisão resultantes para computar o *fitness* dos indivíduos da PG. Nesse caso, a PG resultaria num algoritmo "customizado" para indução de árvores de decisão que geram bons resultados para dados de bioinformática. O mesmo também poderia ser feito para outros domínios de aplicação. No entanto, é comum encontrar bases de dados de um mesmo domínio com características completamente diferentes e vice-versa, daí então a idéia de customizar para bases de dados com características semelhantes, a qual seria uma contribuição bastante significativa na área de aprendizado de máquina. Como resultado dessas customizações, seriam automaticamente gerados diversos novos algoritmos de indução de árvores de decisão, cada um deles sugerido para um domínio de aplicação específico ou então para bases de dados com determinadas características. Como motivação extra, pode-se citar o sucesso obtido na utilização de PG para evoluir algoritmos de indução de regras de classificação (Pappa & Freitas, 2006). Esse sucesso indica que a utilização da mesma técnica (PG) pode gerar resultados satisfatórios também para evoluir algoritmos de indução de árvores de decisão.

Referências

Alpaydin, E. (2004). *Introduction to Machine Learning: Adaptive Computation and Machine Learning*. The MIT Press.

Asuncion, A. & Newman, D. (2007). UCI machine learning repository.

Basgalupp, M., Barros, R., de Carvalho, A., Freitas, A., & Ruiz, D. (2009a). Legal-tree: a lexicographic multi-objective genetic algorithm for decision tree induction. In Shin, S., Ossowski, S., Martins, P., Menezes, R., Virol, M., Hong, J., Shin, D., Palakal, M., Fritzke, U., Crosby, M., & Haddad, H., editors, *Proceedings of the 2009 ACM Symposium on Applied Computing*, pag. 1085–1090. ACM Press.

Basgalupp, M., de Carvalho, A., Barros, R., Ruiz, D., & Freitas, A. (2009b). Lexicographic multi-objective evolutionary induction of decision trees. *International Journal of Bio-Inspired Computation*, 1(1/2):105–117.

Bot, M. C. J. & Langdon, W. B. (2000). Application of genetic programming to induction of linear classification trees. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J. F., Nordin, P., & Fogarty, T. C., editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802, pag. 247–258, Edinburgh. Springer-Verlag.

Bramer, M. (2007). *Principles of data mining*. Springer, London.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

Buntine, W. (1993). Learning classification trees. In Hand, D. J., editor, *Artificial Intelligence frontiers in statistics*, pag. 182–201. Chapman & Hall,London.

Cantu-Paz, E. & Kamath, C. (2000). Using evolutionary algorithms to induce oblique decision trees. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., & Beyer, H.-G., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pag. 1053–1060, Las Vegas, Nevada, USA. Morgan Kaufmann.

Carvalho, D. R. & Freitas, A. A. (2004). A hybrid decision tree/genetic algorithm method for data mining. *Inf. Sci.*, 163(1-3):13–35.

Chapelle, O., Schölkopf, B., & Zien, A. (2006). *Semi-supervised learning.* MIT Press.

Coello, C. A. C., Lamont, G. B., & Veldhuizen, D. A. V. (2006). *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection.* John Murray, London.

Deb, K. & Kalyanmoy, D. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, Inc., New York, NY, USA.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30.

Domingos, P. (1995). Rule induction and instance-based learning a unified approach. In *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*, pag. 1226–1232, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Domingos, P. (1999). The role of occam's razor in knowledge discovery. *Data Min. Knowl. Discov.*, 3(4):409–425.

E. Hunt, J. M. & Stone, P. (1966). *Experiments in Induction.* Academic Press, New York, USA.

Eiben, A. E. & Smith, J. E. (2003). *Introduction to Evolutionary Computing.* SpringerVerlag.

Estrada-Gil, J. K., Fernandez-Lopez, J. C., Hernandez-Lemus, E., Silva-Zolezzi, I., Hidalgo-Miranda, A., Jimenez-Sanchez, G., & Vallejo-Clemente, E. E. (2007). Gpdti: A genetic programming decision tree induction method to find epistatic effects in common complex diseases. *Bioinformatics*, 23(13):i167–174.

Fayyad, U. M., Piatetsky-Shapiro, G., & Smyth, P. (1996). Knowledge discovery and data mining: Towards a unifying framework. In *Knowledge Discovery and Data Mining*, pag. 82–88, Portland, Oregon, USA.

Fonseca, J. (1994). Indução de árvores de decisão. Master's thesis, Lisboa.

Freitas, A. A. (2004). A critical review of multi-objective optimization in data mining: a position paper. *SIGKDD Explorations*, 6(2):77–86.

Freitas, A. A. (2008). *Soft Computing for Knowledge Discovery and Data Mining*, capítulo A Review of evolutionary Algorithms for Data Mining, pag. 79–111. Springer US.

Freitas, A. A., Wieser, D. C., & Apweiler, R. (2008). On the importance of comprehensible classification models for protein function prediction. *To appear in IEEE/ACM Transactions on Computational Biology and Bioinformatics.*

Freund, Y. & Mason, L. (1999). The alternating decision tree learning algorithm. In *Proc. 16th International Conf. on Machine Learning*, pag. 124–133. Morgan Kaufmann, San Francisco, CA.

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. In *Annals of Math. Statistics*, volume 11, pag. 86–92.

Fu, Z., Golden, B., Lele, S., Raghavan, S., & Wasil, E. (2003a). Genetically engineered decision trees: Population diversity produces smarter trees. *Oper. Res.*, 51(6):894–907.

Fu, Z., Golden, B. L., Lele, S., Raghavan, S., & Wasil, E. (2006). Diversification for better classification trees. *Comput. Oper. Res.*, 33(11):3185–3202.

Fu, Z., Golden, B. L., Lele, S., Raghavan, S., & Wasil, E. A. (2003b). A genetic algorithm-based approach for building accurate decision trees. *INFORMS J. on Computing*, 15(1):3–22.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gray, J. B. & Fan, G. (2008). Classification tree analysis using target. *Comput. Stat. Data Anal.*, 52(3):1362–1372.

Han, J. (2001). *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Hong, S. J., Hosking, J., Hong, S. J., & Hosking, J. R. M. (1999). Partitioning nominal attributes in decision trees. *Data Mining and Knowledge Discovery*, 3:21–7.

Jong, K. A. D. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis.

Kohavi, R. (1996). Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pag. 202–207.

Koza, J. R. (1991). Concept formation and decision tree induction using the genetic programming paradigm. In *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pag. 124–128, London, UK. Springer-Verlag.

Kretowski, M. (2004). An evolutionary algorithm for oblique decision tree induction. In *ICAISC*, pag. 432–437.

Kretowski, M. & Grzes, M. (2005). *Information Processing and Security Systems*, capítulo Global learning of decision trees by an evolutionary algorithm, pag. 401–410. Springer US.

Landwehr, N., Hall, M., & Frank, E. (2005). Logistic model trees. *Mach. Learn.*, 59(1-2):161–205.

Llora, X. & Garrell, J. (2000). Automatic classification and artificial life models.

Llora, X. & Garrell, J. M. (2001). Evolution of decision trees. In *In Proceedings of the 4th Catalan Conference on Artificial Intelligence*, pag. 115–122. ACIA Press.

Loveard, T. & Ciesielski, V. (2001). Representing classification problems in genetic programming. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 2.

Loveard, T. & Ciesielski, V. (2002). Employing nominal attributes in classification using genetic programming. In Wang, L., Tan, K. C., Furuhashi, T., Kim, J.-H., & Yao, X., editors, *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02)*, pag. 487–491, Orchid Country Club, Singapore. IEEE.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Monard, M. C. & Baranauskas, J. A. (2003). Conceitos sobre aprendizado de máquina. In Rezende, S. O., editor, *Sistemas Inteligentes: Fundamentos e Aplicações*, capítulo 4, pag. 89–114. Editora Manole Ltda.

Nadeau, C. & Bengio, Y. (2003). Inference for the generalization error. *Mach. Learn.*, 52(3):239–281.

Nemenyi, P. (1983). *Distribution-Free Multiple Comparisons*. PhD thesis.

Pappa, G. L. & Freitas, A. A. (2006). Automatically evolving rule induction algorithms. In Fuernkranz, J., Scheffer, T., & Spiliopoulou, M., editors, *Proc. of the 17th European Conference on Machine Learning*, volume 4212/2006 of *Lecture Notes in Computer Science*, pag. 341–352, Berlin. Springer Berlin / Heidelberg.

Pappa, G. L., Freitas, A. A., & Kaestner, C. A. A. (2002). A multiobjective genetic algorithm for attribute selection. In Lofti, A., Garibaldi, J., & John, R., editors, *Proc. 4th Int. Conf. on Recent Advances in Soft Computing (RASC-2002)*, pag. 116–121. Nottingham Trent University.

Quinlan, J. (1988). Decision trees and multivalued attributes. *Machine Intelligence*, 11:305–318.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Quinlan, J. R. (1993). *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Quinlan, J. R. (1996). Bagging, boosting, and c4.5. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pag. 725–730. AAAI Press.

Rish, I. (2001). An empirical study of the naïve bayes classifier. In *proceedings of IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, Seattle.

Rokach, L. & Maimon, O. (2008). *Data mining with decision trees. Theory and applications.* World Scientific Publishing.

Russell, S. J. & Norvig (2003). *Artificial Intelligence: A Modern Approach (Second Edition).* Prentice Hall.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464.

Shi, H. (2007). Best-first decision tree learning. Master's thesis, University of Waikato, Hamilton, NZ. COMP594.

Shirasaka, M., Zhao, Q., Hammami, O., Kuroda, K., & Saito, K. (1998). Automatic design of binary decision trees based on genetic programming. In Newton, C., editor, *Second Asia-Pacific Conference on Simulated Evolution and Learning*, Australian Defence Force Academy, Canberra, Australia.

Tan, P.-N., Steinbach, M., & Kumar, V. (2005). *Introduction to Data Mining, (First Edition).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Tür, G. & Güvenir, H. A. (1996). Decision tree induction using genetic. In *Proceedings of the Fifth Turkish Symposium on Artificial Intelligence and Neural Networks*, pag. 187–196.

Wettschereck, D. (1994). A hybrid nearest-neighbor and nearest-hyperrectangle algorithm. In *ECML-94: Proceedings of the European conference on machine learning on Machine Learning*, pag. 323–335, Secaucus, NJ, USA. Springer-Verlag New York, Inc.

Witten, I. H. & Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann.

Z. Bandar, A.-A. & McLean, H. (1999). Genetic algorithm based multiple decision tree induction. In *proceedings of the 6th International Conference on Neural Information Processing*, pag. 429–434.

Zhao, H. (2007). A multi-objective genetic programming approach to developing pareto optimal decision trees. *Decis. Support Syst.*, 43(3):809–826.

Zhao, Q. & Shirasaka, M. (1999). A study on evolutionary design of binary decision trees. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., & Zalzala, A., editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pag. 1988–1993, Mayflower Hotel, Washington D.C., USA. IEEE Press.