Árvores de Classificação para Escolha de Estratégias de Operação em Mercados de Capitais

Marcelo de Souza Lauretto

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

Área de Concentração: **Ciência da Computação**
Orientador: **Prof. Dr. Júlio Michael Stern**

Durante a elaboração deste trabalho o autor recebeu apoio financeiro do CNPq

-São Paulo, setembro de 1996-

Árvores de Classificação para Escolha de Estratégias de Operação em Mercados de Capitais

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Marcelo de Souza Lauretto
e aprovada pela comissão julgadora.

São Paulo, 23 de setembro de 1996.

Banca examinadora:

- Prof. Dr. Júlio Michael Stern (orientador) - IME-USP
- Prof. Dr. Flávio Soares Correa da Silva - IME-USP
- Prof. Dr. Marcos Eugênio da Silva - FEA-USP

Resumo

Nesta dissertação damos um tratamento unificado aos algoritmos de classificação do tipo TDIDT (Top Down Induction of Decision Trees), incluindo o tratamento de atributos reais. A seguir propomos técnicas para utilizar esses algoritmos na avaliação e decisão de estratégias de aplicação no mercado financeiro. Os resultados são verdadeiramente encorajadores e motivam várias linhas para futuro desenvolvimento.

Abstract

In this dissertation we give a unified treatment to the classification algorithms of the TDIDT family (Top Down Induction of Decision Trees), including the treatment of real attributes. We then suggest some techniques to use these algorithms to evaluate and support financial markets strategies. The results we got are truly encouraging, motivating several possible future developments.

Ao Lucas.

Agradecimentos

Ao Prof. Júlio M. Stern, pelo constante apoio, pela preocupação profissional e pessoal, pela orientação firme e decidida; e principalmente pela grande paciência.

Ao Prof. Flávio S. C. da Silva, por sua co-orientação e por suas sugestões, as quais nos ajudaram (e muito) a definir as diretrizes do trabalho.

Ao Prof. José Augusto R. Soares, pela sua atenção, por nossas discussões sobre Universidade e sociedade, que certamente me foram muito valiosas; por seu grande esforço e preocupação com o Programa de Pós-Graduação.

Ao prof. Jacob Zimbarg Sobrinho, por nos ter gentilmente cedido as bases de dados, e pelo seu incentivo.

Às Prof^{as}. Maria Carolina Monard e Maria do Carmo Nicoletti, pela atenção, pela hospitalidade dispensada em minhas visitas ao ICMSC e pelas sugestões apresentadas; também por nos terem cedido a implementação do NewID.

À Celma de Oliveira Ribeiro, por ter cedido a implementação dos indicadores e, principalmente, pelo apoio, confiança e amizade.

Ao Fábio Nakano, pelas implementações do Cal5 e do Real e pelas numerosas dicas.

Ao CNPq, pelo apoio financeiro que viabilizou este trabalho.

Aos meus pais, pela eterna dedicação, apoio e amor.

Aos numerosos e queridos amigos.

A todos, meu **muito obrigado**.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação e Objetivos

No mercado de ações, é desejo de todo investidor obter indicadores das tendências de preços que possam auxiliá-lo em suas decisões, isto é, na escolha das estratégias de operação. Por exemplo, antes de comprar um determinado título, o investidor busca evidências de que a cotação deste título deverá subir dentro de um certo período, permitindo-lhe uma margem de lucro que lhe pareça satisfatória na revenda do mesmo.

Uma maneira amplamente utilizada na escolha de estratégias de operação é a chamada *Análise Técnica*, que se baseia no comportamento das cotações dos papéis [dBdV85]. As ferramentas empregadas pelos analistas técnicos são os gráficos e alguns indicadores técnicos obtidos numericamente das cotações.

Atualmente, o volume de papéis disponíveis no mercado, a volatilidade de suas cotações e a necessidade de decisões rápidas têm incrementado a demanda por sistemas computacionais de apoio à decisão para escolha de estratégias. É particularmente importante que os resultados de um sistema desse tipo sejam fornecidos em tempo hábil, isto é, suficientemente curto para que o investidor realize suas operações com uma vantagem comparativa sobre o restante do mercado. Todavia, os métodos de programação tradicionais podem ser limitados, pois cabe ao analista técnico fornecer todos os critérios de decisão, o que nem sempre é possível.

Dificuldades dessa natureza são uma das motivações para as pesquisas no *Aprendizado por Exemplos*, comumente denominado *Aprendizado Indutivo*. Em linhas gerais, um sistema de aprendizado indutivo recebe do especialista um conjunto de exemplos que tenham sido classificados previamente (denominado *conjunto de treinamento*) e induz, a partir desses exemplos, uma teoria (ou uma generalização) capaz de ser utilizada em novas situações.

Dentre os algoritmos de aprendizado indutivo, destacam-se os algoritmos do chamado *Aprendizado Proposicional* [Nic94]. Para os algoritmos dessa categoria, os exemplos são representados através de seus atributos (por exemplo, *cor*, *altura*, *peso*), e a teoria obtida é representada na forma de *regras de produção* ou de *árvores de decisão* (comumente chamadas *árvores de classificação*).

O interesse pela aplicação dos algoritmos de aprendizado proposicional vem se intensificando nos últimos tempos. Por exemplo, o projeto "Statlog", integrante do programa ESPRIT da Comunidade Européia, analisou e comparou vários algoritmos estatísticos, de redes neurais e de aprendizado proposicional sob diversas aplicações reais (análise de créditos, reconhecimento de dígitos e letras, diagnósticos médicos, entre outros) [MST94].

Em nosso trabalho, enfocamos os algoritmos da família TDIDT (Top-Down Induction of Decision Trees), derivados do algoritmo ID3 (Interactive Dichotomizer 3) [Qui79] e de seu variante, o algoritmo CART [BFOS84]. Um algoritmo dessa família é constituído basicamente de duas fases:

- A expansão da árvore, através de sucessivas partições do conjunto de treinamento;
- A eliminação de algumas partes inferiores da árvore, através de reagrupamentos dos sub-conjuntos da partição, de maneira a manter apenas os atributos mais relevantes na árvore e atenuar o efeito de ruídos nos dados disponíveis; esse procedimento é denominado *poda* da árvore.

As duas principais vantagens desses algoritmos são:

- As árvores obtidas são computacionalmente eficientes para a classificação de novas configurações. O procedimento de classificação consiste em testes simples dentro de um único caminho desde a raíz até uma das folhas, gastando portanto tempo proporcional à altura da árvore. Ademais, somente os atributos relevantes - isto é, altamente correlacionados com as classes - aparecem nas árvores, o que as torna relativamente compactas.
- As árvores podem ser compreendidas e validadas pelos usuários, sem qualquer necessidade de compreensão do funcionamento interno dos algoritmos. Por sua clareza, as árvores de decisão podem, teoricamente, ampliar o conhecimento do especialista a respeito do domínio de aplicação. O especialista pode descobrir ou confirmar, através das árvores obtidas, quais os atributos mais relevantes no processo de classificação.

A despeito da aparente viabilidade prática dos algoritmos TDIDT no mercado financeiro, não possuímos ainda registros sobre sua aplicação nessa área no Brasil.

Os objetivos de nosso trabalho são:

1. Apresentar formalmente alguns algoritmos da família TDIDT dentro de uma abordagem unificada;

2. Propor um novo algoritmo, TDIDT, o Real;

3. Discutir e avaliar a aplicabilidade dos algoritmos TDIDT no suporte à definição de estratégias de operação no mercado financeiro, em termos da metodologia a ser apresentada.

Para atingir os objetivos propostos, inicialmente realizaremos testes sobre alguns sistemas já disponíveis: o Cal5[MW94], o NewID[Bos90] e o REAL[SNL96]. As cotações de alguns papéis serão utilizadas como bases para nossos testes.

1.2 Organização da Dissertação

No capítulo 2 serão apresentados os conceitos básicos sobre o Aprendizado Proposicional. Também será discutido o algoritmo geral de construção das árvores de classificação.

Nos dois capítulos seguintes, serão mostrados alguns métodos que implementam duas importantes fases no processo de construção das árvores. No capítulo 3 veremos alguns critérios para seleção dos atributos mais relevantes. No capítulo 4 serão apresentados alguns métodos de poda dos ramos.

No capítulo 5 apresentaremos os programas que foram efetivamente utilizados em nosso trabalho.

No capítulo 6 serão apresentadas técnicas de aplicação e avaliação dos algoritmos TDIDT no suporte à compra e venda de papéis, segundo uma certa estratégia de operação. Também são discutidos os resultados obtidos.

Capítulo 2

Aprendizado Proposicional e Árvores de Decisão

2.1 Considerações Iniciais

Árvores de decisão são estruturas que representam, em última instância, regras de decisão para determinar a classe de um dado objeto. Os algoritmos mais representativos pertencem à família TDIDT (Top-Down Induction of Decision Trees) e descendem do ID3 (Iteractive Dichotomizer 3) [Qui79] e de seu variante, o algoritmo CART [BFOS84]. Nosso objetivo neste capítulo é apresentar o algoritmo TDIDT.

A fim de fixarmos algumas notações, serão apresentados na seção 2.2 alguns conceitos básicos sobre o aprendizado proposicional, categoria à qual pertencem os algoritmos TDIDT [AB92, Ban90, BFOS84].

Para o leitor não familiarizado com a terminologia de árvores, a seção 2.3 apresenta os conceitos necessários [AHU83]. Na seção 2.4 definiremos formalmente as árvores de decisão e apresentaremos o algoritmo básico para sua construção.

2.2 O Aprendizado Indutivo de Máquina

Para sua melhor compreensão, o estudo da construção de árvores de decisão deve ser precedido por um breve estudo a respeito do aprendizado por exemplos, comumente denominado Aprendizado Indutivo. Em primeiro lugar, a noção de *conceito* ou *classe* precisa ser melhor compreendida sob a ótica de conjuntos.

Definição 2.2.1 *Seja U o conjunto universo de objetos, ou seja, o conjunto de todos os objetos que o aprendiz pode encontrar. Uma* classe *C pode ser definida como um subconjunto de U. Sob essa definição,* aprender uma classe *C significa aprender a identificar os elementos de C.*

Em nosso trabalho, adotaremos apenas conjuntos finitos de classes. Sob essa premissa, o conjunto universo U será particionado em J classes $C_1, C_2, \ldots C_J$, para algum $J \geq 1$. Por questão de notação, denotaremos as classes por números de 1 a J.

2.2.1 A Representação de Objetos por Atributos

Os algoritmos estudados ao longo desse trabalho utilizam a descrição de objetos através de atributos. Um conjunto $A = a_1, a_2, \ldots, a_m$ de atributos (ou variáveis) deve ser previamente definido. Após definido o conjunto de atributos, cada objeto do domínio será descrito através de um *vetor de medidas* (também denominado vetor de atributos) $X = [x_1, x_2, \ldots, x_m]$, onde x_i é o valor do atributo a_i, $i = 1, \ldots, m$.

Os atributos classificam-se em dois tipos:

tipo numérico ou ordenado : um atributo numérico ou ordenado assume valores dentro dos números reais. Por exemplo: idade, peso, comprimento;

tipo categórico : um atributo categórico assume valores dentro de um conjunto finito, onde não há nenhuma ordem natural. Por exemplo: cor.

No aprendizado por atributos, o conjunto U citado no início deste capítulo é denominado *espaço de medidas*, e é composto por todos os possíveis vetores de medidas X que podem ser definidos sobre o conjunto de atributos A.

2.2.2 O Aprendizado a Partir de Exemplos

A fonte de informações para um sistema de aprendizado é um conjunto de exemplos E, tambem denominado *conjunto de treinamento*. O conjunto de treinamento E é um subconjunto de U cujos elementos tenham sido previamente classificados e que serão utilizados por um algoritmo de aprendizado. Em outras palavras, cada exemplo do conjunto de treinamento é um vetor de medidas X ao qual está associada uma classe, $C(X)$.

Como ilustração, consideremos o problema de diagnósticos. Os exemplos serão casos clínicos previamente diagnosticados por médicos especialistas. Assim, X será um vetor contendo uma série de parâmetros (incluindo sintomas e resultados de exames) do paciente em questão e $C(X)$ será a doença diagnosticada para aquele paciente.

A partir do conjunto de treinamento E, o problema geral do aprendizado é encontrar, para cada classe j representada em E, uma regra $Desc(j)$ que descreva essa classe.

O conjunto de descrições $Desc(1), Desc(2), \ldots, Desc(J)$ obtidas constituirá uma função $d : U \rightarrow \{1, 2, \ldots, J\}$, isto é, uma função que associa a cada vetor de medidas X uma classe j.

O *Aprendizado Proposicional* é uma modalidade de aprendizado em que os exemplos são representados por meio de vetores de atributos e a teoria gerada (a função $d : U \rightarrow \{1, 2, \ldots, J\}$) é representada por meio de regras lógicas ou árvores de decisão.

Numa situação ideal, cada descrição $Desc(j)$ deve ser:

completa : todos os exemplos da classe j satisfazem à condição imposta por $Desc(j)$.

consistente : nenhum exemplo que não pertença à classe j satisfaz à condição imposta por $Desc(j)$.

Em domínios reais, é muito comum a presença de ruídos, ou seja, erros de medições e de classificações dos exemplos. Ademais, nem sempre se consegue um conjunto satisfatório de atributos para descrever os objetos. Em conseqüência, é comum que objetos de classes distintas sejam representados por vetores de medidas idênticos. Isso implica que as restrições acima precisam ser relaxadas, admitindo que exemplos de classes distintas sejam eventualmente descritos pelas mesmas regras.

Outro fato que deve ser levado em consideração é que não estamos interessados apenas em encontrar regras que descrevam os exemplos fornecidos, mas também (e principalmente) encontrar regras que classifiquem o mais corretamente possível objetos desconhecidos. Encontrar uma regra de produção para cada um dos vetores de atributos de E não é uma boa alternativa, pois essa regra estará utilizando atributos irrelevantes, sem qualquer poder preditivo e somente será útil para objetos que possuam vetores de atributos idênticos ao exemplo que gerou tal regra. Logo, as regras devem ser mais gerais do que os exemplos fornecidos, e devem ser construídas somente com os atributos relevantes para a predição das classes.

2.3 Terminologia Básica de Árvores

Uma árvore é uma coleção de elementos chamados *nós*, dentre os quais um é distinguido como uma *raíz*, juntamente com uma relação de "paternidade" que impõe uma estrutura hierárquica sobre os nós. Formalmente,

Definição 2.3.1 *Uma* árvore *pode ser definida recursivamente da seguinte maneira:*

1. *Um único nó é uma árvore. Este nó é também a raíz da árvore.*
2. *Suponha que t seja um nó e $T_1, T_2, \ldots, T_k$ sejam árvores com raízes $t_1, t_2, \ldots, t_k$, respectivamente. Podemos construir uma nova árvore transformando t no* pai *dos nós $t_1, t_2, \ldots, t_k$. Nessa árvore, t será a raíz e $T_1, T_2, \ldots, T_k$ serão as* sub-árvores *ou* ramos *da raíz. Os nós $t_1, t_2, \ldots, t_k$ são chamados* filhos *do nó t.*

Algumas vezes é conveniente incluir no conceito de árvores a *árvore nula* ou *vazia*, uma "árvore" sem nós.

Considere o exemplo da figura 2.1, em que é apresentado o sumário de um livro e sua representação como uma árvore. A raíz, chamada *Livro*, possui três sub-árvores cujas raízes correspondem aos capítulos $C1$, $C2$ e $C3$. *Livro* é o pai de $C1$, $C2$ e $C3$, e esses três nós são os filhos de *Livro*.

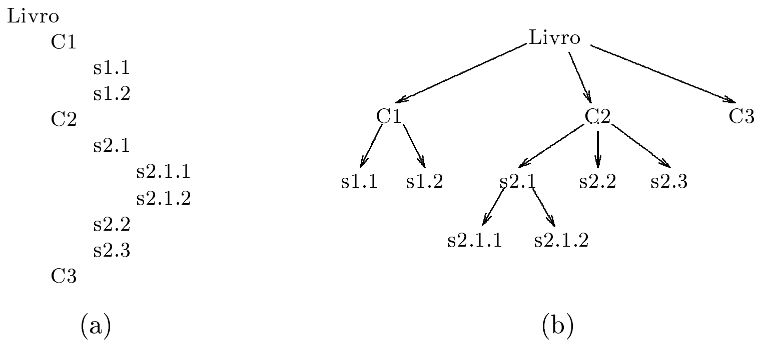


Figura 2.1: Sumário de um livro (a) e sua respectiva representação como uma árvore (b).

Se $t_1, t_2, \ldots, t_k$ é uma seqüência de nós em uma árvore tais que t_i é o pai de t_{i+1} para $1 \leq i < k$, então esta seqüencia é denominada um *caminho* do nó t_1 até o nó t_k. No exemplo anterior, a seqüência $C2, s2.1, s2.1.2$ é um caminho de $C2$ até $s2.1.2$.

Se existe um caminho do nó a ao nó b, então a é um *ancestral* de b e b é um *descendente* de a. Note que todo nó é ancestral e descendente de si mesmo. Um ancestral ou um descendente de um nó, que não seja o próprio nó, é denominado *ancestral próprio* ou *descendente próprio*, respectivamente.

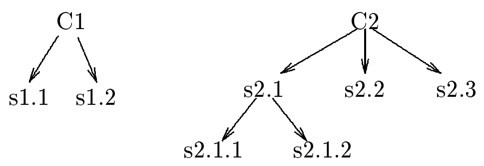


Figura 2.2: Ramos T_{C1} e T_{C2} da árvore descrita na figura 2.1.

Com as definições acima, podemos formalizar a definição de ramos: dado um nó $t \in T$, a *sub-árvore* ou *ramo* T_t de T consistirá do nó t (que será a raíz de T_t) juntamente com todos os descendentes de t em T. A figura 2.2 mostra os ramos T_{C1} e T_{C2} da árvore descrita na figura 2.1.

Todos os nós que não possuem filhos são chamados *nós terminais* ou *folhas*. Os nós que contêm filhos são chamados *nós não-terminais* ou *nós internos*. O conjunto das folhas de T será denotado por $\overline{T}$.

2.4 A Construção de Árvores de Decisão

Nesta seção será apresentada a definição formal de uma árvore de decisão. Será discutido também o algoritmo básico de construção, comum a todos os sistemas geradores de árvores de decisão utilizados neste trabalho.

Definição 2.4.1 [Utg89] *Uma* Árvore de Decisão *é:*

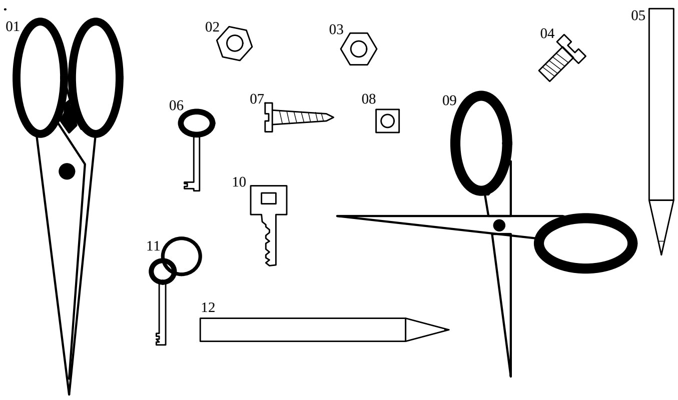
- *um nó folha (ou nó resposta) que contém o nome de uma classe ou o símbolo nulo (nulo indica que não é possível atribuir nenhuma classe ao nó por não haver nenhum exemplo que corresponda a esse nó); ou*
- *um nó interno (ou nó de decisão) que contém o nome de um atributo; para cada possível valor do atributo, corresponde um ramo para uma outra árvore de decisão.*

2.4.1 Exemplo de construção de uma árvore

A figura 2.3(a) mostra alguns objetos que serão utilizados como ilustração nesta seção. Os objetos da figura 2.3(a), numerados de 01 a 12, pertencem a cinco classes: porca, parafuso, chave, caneta, tesoura. Suponha que esses objetos sejam mostrados a um sistema de visão. Suas silhuetas são capturadas por uma câmera e processadas por um programa de visão. Esse programa pode então extrair alguns valores de atributos de cada imagem. Em nosso caso, os atributos são: o tamanho, o formato e o número de orifícios de um objeto. Os possíveis valores desses atributos são:

- *Tamanho*: $pequeno, grande$
- *Formato*: $longo, compacto, outro$
- *Orifícios*: $0, 1, 2, 3, muitos$

Assuma que o sistema de visão tenha extraído os valores dos três atributos para cada objeto. A figura 2.3(b) mostra a representação desses objetos através de seus respectivos vetores de atributos.



(a)

Objeto No.	TAMANHO	FORMATO	ORIFICIOS	CLASSE
01	grande	longo	2	tesoura
02	pequeno	compacto	1	porca
03	pequeno	compacto	1	porca
04	pequeno	longo	0	parafuso
05	grande	longo	0	caneta
06	pequeno	longo	1	chave
07	pequeno	longo	0	parafuso
08	pequeno	compacto	1	porca
09	grande	longo	2	tesoura
10	pequeno	longo	1	chave
11	pequeno	longo	2	chave
12	grande	longo	0	caneta

(b)

Figura 2.3: O Conjunto de treinamento E. (a) Imagem dos objetos captada por uma câmera. (b) Sua representação por atributos.

A figura 2.4 mostra uma árvore de decisão induzida a partir dos exemplos da figura 2.3. Descreveremos em linhas gerais o processo de construção dessa árvore.

Iniciamos a construção da árvore com apenas um nó (a raíz) e associamos a esse nó o conjunto E. Como os exemplos não estão na mesma classe, decidimos "dividir o nó", ou seja, expandir a árvore. Para isso, escolhemos um atributo (em nossa ilustração, *Orifícios*), com o qual rotulamos a raíz. Dividimos o conjunto E em subconjuntos disjuntos, agrupando exemplos de mesmo valor de *Orifícios* em cada subconjunto. Para cada um desses subconjuntos, criamos um novo nó-filho do nó *Orifícios*. Na figura 2.4, os subconjuntos de E são:

$$
\begin{aligned}
E_1 &= \{X \in E \text{ tq } X_{\mathit{Orifícios}} = 0\} \\
E_2 &= \{X \in E \text{ tq } X_{\mathit{Orifícios}} = 1\} \\
E_3 &= \{X \in E \text{ tq } X_{\mathit{Orifícios}} = 2\}
\end{aligned}
$$

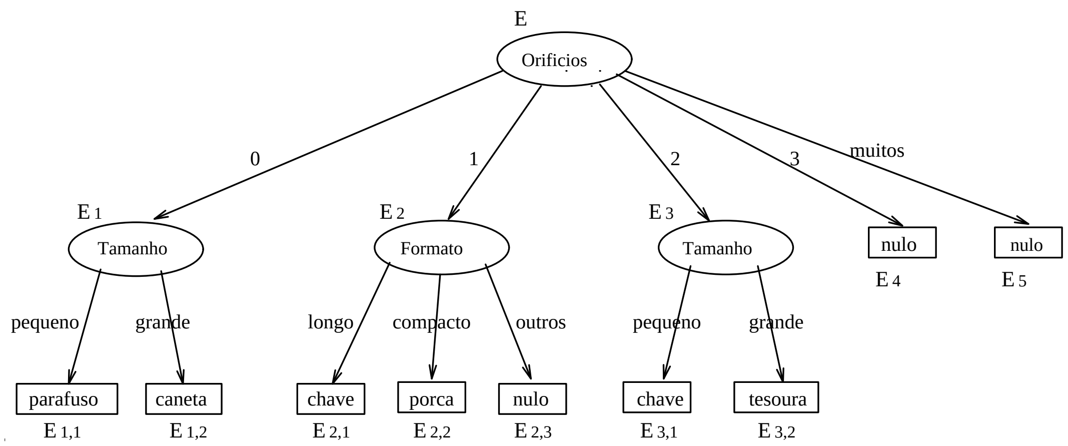


Figura 2.4: Uma árvore de decisão induzida a partir dos exemplos da figura 2.3

$$\begin{aligned} E_4 &= \{X \in E \text{ tq } X_{Orifícios} = 3\} \\ E_5 &= \{X \in E \text{ tq } X_{Orifícios} = muitos\} \end{aligned}$$

Analisemos agora o nó referente ao subconjunto E_1. Uma vez que E_1 contém exemplos de classes distintas, o nó deve ser dividido: rotulamos o nó com um novo atributo (*Tamanho*), criamos novos filhos para esse nó e dividimos E_1 conforme os valores de *Tamanho* $(pequeno, grande)$:

$$\begin{aligned} E_{1,1} &= \{X \in E_1 \text{ tq } X_{Tamanho} = pequeno\} \\ E_{1,2} &= \{X \in E_1 \text{ tq } X_{Tamanho} = grande\} \end{aligned}$$

Todos os exemplos em $E_{1,1}$ pertencem à mesma classe. Nesse caso, simplesmente rotulamos o nó correspondente com a classe à qual aqueles exemplos pertencem (parafuso) e declaramos esse nó como uma folha, passando a examinar outros nós que ainda não tenham sido devidamente tratados. O mesmo ocorre com o nó referente ao subconjunto $E_{1,2}$, cujos exemplos são todos de classe caneta.

O procedimento é análogo para os demais subconjuntos (E_2, E_3, E_{2_1},etc.); se o subconjunto é vazio (como ocorre em $E_4, E_5, E_{2,3}$), o nó correspondente é rotulado com *nulo*.

Suponha agora que a árvore de decisão tenha sido construída e que seja apresentado ao sistema o vetor de atributos de um objeto de classe desconhecida. Queremos que o sistema atribua uma classe ao objeto. O procedimento para classificação desse objeto é realizar o teste de atributo na raíz, ir para o ramo correspondente, realizar o teste de atributo no segundo nó, ir para o próximo ramo, sucessivamente até atingir uma folha. Então o objeto será classificado com o rótulo da respectiva folha.

Como uma ilustração, considere um objeto cujo vetor de atributos seja:

$$(\mathit{Tamanho} = grande, \mathit{Formato} = longo, \mathit{Orifícios} = 0).$$

Se tal objeto for submetido à árvore da figura 2.4, atingirá o nó referente ao conjunto $E_{1,2}$ e será, portanto, classificado como uma caneta.

Para a construção da árvore descrita acima, os nós internos foram rotulados ao acaso. Em geral, a escolha do atributo para a divisão de um nó é feita por meio de heurísticas (usualmente baseadas em medida de informação). Tais heurísticas buscam atributos de maior correlação com as classes. Na seção 2.4.5 e no capítulo 3 estudaremos algumas heurísticas de divisão dos nós com mais detalhes.

2.4.2 O algoritmo de construção de árvores de decisão

Na ilustração vista anteriormente, os objetos foram descritos em termos de atributos categóricos. Naquele caso, o número de ramos descendentes de cada nó interno corresponde ao número de valores possíveis do atributo que rotula este nó. Na maioria dos problemas reais de classificação os atributos são numéricos, e podem assumir um número muito grande de valores possíveis. Por essa razão, os atuais algoritmos da família TDIDT "categorizam" os atributos numéricos, através da divisão desses atributos em intervalos.

A figura 2.5 apresenta o algoritmo de construção das árvores. Inicialmente, a árvore é constituída por um único nó t sobre o qual armazenamos o conjunto de treinamento E. Se E for "puro" o suficiente (isto é, se houver uma classe que domina em E com grau suficiente de convicção), podemos considerar o nó t como um nó terminal (rotulado com a classe dominante em E) e interromper a expansão da árvore. Caso contrário, selecionamos o melhor atributo (em geral, o atributo de maior correlação com as classes presentes em E) e particionamos E de acordo com os valores desse atributo. Criamos um nó terminal correspondente a cada um dos sub-conjuntos obtidos, o qual será um filho de t. Repetimos a mesma operação recursivamente para cada nó terminal obtido.

Algumas componentes do algoritmo serão descritas brevemente a seguir:

- A regra de parada $P(E)$ é uma condição que o conjunto de treinamento E deve satisfazer para que seu nó correspondente seja considerado um nó terminal. As situações mais simples em que a expansão de um nó deve ser interrompida ocorrem quando o conjunto E é vazio ou quando todos os exemplos incidentes sobre t pertencem à mesma classe. Contudo, existem algoritmos que interrompem a expansão do nó sob condições mais complexas. Nesses casos, dizemos que ocorre uma *pré-poda* da árvore de decisão. Alguns dos algoritmos que realizam a pré-poda são o Cal5, o C4 [QCHL86] e o Assistant-86 [CKB87].

- A regra $classe(t)$ é uma função que atribui um nome de classe a um nó t. Em geral,

Algoritmo

Dados um conjunto de treinamento E
um nó terminal t
uma condição de parada $P(E)$
uma função de atribuição $classe(t)$
uma função de avaliação de atributos $score(E, a)$
uma função de categorização de atributos $categ(E, a)$

Se o conjunto E satisfaz a condição de parada $P(E)$,

então rotule t conforme a regra $classe(t)$

caso contrário

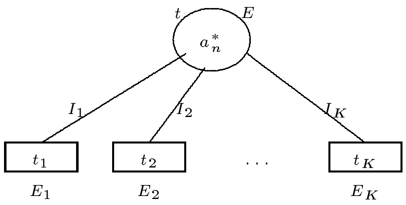
1. para cada atributo a_i, calcule a função $score(E, a_i)$;
   selecione o atributo $a_n^* = \arg\max score(E, a_i)$

2. rotule t com a_n^*

3. sejam $I_{n1}, I_{n2}, \ldots, I_{nK}$ os intervalos gerados segundo $categ(E, a_n^*)$;
   crie um ramo correspondente a cada intervalo, rotulando-o com uma condição do tipo

   $$x_n \in I_{nk}?, k = 1, \ldots, K$$

4. particione E em K sub-conjuntos $E_1, E_2, \ldots, E_K$ correspondentes aos ramos obtidos



5. aplique o algoritmo recursivamente para cada subconjunto E_i, a partir do nó t_i

6. após a expansão da árvore ter terminado, ajuste seu tamanho, podando alguns ramos inferiores se necessário.

Fim

Figura 2.5: Algoritmo recursivo para construção de uma árvore T a partir de um conjunto de treinamento E.

o critério adotado é atribuir o nome da classe que ocorre com maior freqüência sobre t. Existem ainda outros critérios de atribuição de classes, alguns dos quais serão discutidos na seção 2.4.3.

- A função $score(a_i)$ é utilizada para tentar identificar o atributo mais relevante existente sobre E, isto é, o atributo com maior correlação com as classes dos exemplos. Pode ser baseada em medida de informação, em medidas de custos ou em métodos estatísticos. No anexo discutimos uma série de critérios existentes.

- Para tratar atributos numéricos, os algoritmos "discretizam" os mesmos, criando sub-intervalos do tipo

$$
\begin{aligned}
I_{n1} &=]c_{n0}, c_{n1}] \\
I_{n2} &=]c_{n1}, c_{n2}] \\
&\vdots \\
I_{nK} &=]c_{nK-1}, c_{nK}[
\end{aligned}
$$

onde $c_{n0}, c_{n1}, \ldots, c_{nK}$ são constantes definidas segundo o critério de categorização $categ(E, a)$. A forma mais simples de discretização é a binarização dos atributos, através de condições do tipo $x_n \leq c$?, onde a constante c é obtida através de testes exaustivos sobre o conjunto de treinamento disponível.

Uma vez que os atributos categóricos também podem ser reduzidos a conjuntos $I_{n1} = \{a_{n1}\}, \ldots, I_{nK} = \{a_{nK}\}$, para simplificar a notação não faremos distinção entre atributos ordenados e categóricos, exceto em casos explícitos.

- Após a expansão da árvore, é comum que os subconjuntos de E correspondentes aos nós folhas sejam muito pequenos e altamente suscetíveis ao problema de ruídos. Além disso, quando uma árvore é excessivamente expandida, são incluídos atributos inúteis sob o ponto de vista da classificação dos objetos. É necessário então eliminar-se alguns dos ramos inferiores da árvore, de forma a atenuar o efeito dos ruídos e de forma a manter na árvore apenas os atributos que tenham poder preditivo ou, em outras palavras, que possuam uma correlação razoável com as classes. Esse processo é denominado *pós-poda* (ou simplesmente *poda*) das árvores.

Com base no algoritmo exposto acima, os elementos necessários para a construção de árvores de decisão são:

1. Uma regra $classe(t)$ para associar uma classe a cada nó terminal t.

2. Um critério de discretização dos atributos numéricos, $categ(E, a)$;

3. Um critério de avaliação dos testes para a divisão de cada nó da árvore, $score(E, a)$; alguns critérios serão discutidos no capítulo 3.

4. Um critério de poda de sub-ramos desnecessários na árvore; estudaremos alguns desses critérios no capítulo 4.

2.4.3 A regra de atribuição de classes

Nessa seção detalharemos alguns critérios de atribuição de classes a um nó, com as respectivas estimativas de erro. No restante desta seção, denotaremos $p(j|t)$ a probabilidade estimada de um exemplo incidente em t possuir classe j. Por exemplo, $p(j|t)$ pode ser definido como a proporção de exemplos de t que possuem classe j.

O critério mais simples é atribuir a classe de maior probabilidade estimada no nó, isto é, a classe que maximiza $p(j|t)$. Assim, a probabilidade estimada de um exemplo incidente sobre t ser classificado erroneamente é:

$$\begin{aligned} r(t) &= 1 - \max_j p(j|t) \\ &= \min_j \sum_{i \neq j} p(i|t) \end{aligned} \tag{2.1}$$

[BFOS84] utiliza o conceito de custo de erro de classificação e utiliza um critério de atribuição que minimiza esse custo no nó, como veremos a seguir.

Definição 2.4.2 *O* custo de erro *em classificar um objeto pertencente à classe j como sendo de classe i é denotado por $C(i, j)$ e satisfaz*

- $C(i, j) \geq 0, i \neq j;$
- $C(i, j) = 0, i = j.$

Suponha que um objeto de classe desconhecida incida sobre um nó t, cuja distribuição de probabilidades estimadas das classes seja $p(j|t)$, $j = 1, 2, \ldots, J$. Se esse objeto for classificado como classe i, então o custo esperado do erro na classificação será $\sum_j C(i, j)p(j|t)$. Assim, um critério natural de atribuição é escolher a classe i que minimiza a expressão acima.

Sob essa regra, o custo esperado no erro de classificação de um objeto incidente sobre t será:

$$r(t) = \min_i \sum_j C(i, j)p(j|t). \tag{2.2}$$

Note que a primeira regra (escolher a classe de maior probabilidade estimada no nó) é um caso particular do critério de minimização de custos, onde $C(i, j) = 1$ para $i \neq j$. Basta verificar que, com os custos $C(i, j)$ unitários, as equações 2.1 e 2.2 se tornam equivalentes.

Durante a classificação de um objeto desconhecido, em muitas situações pode ser desejável que a árvore T não forneça apenas o nome de uma classe j para esse objeto, mas forneça as probabilidades estimadas desse objeto pertencer a cada uma das classes $1, 2, \ldots, J$. Como um exemplo, considere o problema do diagnóstico médico. Suponha que um determinado paciente possa estar sofrendo de uma das três doenças D_1, D_2 ou D_3, e que o vetor de atributos referente a esse paciente incida sobre uma folha t da árvore. Ao invés de atribuir apenas uma doença ao paciente e ignorar as demais, o sistema poderá ser mais confiável se for capaz de estimar as probabilidades do paciente estar sofrendo de cada uma das doenças D_1, D_2, D_3.

Para essas situações, a função de atribuição $classe(t)$ pode ser redefinida como

$$classe(t) = (p(1|t), p(2|t), \ldots, p(J|t)).$$

Nesse caso, a árvore T passa a ser chamada de *árvore probabilística*, e não mais árvore de decisão [MST94].

Podemos estimar o custo de erro no nó t usando o seguinte raciocínio: sendo $p(j|t)$ a probabilidade estimada de um objeto incidente sobre t pertencer à classe j, o custo estimado de erro em atribuir a classe i a esse objeto será

$$\sum_j C(i,j)p(j|t).$$

Como a função $classe(t)$ atribui a classe i ao objeto com probabilidade $p(i|t)$, o custo estimado de erro no nó t será

$$r(t) = \sum_i p(i|t) \left(\sum_{i \neq j} C(i,j)p(j|t) \right) = \sum_{i,j} C(i,j)p(i|t)p(j|t).$$

Uma vez apresentadas as regras de atribuição de classe e as respectivas estimativas de erro $r(t)$, será útil definirmos os custos estimados de erros de cada nó e o custo de erro da árvore T. Os conceitos que se seguem são válidos para as três regras indistintamente.

Seja $R(t)$ o custo total de erro no nó t, dado por

$$R(t) = r(t)p(t),$$

onde $p(t)$ é a proporção de exemplos em E que incidem sobre o nó t.

Então

Definição 2.4.3 *O custo total de erros de classificação da árvore $R(T)$ é dado por*

$$R(T) = \sum_{t \in \overline{T}} R(t),$$

onde $\overline{T}$ é o conjunto de folhas de T.

2.4.4 A discretização dos atributos numéricos

Durante a construção da árvore, o algoritmo TDIDT verifica cada um dos nós que não tenha filhos. Considere um nó t que não tenha filhos e que não satisfaça à condição de parada. Esse nó precisará ser expandido, isto é, o conjunto E_t de exemplos incidentes sobre t será particionado, e a cada subconjunto obtido corresponderá um filho de t. Para esta expansão, todos os atributos deverão ser levados em conta, e aquele que otimizar a função *score* será escolhido.

Atributos que possuem muitos valores possíveis, como é o caso de atributos numéricos, tendem a ser privilegiados nesta escolha, podendo gerar árvores muito esparsas e com baixo poder preditivo.

Por essa razão, cada atributo numérico a_n necessita ser "discretizado", através do particionamento em intervalos

$$\begin{aligned} I_{n1} &=]c_{n0}, c_{n1}] \\ I_{n2} &=]c_{n1}, c_{n2}] \\ &\vdots \\ I_{nK} &=]c_{nK-1}, c_{nK}[\end{aligned}$$

onde $c_{n0}, c_{n1}, \ldots, c_{nK}$ são constantes definidas segundo um certo critério.

Cada tupla $s = (I_{n1}, \ldots, I_{nK})$ de intervalos de a_n fornece uma divisão possível para o nó t. Sendo E_t o conjunto de exemplos incidentes sobre t, a divisão s gera os sub-conjuntos

$$E_{tk} = \{X \in E_t | x_k \in I_{nk}\}, k = 1, 2, \ldots, K,$$

onde x_n é o valor do atributo a_n para cada exemplo X.

A maioria dos algoritmos realiza uma binarização dos atributos, formando uma condição do tipo

$$x_{nk} \leq c_{nk}?$$

Nestes casos, para a determinação da melhor binarização, é necessário ordenar o conjunto de exemplos em ordem crescente de valores de a_n, e realizar um teste de *score* (que veremos na próxima seção) para cada condição do tipo

$$x_{nk} \leq c_{nk}^i$$

onde c_n^i é o valor de a_n para o i-ésimo exemplo dentro da ordenação obtida. A binarização ótima s de a_n será aquela que otimizar a função *score*.

Alguns algoritmos, como o Cal5 e o REAL, realizam discretizações mais complexas, onde a definição dos intervalos obedece a critérios ligeiramente diferentes.

Nos capítulos que seguem, utilizaremos a notação de partições binárias, já que sua generalização é imediata.

2.4.5 A escolha do atributo ótimo

Após termos definido as possíveis divisões para os nós da árvore, precisamos definir algum critério de avaliação que escolha a melhor divisão para cada nó. A grosso modo, a melhor divisão para um nó deve ser aquela que agrupe, da melhor forma possível, os exemplos de mesma classe.

Para o problema em que os custos de erro $C(i, j)$ são unitários, isto é,

$$C(i,i) = 0\ ,\ \ C(i,j) = 1,\ i \neq j$$

muitos dos critérios de divisões dos nós são baseados no conceito de impureza, definido como segue:

Definição 2.4.4 *Uma* função de impureza *é uma função ϕ definida sobre o conjunto de todas as J-tuplas de números $(p_1, p_2, \ldots, p_J)$ satisfazendo $p_j \geq 0,\ j = 1, 2, \ldots, J\ \sum_j p_j = 1$ com as propriedades*

- *ϕ atinge seu máximo somente no ponto $(\frac{1}{J}, \frac{1}{J}, \ldots, \frac{1}{J})$;*
- *ϕ atinge seu mínimo somente nos pontos $(1, 0, \ldots, 0), (0, 1, 0, \ldots), \ldots, (0, \ldots, 0, 1)$;*
- *ϕ é uma função simétrica de $p_1, p_2, \ldots, p_J$.*

Note que as restrições acima retratam bem as noções intuitivas de impureza: de fato, a impureza de um nó deve ser maior quando todas as classes estão igualmente presentes no mesmo, e menor quando o nó contém apenas uma classe.

No capítulo 3, abordaremos algumas funções de impureza conhecidas.

Definição 2.4.5 *Dada uma função de impureza ϕ, define-se a medida de impureza de um nó t como*

$$imp(t) = \phi(p(1|t), p(2|t), \ldots, p(J|t))$$

onde p(j—t) é a probabilidade de um objeto incidente sobre o nó t pertencer à classe j.

Para todo nó t, suponha que haja uma divisão candidata s que divide t em t_v e t_f, de forma que uma proporção p_v dos exemplos de t vão para t_v e uma proporção p_f dos exemplos de t vão para t_f. O *critério de qualidade da divisão* é definido pelo decréscimo na impureza

$$\Delta imp(s,t) = imp(t) - p_v imp(t_v) - p_f imp(t_f)\ .$$

$\Delta imp(s,t)$ pode ser visto como o ganho de pureza obtido pela divisão s. Então, a melhor divisão para a divisão de t será aquela que maximiza esse ganho, isto é,

$$s^* = \arg\max_s \Delta imp(s,t).$$

Suponha agora que tenhamos construído uma árvore binária T, realizando algumas divisões sucessivas a partir da raíz (onde cada divisão corresponde a um teste). Denote o conjunto de folhas por $\overline{T}$; ponha $I(t) = p(t)imp(t)$ e defina a impureza da árvore $I(T)$ como

$$I(T) = \sum_{t\in T} I(t) = \sum_{t\in\overline{T}} p(t)imp(t).$$

O seguinte teorema, demonstrado por [BFOS84], mostra que a aplicação sucessiva do critério acima gera uma árvore com impureza global mínima.

Teorema 2.4.6 *A impureza global $I(T)$ da árvore será mínima se, a cada nó não terminal t, tivermos escolhido a divisão $s^* = arg \max_s \Delta i(s,t)$.*

Dem. Tome um nó qualquer $t \in \overline{T}$ e, usando uma divisão s, particione o nó em t_v e T_f. A nova árvore T' terá impureza

$$I(T') = \sum_{\overline{T}-\{t\}} I(t) + I(t_v) + I(T_f).$$

O decréscimo global de impureza será

$$I(T) - I(T') = I(t) - I(t_v) - I(t_f).$$

Por depender somente do nó t e da divisão s, o decréscimo global de impureza pode ser escrito como

$$\begin{aligned} \Delta I(s,t) &= I(t) - I(t_v) - I(t_f) \\ &= p(t)imp(t) - p(t_v)imp(t_v) - p(t_f)imp(t_f) \end{aligned} \tag{2.3}$$

Definindo as proporções p_v e p_f dos exemplos de t que vão para t_v e t_f, respectivamente, como

$$p_v = p(t_v)/p(t)\ , \quad p_f = p(t_f)/p(t)$$

podemos reescrever a expressão 2.3 como

$$\begin{aligned} \Delta I(s,t) &= p(t)[imp(t) - p_v imp(t_v) - p_f imp(t_f)] \\ &= p(t)\Delta imp(s,t). \end{aligned}$$

Uma vez que $\Delta I(s,t)$ difere de $\Delta i(s,t)$ apenas pelo fator $p(t)$, a mesma divisão s^* maximiza as duas expressões. Portanto, o critério de seleção da melhor divisão de cada nó pode ser visto como uma tentativa sucessiva de minimizar a impureza global da árvore.

□

O conceito de impureza não se aplica adequadamente em problemas onde os custos de erro não sejam unitários. Nesses problemas, o critério de divisão de nós deve dar preferência para que as classes com maiores custos de erro fiquem agrupadas nos mesmos nós descendentes, em detrimento àquelas classes de custos de erro inferiores.

2.5 Considerações Finais

Vimos neste capítulo alguns conceitos básicos sobre o aprendizado proposicional, bem como o algoritmo geral da família TDIDT (Top-Down Induction of Decision Trees).

O que diferencia os algoritmos TDIDT entre si são suas componentes internas: a condição de parada $P(E)$, a função de atribuição $classe(t)$, a função de avaliação dos atributos $score(E, a)$, a função de categorização de atributos $categ(E, a)$ e o critério de poda da árvore. Nem todos os algoritmos TDIDT implementam pós-poda; alguns realizam uma "pré-poda" durante a fase de expansão da árvore, como é o caso do Cal5 e do Real, que serão discutidos em capítulos posteriores.

Algumas componentes foram discutidas neste capítulo. Nos capítulos 3 e 4 serão apresentados, respectivamente, algumas funções de avaliação de atributos e alguns critérios de pós-poda.

Capítulo 3

Critérios Para Avaliação dos Atributos

3.1 Considerações Iniciais

No capítulo anterior, apresentamos o algoritmo geral da família TDIDT. Durante a construção da árvore, após decidir pela expansão de um nó t, o algoritmo gera um conjunto de divisões possíveis[1]. Para cada divisão possível s, é medido o decréscimo na impureza $\Delta imp(s,t)$ obtido pela divisão do nó t através de s.

O critério de avaliação adotado é baseado, geralmente, em uma *função de impureza*, definida sobre todas as J-tuplas de $(p_1, p_2, \ldots, p_J)$ satisfazendo

$$\begin{aligned} p_j &\geq 0,\ j = 1, 2, \ldots, J \\ \sum_j p_j &= 1. \end{aligned}$$

A função de impureza ϕ deve satisfazer às restrições

1. ϕ atinge seu máximo somente no ponto $(\frac{1}{J}, \frac{1}{J}, \ldots, \frac{1}{J})$;
2. ϕ atinge seu mínimo somente nos pontos $(1, 0, \ldots, 0), (0, 1, 0, \ldots), \ldots, (0, \ldots, 0, 1)$;
3. ϕ é uma função simétrica de $p_1, \ldots, p_J$.

[BFOS84] e [Min87] observaram que os critérios para a divisão dos nós não diferem significativamente em termos de precisão da árvore construída, mas sim em termos do tamanho da árvore.

[1] Ver seção 2.4.4

Neste capítulo, estudaremos algumas medidas utilizadas para seleção da melhor divisão de um nó. Algumas dessas medidas se adequam às definições de impureza expostas acima.

3.2 O Índice Gini

O índice Gini, implementado no sistema CART [BFOS84], foi criado fundamentalmente como uma medida de variância para dados categóricos, isto é, dados que não podem ser quantificados [LM71].

Inicialmente, apresentaremos este índice para a situação em que o os custos de erro $C(i,j)$[2] são unitários; em seguida, apresentaremos duas versões possíveis do índice para custos de erros variáveis.

3.2.1 Índice Gini para custos unitários

O índice de diversidade Gini possui a forma

$$\mathcal{G}(p_1, p_2, \ldots, p_J) = \sum_{i \neq j} p_i p_j \tag{3.1}$$

e também pode ser escrita como

$$\begin{aligned} \mathcal{G}(p_1, p_2, \ldots, p_J) &= \sum_j p_j \sum_{i \neq j} p_i \\ &= \sum_j p_j (1 - p_j) \\ &= 1 - \sum_j p_j^2 \end{aligned}$$

Uma interpretação do índice Gini pode ser dada em termos de variâncias. Em um nó t, associe a todos os exemplos de classe j o valor 1, e aos demais o valor 0. Então a variância desses valores será $p(j|t)(1 - p(j|t))$. Se esse procedimento for repetido para todas as J classes e as variâncias somadas, o resultado será

$$\sum_j p(j|t)(1 - p(j|t)).$$

Outra interpretação interessante pode ser dada em termos erro de classificação. Ao invés de definir $classe(t)$ com o rótulo da classe majoritária, defina classe(t) como uma função que atribui, para um objeto selecionado aleatoriamente dentro do nó, a classe i

[2]Definidos na seção 2.4.3

com probabilidade $p(i|t)$. A probabilidade deste objeto ser da classe j é $p(j|t)$. Portanto, a probabilidade estimada de erro sob esta regra é o índice de Gini

$$\sum_{i \neq j} p_i p_j.$$

Podemos verificar que o índice Gini satisfaz às restrições de uma função de impureza. As restrições (2) e (3) são trivialmente satisfeitas; para mostrarmos que a restrição (1) também é satisfeita, usaremos uma desigualdade válida para toda função côncava $f(a)$:

$$f\left(\frac{1}{J}\sum_{j=1}^{J} a_j\right) \geq \frac{1}{J}\sum_{j=1}^{J} f(a_j),$$

onde $a_1, \ldots, a_J$ são quaisquer números positivos. Fixando $a_j = p_j$ e $f(a) = a(1-a)$ e tendo em mente que $\sum_j p_j = 1$, temos:

$$\begin{aligned} f\left(\frac{1}{J}\sum_{j=1}^{J} p_j\right) &= \frac{1}{J}\left(1 - \frac{1}{J}\right) \\ &\geq \frac{1}{J}\sum_{j=1}^{J} p_j(1-p_j) \\ &= \frac{1}{J}\mathcal{G}(p_1, p_2, \ldots, p_J). \end{aligned}$$

Logo,

$$\mathcal{G}(p_1, p_2, \ldots, p_J) \leq 1 - \frac{1}{J} = \mathcal{G}\left(\frac{1}{J}, \frac{1}{J}, \ldots, \frac{1}{J}\right).$$

A concavidade estrita de $\mathcal{G}(p_1, p_2, \ldots, p_J)$ garante que o ponto $(\frac{1}{J}, \frac{1}{J}, \ldots, \frac{1}{J})$ é o único ponto de máximo.

□

3.2.2 Índice Gini para custos variáveis

Na presença de custos variáveis de erro, adotar um critério de divisão de nós baseado exclusivamente na função de impureza deixa de ser a melhor estratégia, pois em tais situações deve-se dar prioridade para que classes de custo de erro mais elevado fiquem no mesmo nó descendente, em detrimento daquelas classes de custo de erro inferior. O índice Gini possui duas extensões para a estratégia de divisão dos nós, aplicáveis na presença de custos variáveis.

A estratégia mais direta é a que segue: dados os custos de erro de classificação $C(i,j)$ tais que:

$$C(i,i) = 0\ ;\ \ C(i,j) \geq 0,\ i \neq j$$

e as probabilidades estimadas $p(j|t)$, definimos o índice Gini por

$$\mathcal{G}(p(1|t), p(2|t), \ldots, p(J|t)) = \sum_{i,j} C(i,j)p(i|t)p(j|t). \tag{3.2}$$

No caso particular de um problema que contenha apenas duas classes, a equação acima se reduz a

$$\mathcal{G}(p(1|t), p(2|t)) = (C(1|t) + C(2|t))p(1|t)p(2|t),$$

dando essencialmente o mesmo critério de divisão do caso com custos unitários.

Surge aqui uma dificuldade: $p(i|t)p(j|t)$ na equação é $C(i,j) + C(j,i)$. O critério não consegue identificar qualquer distinção entre $C(i,j)$ e $C(j,i)$, tratando-os como se cada um fosse igual a $(C(i,j) + C(j,i))/2$. Por essa razão, essa extensão do índice Gini só é apropriada para casos em que $C(i,j)$ e $C(j,i)$ não sejam significativamente diferentes.

Apresentamos agora a segunda extensão para o índice Gini, que trata dos casos em que $C(i,j)$ e $C(j,i)$ diferem consideravelmente. Essa extensão tenta alterar a distribuição *a priori* de probabilidades das classes, $\{\pi(j)\}$, de forma a tratar o problema como se os custos fossem unitários.

Considere um problema de duas classes equiprováveis, e suponha que $C(1,2) = 2$, $C(2,1) = 1$, isto é, que o custo de erro da classe 2 seja o dobro do custo de erro da classe 1. Queremos então uma árvore que classifique erroneamente um número pequeno de instâncias da classe 2. Pode-se pensar ainda que cada instância da classe 2 classificada erroneamente conta em dobro, assim a situação é similar àquela em que os custos sejam unitários e a probabilidade de ocorrência da classe 2 seja o dobro da probabilidade de ocorrência da classe 1.

Com base nessa idéia, seja $Q(i|j)$ a proporção de exemplos de classe j em E classificados como se fossem de classe i pela árvore T. Então o custo estimado de erro da árvore é definido como[3]

$$R(T) = \sum_{i,j} C(i,j)Q(i|j)\pi(j).$$

Sejam $\{\pi'(j)\}$ e $\{C'(i,j)\}$ formas alteradas de $\{\pi(j)\}$ e $\{C(i,j)\}$ tais que

$$C'(i,j)\pi'(j) = C(i,j)\pi(j). \tag{3.3}$$

Então $R(T)$ permanece o mesmo quando computado usando-se $\{\pi'(j)\}$ e $\{C'(i,j)\}$.

Tome $\{C'(i,j)\}$ como sendo a matriz de custos unitários e suponha que seja possível encontrar a distribuição alterada $\{\pi'(j)\}$ que satisfaça 3.3. Então, a estrutura da árvore será a mesma (e o custo estimado de erro também o será) para um problema onde os custos são unitários e as probabilidades das classes são dadas por $\{\pi'(j)\}$.

[3]Ver seção 4.2.2

Se as colunas da matriz de custos forem constantes, isto é,

$$C(i,j) = C(j),$$

então os custos $C'(i,j)$ pode ser tomados como unitário através das probabilidades *a priori*

$$\pi'(j) = \frac{C(j)\pi(j)}{\sum_j C(j)\pi(j)}. \tag{3.4}$$

Então, o método de alteração de probabilidades *a priori* consiste nos seguintes passos:

1. Calcule um custo de erro para cada classe, $C(j)$ (no sistema CART, $C(j)$ é calculado como $C(j) = \sum_i C(i,j)$);
2. Obtenha $\{\pi'(j)\}$ através da equação 3.4;
3. Construa a árvore usando custos unitários e usando a distribuição $\{\pi'(j)\}$; utilize o índice Gini definido na seção 3.2.1.

3.3 O Critério Twoing

O critério twoing, também implementado no sistema CART, pode ser formalizado como segue. Denote o conjunto de classes por C, isto é, $C = \{1, 2, \ldots, J\}$. A cada nó, separe as classes em duas superclasses,

$$C_1 = \{j_1, \ldots, j_k\}\ , \quad C_2 = C - C_1.$$

Considere agora o problema original como um problema de duas classes, isto é, associe a superclasse C_1 a todos os exemplos cujas classes pertençam a C_1 e associe a superclasse C_2 aos exemplos cujas classes pertençam a C_2.

Para toda divisão s de um nó t, compute $\Delta imp(s,t)$ como se houvesse apenas duas classes. Para o cálculo de $\Delta imp(s,t)$, adote a medida de impureza

$$imp(t) = p(C_1|t)p(C_2|t),$$

onde

$$p(C_1|t) = \sum_{j \in C_1} p(j|t) \ \text{ e } \ p(C_2|t) = \sum_{j \in C_1} p(j|t).$$

Uma vez que $imp(t)$ e $\Delta imp(s,t)$ dependem da escolha de C_1, são adotadas as notações

$$imp(t, C_1) \ \text{ e } \ \Delta imp(s, t, C_1).$$

Para cada superclasse $C_1 \subseteq 2^C$, escolha o teste

$$s^*(C_1) = \arg\max_s \Delta imp(s, t, C_1);$$

Finalmente, divida o nó t usando o teste $s^* = s^*(C_1^*)$, onde

$$C_1^* = \arg \max_{C_1 \subseteq 2^C} \Delta imp(s^*(C_1), t, C_1).$$

Uma vantagem que este método traz é fornecer divisões estratégicas no sentido de que, nos nós próximos ao topo da árvore, o método tenta agrupar grandes quantidades de classes que sejam similares em algumas características; nos nós mais próximos às folhas, o método tenta isolar as classes entre si.

Por exemplo, suponha que, em um problema contendo quatro classes, originalmente as classes 1 e 2 tenham sido colocadas no mesmo grupo e separadas das classes 3 e 4, resultando em um nó cuja configuração seja

Classe:	1	2	3	4
Numero de exemplos:	50	50	3	1

Então, na próxima divisão desse nó, o maior ganho de informação seria obtido separando a classe 1 da classe 2.

O método twoing parece ser computacionalmente ineficiente quando existe um número grande de classes. De fato, para um problema com J classes, existem 2^{J-1} partições possíveis de C em duas superclasses. Entretanto, o seguinte teorema, demonstrado por [BFOS84] e refinado em nosso trabalho, garante que esse critério pode ser reformulado de forma que seu tempo de execução se torne muito próximo ao tempo de cálculo do índice Gini.

Teorema 3.3.1 *Seja s uma divisão possível para um nó t, e sejam p_v e p_f as proporções de exemplos de t que vão, respectivamente, para os nós descendentes t_v e t_f. Então, sob o critério $p(C_1|t)p(C_2|t)$, uma superclasse $C_1(s)$ que maximiza $\Delta imp(s, t, C_1)$ é*

$$C_1(s) = \{j : p(j|t_v) \geq p(j|t_f)\}$$

e

$$\max_{C_1} \Delta imp(s, t, C_1) = \frac{p_v p_f}{4} \left[\sum_j |p(j|t_v) - p(j|t_f)| \right]^2.$$

Dem. Para maior simplicidade de notação, considere $p(C_i|t) = q_i$, $p(C_i|t_v) = q_{i|v}$ e $p(C_i|t_f) = q_{i|f}$, $i = 1, 2$.

Em primeiro lugar, mostraremos a relação

$$\Delta imp(s,t,C_1) = p_v p_f [q_{1|v} - q_{1|f}]^2. \tag{3.5}$$

A igualdade acima é obtida como segue:

$$\begin{aligned}
\Delta imp(s,t,C_1) &= imp(t) - p_v imp(t_v) - p_f imp(t_f) \\
&= q_1 q_2 - p_v q_{1|v} q_{2|v} - p_f q_{1|f} q_{2|f} \\
&= q_1(1-q_1) - p_v q_{1|v}(1-q_{1|v}) - p_f q_{1|f}(1-q_{1|f}) \\
&= q_1 - p_v q_{1|v} - p_f q_{1|f} + p_v q_{1|v}^2 + p_f q_{1|f}^2 - q_1^2
\end{aligned}$$

Note que, para toda classe $j = 1, 2, \ldots, J$,

$$\begin{aligned}
p_v p(j|t_v) + p_f p(j|t_f) &= \frac{p(t_v)}{p(t)} \frac{p(j,t_v)}{p(t_v)} + \frac{p(t_f)}{p(t)} \frac{p(j,t_f)}{p(t_f)} \\
&= \frac{p(j,t_v) + p(j,t_f)}{p(t)} \\
&= p(j|t)
\end{aligned}$$

e portanto,

$$q_1 = p_v q_{1|v} + p_f q_{1|f}.$$

Utilizando a igualdade acima, temos

$$\begin{aligned}
\Delta imp(s,t,C_1) &= p_v q_{1|v}^2 + p_f q_{1|f}^2 - (p_v q_{1|v} + p_f q_{1|f})^2 \\
&= (1-p_v) p_v q_{1|v}^2 + (1-p_f) p_f q_{1|f}^2 - 2 p_v p_f q_{1|v} q_{1|f} \\
&= p_f p_v q_{1|v}^2 + p_v p_f q_{1|f}^2 - 2 p_v p_f q_{1|v} q_{1|f} \\
&= p_v p_f [q_{1|v} - q_{1|f}]^2.
\end{aligned}$$

Tendo demonstrado a relação 3.5, passaremos à demonstração do teorema. Nosso problema é encontrar, dada uma divisão s, uma superclasse $C_1(s)$ que maximize $\Delta imp(s,t,C_1)$.

Para todo real z, definimos z^+ e z^- como as partes positiva e negativa de z: se $z \geq 0$ tome $z^+ = z$ e $z^- = 0$; se $z \leq 0$, tome $z^+ = 0$ e $z^- = -z$. Por esta definição, $z = z^+ - z^-$, e $|z| = z^+ + z^-$.

Para $j = 1, 2, \ldots, J$, defina $z_j = p(j|t_v) - p(j|t_f)$. Defina também

$$Z = q_{1|v} - q_{1|f} = \sum_{j \in C_1} z_j.$$

Na relação 3.5, uma vez que p_v e p_f dependem apenas de s e não de C_1, uma classe $C(s)$ que maximiza $\Delta imp(s,t,C_1)$ será aquela que maximize ou minimize Z. O valor máximo de Z é igual a $\sum_{j \in C_1} z_j^+$ e é obtido por $C_1 = \{j : z_j \geq 0\}$. O valor mínimo de Z é $-\sum_{j \in C_1} z_j^-$ e é obtido por $C_1 = \{j : z_j < 0\}$.

Uma vez que

$$\sum_j z_j^+ - \sum_j z_j^- = \sum_j z_j = 0,$$

temos que o valor máximo de Z é igual ao valor absoluto do mínimo de Z ($\max Z = |\min Z|$); assim,

$$\max Z = |\min Z| = \frac{1}{2}\left(\sum_j z_j^+ + \sum_j z_j^-\right) = \frac{1}{2}\left(\sum_j |z_j|\right).$$

Então, da relação 3.5 resulta que

$$\begin{aligned} \max_{C_1} \Delta imp(s,t,C_1) &= p_v p_f [\max Z] \\ &= \frac{p_v p_f}{4}\left[\sum_j |p(j|t_v) - p(j|t_f)|\right]^2, \end{aligned}$$

e uma classe maximizadora é

$$C_1(s) = \{j : p(j|t_v) \geq p(j|t_f)\}.$$

□

Usando o teorema 3.3.1, o procedimento para buscar a melhor partição de nós é reformulado da seguinte maneira. Para cada divisão s de um nó t em T_v e t_f, defina a função

$$\mathcal{T}(s,t) = \frac{p_v p_f}{4}\left[\sum_j |p(j|t_v) - p(j|t_f)|\right]^2.$$

Então, a divisão selecionada $s^*(C_1^*)$ é

$$s^* = \arg\max_s \mathcal{T}(s,t)$$

e C_1^* é dado por

$$C_1^* = \{j : p(j|t_v^*) \geq p(j|t_f^*)\},$$

onde t_v^* e t_f^* são os nós dados pela divisão s^*.

[BFOS84] não obteve uma generalização do teorema 3.3.1 para medidas de impureza distintas de $p(C_1|t)p(C_2|t)$.

A idéia do critério twoing também pode ser aplicada em situações nas quais as classes em C, embora categóricas, possuam uma ordem natural. Por exemplo, em um estudo da dor remanescente após um tratamento, as classes poderiam ser definidas como pior, a mesma, leve melhora, sensível melhora, notável melhora, recuperação plena.

Em tais aplicações, é natural considerar o *critério twoing ordenado* dado por

$$\mathcal{T}(s,t) = \max_{C_1} \Delta imp(s,t,C_1),$$

onde C_1 e C_2 formam uma partição de $C = 1, 2, \ldots, J$ em duas superclasses restritas à condição de serem da forma

$$C_1 = \{1, \ldots, j_1\}\ , \quad C_2 = \{j_1 + 1, \ldots, J\}\ , \quad j_1 = 1, 2, \ldots, J-1.$$

Finalmente, deve-se notar que o critério twoing não se enquadra no conceito geral de impureza apresentado, embora isso não seja necessariamente uma desvantagem; um critério de partição deve ser julgado principalmente em termos de seu desempenho na construção de árvores.

3.4 A Medida de Entropia

O conceito de entropia surgiu inicialmente a partir da mecânica estatística, e sua aplicação tem se extendido para diversos outros fenômenos (físicos ou não). A medida de entropia, como veremos, também é muito conveniente como medida de informação. Em nosso trabalho, empregaremos a abordagem adotada por [Khi53], baseada na teoria de probabilidade.

Em teoria de probabilidade um *sistema completo de eventos* $A_1, A_2, \ldots, A_J$ é um conjunto de eventos tal que um e somente um deles deve ocorrer a cada tentativa (por exemplo, o aparecimento de $1, 2, 3, 4, 5$ ou 6 pontos no lancamento de um dado). Se tivermos os eventos $A_1, A_2, \ldots, A_J$ de um sistema completo, juntamente com suas probabilidades estimadas $p_1, p_2, \ldots, p_J$ ($p_i \geq 0,\ \sum_{i=1}^{J} p_i = 1$), então temos um *esquema finito*

$$A = \begin{pmatrix} A_1 & A_2 & \ldots & A_J \\ p_1 & p_2 & \ldots & p_J \end{pmatrix}$$

Todo esquema finito descreve um estado de incerteza, onde se deseja prever o resultado de um experimento com base nas probabilidades de cada evento. Claramente, o grau de incerteza é diferente para esquemas diferentes.

A medida de entropia busca, então, medir o grau de incerteza presente em cada esquema finito, e é dada pela função

$$\mathcal{E}(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log p_i.$$

onde os logaritmos são tomados numa base fixa qualquer e atribuímos $p_i \log p_i = 0$ sempre que $p_i = 0$.

[Khi53] demonstrou que $\mathcal{E}(p_1, p_2, \ldots, p_J)$ possui uma série de propriedades que se pode esperar de uma medida de impureza.

Em primeiro lugar, é imediato ver que $\mathcal{E}$ é simétrica e que $\mathcal{E}(p_1, p_2, \ldots, p_J) = 0$ se, e somente se, algum dos números p_i for igual a 1 e todos os demais iguais a 0.

A concavidade estrita de $\mathcal{E}$ garante que $\mathcal{E}$ possui um único ponto de máximo; mostraremos agora que

$$\mathcal{E}(p_1, p_2, \ldots, p_J) \leq \mathcal{E}\left(\frac{1}{J}, \frac{1}{J}, \ldots, \frac{1}{J}\right).$$

Para isso, usaremos a propriedade válida para toda função convexa $f(a)$:

$$f\left(\frac{1}{J}\sum_{j=1}^{J} a_j\right) \leq \frac{1}{J}\sum_{j=1}^{J} f(a_j),$$

onde $a_1, \ldots, a_J$ são quaisquer números positivos. Fixando $a_j = p_j$ e $f(a) = a \log a$ e tendo em mente que $\sum_j p_j = 1$, temos:

$$\begin{aligned} f\left(\frac{1}{J}\sum_{j=1}^{J} p_j\right) &= \frac{1}{J}\log\frac{1}{J} \\ &\leq \frac{1}{J}\sum_{j=1}^{J} p_j \log p_j \\ &= -\frac{1}{J}\mathcal{E}(p_1, p_2, \ldots, p_J). \end{aligned}$$

Logo,

$$\mathcal{E}(p_1, p_2, \ldots, p_J) \leq \log J = \mathcal{E}\left(\frac{1}{J}, \frac{1}{J}, \ldots, \frac{1}{J}\right).$$

□

Suponha agora que tenhamos dois esquemas finitos

$$A = \begin{pmatrix} A_1 & A_2 & \ldots & A_J \\ p_1 & p_2 & \ldots & p_J \end{pmatrix}, \quad B = \begin{pmatrix} B_1 & B_2 & \ldots & B_K \\ q_1 & q_2 & \ldots & q_K \end{pmatrix}$$

e que esses dois esquemas sejam mutuamente independentes, isto é, que a probabilidade r_{jk} da ocorrência simultânea dos eventos A_j e B_k seja $p_j q_k$. Então, o conjunto de eventos simultâneos $A_j B_k$ ($1 \leq j \leq J$, $1 \leq k \leq K$), com probabilidades r_{jk} representa outro esquema finito, chamado *produto* dos esquemas A e B e denotado por AB. Sejam $\mathcal{E}(A)$, $\mathcal{E}(B)$ e $\mathcal{E}(AB)$ as entropias correspondentes aos esquemas A, B, AB. Uma vez que a realização do esquema AB é equivalente à realização dos esquemas A e B individualmente, é natural considerar que a incerteza presente no esquema AB seja igual à soma das incertezas nos esquemas A e B, ou seja,

$$\mathcal{E}(AB) = \mathcal{E}(A) + \mathcal{E}(B). \tag{3.6}$$

A medida de entropia possui esta propriedade, pois

$$\begin{aligned}
-\mathcal{E}(AB) &= \sum_j \sum_k r_{jk} \log r_{jk} \\
&= \sum_j \sum_k p_j q_k (\log p_j + \log q_k) \\
&= \sum_j p_j \log p_j \sum_k q_k + \sum_k q_k \log q_k \sum_j p_j \\
&= -\mathcal{E}(A) - \mathcal{E}(B).
\end{aligned}$$

Consideremos agora o caso onde A e B são mutuamente dependentes. Denote por $q_{k|j}$ a probabilidade do evento B_k ocorrer, dado que o evento A_j tenha ocorrido, de forma que

$$r_{jk} = p_j q_{k|j}, \quad (1 \leq j \leq J,\ 1 \leq k \leq K).$$

Então

$$\begin{aligned}
-\mathcal{E}(AB) &= \sum_j \sum_k p_j q_{k|j} (\log p_j + \log q_{k|j}) \\
&= \sum_j p_j \log p_j \sum_k q_{k|j} + \sum_j p_j \sum_k q_{k|j} \log q_{k|j}.
\end{aligned}$$

Aqui $\sum_k q_{k|j} = 1$ para todo j, e a soma $-\sum_k q_{k|j} \log q_{k|j}$ pode ser considerada como a entropia condicional $\mathcal{E}_j(B)$ do esquema B, calculado sob o pressuposto de que o evento A_j tenha ocorrido. Em outras palavras, $\mathcal{E}_j(B)$ será a incerteza residual no esquema B, dado que o evento A_j tenha ocorrido no esquema A. Temos então

$$\mathcal{E}(AB) = \mathcal{E}(A) + \sum_j p_j H_j(B)$$

onde

$$H_j(B) = \sum_k q_{k|j} log q_{k|j}.$$

É interessante notar que $-H_j(B)$ pode ser visto como uma variável aleatória no esquema A, pois seu valor é completamente determinado pelo conhecimento de qual evento A_j do esquema A realmente ocorreu. Assim, o termo $-\sum_j p_j H_j(B)$ pode ser definido como a *esperança* da incerteza no esquema B após a realização do esquema A, e será denotado por $\mathcal{E}_A(B)$.

Assim, no caso geral, temos

$$\mathcal{E}(AB) = \mathcal{E}(A) + \mathcal{E}_A(B). \tag{3.7}$$

Essa relação também é desejável em uma medida de impureza; se A e B são mutuamente dependentes, a incerteza no esquema AB não pode ser $\mathcal{E}(A)+\mathcal{E}(B)$. Considere,por exemplo, o caso extremo em que a saída do esquema A determina unicamente a saída do esquema B, de forma que a ocorrência de um evento A_j em A implica na ocorrência de um evento B_k em B. Então, depois da realização do esquema A, o esquema B perde completamente sua incerteza e sua realização não fornece nenhuma informação adicional. Conseqüentemente, $\mathcal{E}_A(B) = 0$ e $\mathcal{E}(AB) = \mathcal{E}(A)$, e a relação 3.7 permanece válida. É fácil também ver que a igualdade 3.7 é equivalente à igualdade 3.6 no caso em que os eventos A e B são independentes.

Um fato interessante é notar que $\mathcal{E}_A(B) \leq \mathcal{E}(B)$. Essa desigualdade pode ser interpretada da seginte maneira: o conhecimento do resultado do esquema A só pode diminuir a incerteza do esquema B, jamais aumentá-la. Para provarmos este fato, usaremos a propriedade válida para toda função convexa $f(a)$:

$$\sum_j \lambda_j f(a_j) \geq f\left(\sum_j \lambda_j a_j\right),$$

onde $\lambda_j \geq 0$ e $\sum_j \lambda_j = 1$. Portanto, ajustando $f(a) = a \log a$, $\lambda_j = p_j$, $a_j = q_{k|j}$ temos, para todo k,

$$\sum_j p_j q_{k|j} \log q_{k|j} \geq \sum_j p_j q_{k|j} \log \left(\sum_j p_j q_{k|j}\right) = q_k \log q_k,$$

uma vez que $\sum_j p_j q_{k|j} = q_k$. Somando a desigualdade acima sobre k, obtemos

$$\begin{array}{rcrcl} \sum_j p_j \sum_k q_{k|j} \log q_{k|j} & = & -\sum_j p_j \mathcal{E}_j(B) & = & -\mathcal{E}_A(B) \\ & \geq & \sum_k q_k \log q_k & = & -\mathcal{E}(B). \end{array}$$

□

3.4.1 O teorema da unicidade

Dentre as características da medida de entropia verificadas em nosso trabalho, duas podem ser consideradas básicas:

1. Para toda tupla $(p_1, p_2, \ldots, p_J)$ satisfazendo $p_j \geq 0$, $\sum_j p_j = 1$, tem-se $\mathcal{E}(p_1, p_2, \ldots, p_J) \leq \mathcal{E}(\frac{1}{J}, \frac{1}{J}, \ldots, \frac{1}{J})$.
2. $\mathcal{E}(AB) = \mathcal{E}(A) + \mathcal{E}_A(B)$.

Pode-se acrescentar uma terceira propriedade básica, a qual se poderia esperar de uma medida de impureza razoável. Uma vez que os esquemas

$$\begin{pmatrix} A_1 & A_2 & \dots & A_J \\ p_1 & p_2 & \dots & p_J \end{pmatrix} \text{ e } \begin{pmatrix} A_1 & A_2 & \dots & A_J & A_{J+1} \\ p_1 & p_2 & \dots & p_J & 0 \end{pmatrix}$$

não são substancialmente diferentes (a inclusão de um evento impossível não deve alterar a incerteza do sistema), temos

3. $\mathcal{E}(p_1, p_2, \dots, p_J, 0) = \mathcal{E}(p_1, p_2, \dots, p_J)$.

O teorema a seguir, demonstrado por [Khi53], garante que a medida de entropia é a única que satisfaz simultaneamente às três condições acima:

Teorema 3.4.1 *Seja $\mathcal{E}(p_1, p_2, \dots, p_J)$ uma função definida para todo inteiro J e para todos os valores $p_1, p_2, \dots, p_J$ tais que $p_j \geq 0$, $\sum_j p_j = 1$. Se para todo J esta função for contínua com respeito a todos os seus argumentos e se possuir as propriedades (1), (2) e (3) acima, então*

$$\mathcal{E}(p_1, p_2, \dots, p_J) = -\lambda \sum_{j=1}^{J} p_j \log p_j,$$

onde λ é uma constante positiva.

Dem. Por simplicidade de notação, consideremos

$$\mathcal{E}\left(\frac{1}{J}, \frac{1}{J}, \dots, \frac{1}{J}\right) = L(J).$$

Em primeiro lugar mostraremos que

$$L(J) = \lambda \log J = -\lambda \frac{1}{J} \sum_{j=1}^{J} \log \frac{1}{J}, \tag{3.8}$$

onde λ é uma constante positiva.

Pelas propriedades (1) e (3), temos

$$L(J) = \mathcal{E}\left(\frac{1}{J}, \frac{1}{J}, \dots, \frac{1}{J}, 0\right) \leq \mathcal{E}\left(\frac{1}{J+1}, \frac{1}{J+1}, \dots, \frac{1}{J+1}\right) = L(J+1),$$

e temos que $L(J)$ é uma função não-decrescente de J. Sejam m e r inteiros positivos. Considere m esquemas independentes $A_1, A_2, \dots, A_m$, cada um dos quais contendo r eventos equiprováveis, de forma que

$$\mathcal{E}(A_k) = \mathcal{E}\left(\frac{1}{r}, \frac{1}{r}, \dots, \frac{1}{r}\right) = L(r) \quad (1 \leq k \leq m).$$

Pela propriedade (2) (generalizada para o caso de m esquemas) e pela independência dos esquemas A_k, temos

$$\mathcal{E}(A_1A_2\ldots A_m) = \sum_{k=1}^{m} H(A_k) = mL(r).$$

Por outro lado, o esquema $A_1A_2\ldots A_m$ consiste de r^m eventos equiprováveis, e assim $\mathcal{E}(A_1A_2\ldots A_m) = L(r^m)$. Portanto,

$$L(r^m) = mL(r), \tag{3.9}$$

e, analogamente para outro par de inteiros positivos n e s,

$$L(s^n) = nL(s). \tag{3.10}$$

Agora, sejam os números r, s e n inteiros positivos dados arbitrariamente, mas seja o número m determinado pelas desigualdades

$$r^m \leq s^n \leq r^{m+1}, \tag{3.11}$$

logo

$$\begin{aligned} m\log r &\leq n\log s \leq (m+1)\log r, \\ \frac{m}{n} &\leq \frac{\log s}{\log r} \leq \frac{m}{n} + \frac{1}{n}. \end{aligned} \tag{3.12}$$

Pela desigualdade 3.11 e pela monotonicidade da função $L(n)$, temos

$$L(r^m) \leq L(s^n) \leq L(r^{m+1})$$

e, conseqüentemente, por 3.9 e 3.10

$$mL(r) \leq nL(s) \leq (m+1)L(r).$$

Assim,

$$\frac{m}{n} \leq \frac{L(s)}{L(r)} \leq \frac{m}{n} + \frac{1}{n}. \tag{3.13}$$

Finalmente, segue das desigualdades 3.12 e 3.13 que

$$\left|\frac{L(s)}{L(r)} - \frac{\log s}{\log r}\right| \leq \frac{1}{n}.$$

Uma vez que o lado esquerdo dessa desigualdade é independente de m, e pelo fato de n poder ser arbitrariamente grande (e assim a fração $1/n$ poder ser arbitrariamente pequena), temos

$$\frac{L(s)}{\log s} = \frac{Ł(r)}{\log r},$$

e pelo fato de r e s serem arbitrários, temos $L(s)/\log s = \lambda$, onde λ é uma constante; logo,

$$L(s) = \lambda \log s.$$

Pela monotonicidade da função $L(s)$, temos $\lambda \geq 0$, e assim acabamos de provar a igualdade 3.8.

Demonstramos que o teorema vale para o caso especial em que $p_j = 1/J \;\; (1 \leq j \leq J)$. Consideraremos agora o caso em que os argumentos de $\mathcal{E}$ são todos racionais. Seja A um esquema composto por J eventos de probabilidades $p_1, p_2, \ldots, p_J$, onde cada p_j é um número racional $(1 \leq j \leq J)$. Então os números p_j podem ser expressos como

$$p_j = \frac{g_j}{g} \;\; (1 \leq j \leq J),$$

onde todos os g_j são inteiros positivos e $\sum_j g_j = g$. Nosso problema consiste em definir a incerteza do esquema A. Para isso, definiremos um segundo esquema B, o qual depende de A e é definido como segue: o esquema B contém g eventos $B_1, B_2, \ldots, B_g$, os quais são divididos em J grupos, contendo $g_1, g_2, \ldots, g_J$ eventos, respectivamente (cada um desses grupos corresponde a um evento de A). Se o evento A_j ocorreu no esquema A, então no esquema B todos os g_j elementos do j-ésimo grupo têm a mesma probabilidade $1/g_j$, e todos os eventos dos demais grupos têm probabilidade 0 (são impossíveis). Pela propriedade (3) da função $\mathcal{E}$, dada uma saída A_j do esquema A, teremos

$$\begin{aligned} \mathcal{E}_j(B) &= \mathcal{E}(\frac{1}{g_j}, \frac{1}{g_j}, \ldots, \frac{1}{g_j}) \\ &= L(g_j) \\ &= \lambda \log g_j, \end{aligned}$$

o que significa que

$$\begin{aligned} \mathcal{E}_A(B) &= \sum_{j=1}^{J} p_j H_j(B) \\ &= \lambda \sum_{j=1}^{J} p_j \log g_j \\ &= \lambda \sum_{j=1}^{J} p_j \log p_j + \lambda \log g. \end{aligned} \tag{3.14}$$

Analisemos agora o esquema-produto AB, consistindo de todos os eventos $A_j B_k$ $(1 \leq j \leq J,\; 1 \leq k \leq g)$. Um evento desse esquema somente é possível se B_k pertencer ao j-ésimo grupo. Dessa forma, o número de esquemas possíveis $A_j B_k$ para um dado j é g_j, e o número total de eventos possíveis no esquema AB é $\sum_{j=1}^{J} g_j = g$. A probabilidade de cada evento possível A_jB_k é $p_j/g_j = 1/g$, isto é, é a mesma para todos os eventos possíveis. Dessa forma, o esquema AB consiste de g eventos equiprováveis, juntamente com $ng - g$ eventos impossíveis; Assim, pela propriedade (3),

$$\mathcal{E}(AB) = \mathcal{E}\left(\frac{1}{g}, \frac{1}{g}, \dots, \frac{1}{g}\right) = L(g) = \lambda \log g.$$

Usando a propriedade (2) e a relação 3.14, temos

$$\lambda \log g = \mathcal{E}(A) + \lambda \sum_{j=1}^{J} p_j \log p_j + \lambda \log g,$$

e portanto

$$\mathcal{E}(A) = \mathcal{E}(p_1, p_2, \dots, p_J) = -\lambda \sum_{j=1}^{J} p_k \log p_k. \tag{3.15}$$

Finalmente, a relação 3.15 que acabamos de demonstrar para números racionais $p_1, p_2, \dots, p_J$ é válida também para números reais quaisquer, por causa da hipótese de $\mathcal{E}(p_1, p_2, \dots, p_J)$ ser contínua. Assim, a demonstração do teorema está completa.

□

3.5 O Teste Exato de Fisher

Considere um problema em que são realizados n experimentos independentes. Considere também um conjunto de propriedades $P = \{P_1, P_2\}$ e um conjunto de resultados $R = \{R_1, R_2\}$, onde cada experimento possui uma (e somente uma) propriedade $P_i \in P$ e produz um (e somente um) resultado $R_j \in P$.

Para cada $i = 1, 2$ e cada $j = 1, 2$, sejam

- $n_{i,j}$ o número de experimentos que possuem a propriedade P_i e produzem o resultado P_j;
- $n_{i,j}$ o número de experimentos que possuem a propriedade i; $n_{i,j} = \sum_j n_{i,j}$;
- $n_{i,j}$ o número de experimentos que produzem a propriedade j; $n_{i,j} = \sum_i n_{i,j}$.

Os valores acima fornecem uma tabela de contingência Δ (figura 3.1), que relaciona as propriedades com os resultados verificados.

Desejamos agora saber se os valores da tabela de contingência indicam alguma dependência entre a propriedade P_i e o resultado R_j; em outras palavras, desejamos saber se

Δ

	R_1	R_2	Totais
P_1	$n_{1,1}$	$n_{1,2}$	$n_{1.}$
P_2	$n_{2,1}$	$n_{2,2}$	$n_{2.}$
Totais	$n_{.1}$	$n_{.2}$	n

Tabela 3.1: Tabela de contingência 2×2 para n experimentos.

	R_1	R_2	Totais
P_1	12	8	20
P_2	6	4	10
Totais	18	12	30

(a)

	R_1	R_2	Totais
P_1	18	2	20
P_2	0	10	10
Totais	18	12	30

(b)

Tabela 3.2: Duas tabelas de contingência hipotéticas.

o conhecimento da propriedade P_i de um experimento é relevante ou não para prevermos seu resultado R_j.

Este problema pode ser ilustrado através das tabelas da figura 3.2, que representam duas situações hipotéticas onde um pesquisador realiza uma série de 30 experimentos. Comparemos as duas situações:

- Suponha que o pesquisador tenha obtido a tabela (a). Pela tabela, 60% dos experimentos verificados produzem o resultado R_1, independentemente de possuírem a propriedade P_1 ou P_2. O pesquisador conclui então que as propriedades P_1 e P_2 parecem não alterar o resultado de um experimento, e que portanto não há nenhuma evidência de que os resultados e as propriedades em estudo sejam dependentes.

- Suponha agora que o pesquisador tenha obtido a tabela (b) da figura 3.2. Neste caso, 90% dos experimentos com propriedade P_1 verificados produziram o resultado R_1; e todos os experimentos com propriedade P_2 verificados produziram o resultado R_2; o pesquisador conclui então que existe uma alta correlação entre a propriedade de um experimento e seu resultado.

Estudaremos nesta seção o teste exato de Fisher, desenvolvido por [Yat34] e [Fis35], que fornece uma medida do grau de dependência entre as propriedades e os resultados dos experimentos.

O primeiro passo do teste é estabelecer a hipótese a ser verificada. É possível pensar em verificar a hipótese de dependência entre R e P. Infelizmente, a dependência assume um número muito grande de formulações. Uma vez que a independência estatística é mais fácil de ser formalizada, a abordagem adotada usualmente é testar a hipótese de P e R

serem *independentes*. A hipótese de independência é comumente denominada *hipótese nula* e é denotada por H_o.

Para $i = 1, 2$ e $j = 1, 2$, sejam

- $p_{i.}$ a probabilidade de um experimento possuir a propriedade P_i;
- $p_{.j}$ a probabilidade de um experimento produzir o resultado P_j;
- $p_{i,j}$ a probabilidade de um experimento possuir a propriedade P_i e produzir o resultado R_j.

A hipótese nula será formulada como:

$$H_0 : p_{i,j} = p_{i.}p_{.j} \;\; (i = 1, 2, \;\; j = 1, 2). \tag{3.16}$$

O segundo passo é assumir que as somas marginais $n_{i,j}$ e $n_{i,j}$ são fixas, isto é, assumir que são conhecidos o número de experimentos que possuem a propriedade P_i e o número de experimentos que produzem o resultado R_j. Uma vez fixadas as somas marginais, podemos escolher livremente um valor para apenas uma das freqüências $n_{i,j}$; essa freqüência determinará, em conjunto com as somas marginais, todas as demais.

A terceira etapa será estimar a probabilidade de ocorrência da configuração $n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2}$, dadas as somas marginais (fixas) $n_{1.}, n_{2.}, n_{.1}, n_{.2}$ e dado que P e R são independentes. Apresentaremos detalhadamente este cálculo.

A probabilidade de uma certa configuração $n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2}$ em um total de n experimentos é dada pela distribuição multinomial [MG63]

$$f_n(n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2}|n) = \frac{n!}{n_{1,1}n_{1,2}n_{2,1}n_{2,2}} p_{1,1}^{n_{1,1}} p_{1,2}^{n_{1,2}} p_{2,1}^{n_{2,1}} p_{2,2}^{n_{2,2}}. \tag{3.17}$$

Em n experimentos, a probabilidade de $n_{1.}$ experimentos possuírem a propriedade P_1 é dada pela distribuição binomial

$$f_P(n_{1.}|n) = \frac{n!}{n_{1.}!n_{2.}!} p_{1.}^{n_{1.}} p_{2.}^{n_{2.}}. \tag{3.18}$$

Analogamente, a probabilidade de $n_{.1}$ experimentos produzirem o resultado R_1 (num total de n experimentos) é

$$f_R(n_{.1}|n) = \frac{n!}{n_{.1}!n_{.2}!} p_{.1}^{n_{.1}} p_{.2}^{n_{.2}}. \tag{3.19}$$

Uma vez que as somas parciais $n_{1.}$ e $n_{.1}$ são independentes (isto é, a escolha de $n_{1.}$ não interfere na escolha de $n_{.1}$ e vice-versa), a probabilidade da ocorrência de tabelas de contingência cujas somas parciais sejam $n_{1.}$ e $n_{.1}$ será

$$f_P(n_{1.}|n) f_R(n_{.1}|n).$$

Assim, a probabilidade de ocorrência de uma configuração $n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2}$ dadas as somas parciais $n_{1.} = n_{1,1} + n_{1,2}$ e $n_{.1} = n_{1,1} + n_{2,1}$ será

$$p(n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2} | n_{1.}, n_{.1}, n) = \frac{f_n(n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2} | n)}{f_P(n_{1.}|n) f_R(n_{.1}|n)}.$$

Da hipótese de independência descrita em 3.16 e das equações 3.17, 3.18 e 3.19 obtemos

$$p(n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2} | n_{1.}, n_{.1}, n) = \frac{n_{1.}! n_{2.}! n_{.1}! n_{.2}!}{n! n_{1,1}! n_{1,2}! n_{2,1}! n_{2,2}!}. \qquad (3.20)$$

Conforme mencionado anteriormente, uma vez fixadas as somas marginais, uma configuração pode ser determinada apenas em função de uma das freqüências. Por essa razão, a probabilidade $p(n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2} | n_{1.}, n_{.1}, n)$ será denotada simplesmente por $p(n_{i,j} | n_{1.}, n_{.1}, n)$.

Finalmente, resta-nos validar (ou refutar) a hipótese nula. Para a tabela Δ, defina a função

$$\mathcal{F}(\Delta) = p(n^*_{i,j} | n_{1.}, n_{.1}, n) + p(n^*_{i,j} - 1 | n_{1.}, n_{.1}, n) + \ldots + p(0 | n_{1.}, n_{.1}, n),$$

onde $n^*_{i,j}$ é a menor das freqüências verificadas em Δ.

Os valores $n^*_{i,j}, n^*_{i,j} - 1, \ldots, 0$ determinam as configurações "mais extremas", isto é, os casos que mais se distanciam da hipótese nula (veja a figura 3.3). Se, para um certo parâmetro β (por exemplo, 0.025) tivermos $\mathcal{F}(n^*_{i,j}, n_{1.}, n_{.1}, n) < \beta$, isto significa que a probabilidade da configuração verificada, dada a hipótese de independência, é muito pequena, e portanto a hipótese nula não deve ser verdadeira. Daí, concluímos (de forma indireta) que existe algum grau de dependência entre P e R.

Vejamos um exemplo. Considere a tabela de contingência da figura 3.3(a), que representa as freqüências observadas num certo conjunto de experimentos aleatórios. Os casos mais extremos são mostrados na figura 3.3(b).

O menor valor verificado na tabela é $n_{2,1} = 2$. Utilizando o teste de Fisher, temos

$$\begin{aligned} \mathcal{F}(\Delta) &= \sum_{i=0}^{2} p(n_{2,1} - i | n_{1.} = 13, n_{.1} = 12, n = 30) \\ &= \frac{18!12!17!13!}{30!} \left(\frac{1}{10!3!2!15!} + \frac{1}{11!2!1!16!} + \frac{1}{12!1!0!17!} \right) \\ &\approx 0.00047, \end{aligned}$$

e concluímos que existe uma dependência estatística entre P e R.

Através do teste de Fisher é possível medir a correlação entre os atributos e as classes dos exemplos disponíveis no conjunto de treinamento, selecionando os atributos mais significativos para as divisões dos nós. Tal aplicação pode ser útil para algoritmos que geram árvores binárias, sobre domínios onde são consideradas apenas duas classes possíveis.

Tabela original Δ

	R_1	R_2	Totais
P_1	10	3	13
P_2	2	15	17
Totais	12	18	30

(a)

Casos mais extremos

	R_1	R_2	Totais
P_1	11	2	13
P_2	1	16	17
Totais	12	18	30

	R_1	R_2	Totais
P_1	12	1	13
P_2	0	17	17
Totais	12	18	30

(b)

Tabela 3.3: Um exemplo da aplicação do Teste de Fisher.

Considere um nó t de uma árvore de decisão. Seja s uma divisão possível para o nó t, e sejam t_v e t_f os nós descendentes resultantes da partição de t através de s.

Seja N_t o número total de exemplos que incidem sobre o nó t e sejam

- $n_{v.}$ e $n_{f.}$ as quantidades de exemplos que vão, respectivamente, para os nós t_v e T_f;
- $n_{.j}$ o número de exemplos da classe j incidentes sobre t;
- $n_{v,j}$ e $n_{f,j}$ o número de exemplos pertencentes à classe j que vão, respectivamente, para os nó t_v e t_f.

Esses valores fornecem uma tabela de contingência de ordem 2×2, apresentada na figura 3.4.

	1	2	Totais
t_v	$n_{v,1}$	$n_{v,2}$	$n_{v.}$
t_f	$n_{f,1}$	$n_{f,2}$	$n_{f.}$
Totais	$n_{.1}$	$n_{.2}$	N_t

Tabela 3.4: Tabela de contingência 2×2 obtida a partir da divisão s sobre o nó t.

O teste de Fisher é utilizado então como segue: para cada divisão possível s, calcule $\mathcal{F}(\Delta(s))$. A divisão s^* escolhida será

$$s^* = \arg\min_s \mathcal{F}(\Delta(s)).$$

3.6 Considerações Finais

Estudamos neste capítulo quatro métodos para a escolha do melhor atributo para a divisão dos nós não-terminais. Esses métodos buscam encontrar o atributo com maior grau de correlação com as classes, fazendo com que subconjuntos resultantes da divisão do nó sejam os mais "homogêneos" possíveis, ou seja, possuam maior número de exemplos de mesma classe.

Dois dos métodos apresentados são baseados no conceito de impureza. A medida de entropia é uma das mais famosas medidas de impureza, empregada em diversos campos da física e da teoria de informação, entre outras áreas. O índice de Gini também é uma importante função, que mede a variância sobre conjuntos com classes categóricas. Uma vantagem do índice de Gini é a possibilidade de se incorporar custos de erros diferenciados para as classes, como por exemplo em diagnósticos médicos. Também foi apresentado o critério Twoing, uma variante do índice de Gini.

Finalmente, foi apresentado o teste exato de Fisher, o qual mede probabilisticamente o grau de dependência entre a presença de uma certa característica (atributo) e a ocorrência de um certo resultado (classe).

O algoritmo Real apresenta um critério diferente dos apresentados neste capítulo para a escolha do melhor atributo. Este critério será apresentado na seção 5.4.

Capítulo 4

A Poda de Árvores de Decisão

4.1 Considerações Iniciais

Qualquer algoritmo de construção de árvores de decisão tenta encontrar as correlações entre os atributos dos objetos e a classificação dos mesmos. A força dessas correlações varia de acordo com o número de exemplos que suportam ou negam a relação observada. Algumas dessas correlações refletirão características genuínas do domínio, no qual haverá uma relação causal entre os atributos e a classe de um objeto (isto é, a classe é conseqüência dos atributos ou vice-versa) ou uma relação associativa (isto é, os atributos e a classe do objeto possuem a mesma causa) [CN87]. Outras serão encontradas ao acaso, devido à escolha particular dos exemplos presentes no sistema. Geralmente, as correlações verdadeiras envolvem um número maior de exemplos estando, assim, menos sujeitos a ruídos.

Uma vez que o número de exemplos incidentes nos nós inferiores diminui à medida que a árvore é expandida, a expansão exagerada da mesma faz com que sua precisão de classificação diminua. Isto ocorre devido a dois fatores: em primeiro lugar, porque são incluídos atributos com baixo poder preditivo (ou altamente correlacionados com atributos utilizados nos nós superiores), tornando a teoria gerada excessivamente especializada (over-fitting); em segundo lugar, porque os nós inferiores (especialmente as folhas) ficam altamente suscetíveis à interferência de ruídos.

Por outro lado, uma árvore muito pequena não usará toda a informação disponível no conjunto de treinamento, resultando novamente numa precisão menor na classificação.

Durante a construção de árvores de classificação surge, então, um novo problema: a árvore gerada deve possuir um tamanho satisfatório, não sendo excessivamente geral (muito pequena) nem excessivametne especializada (muito grande). Esse problema tem sido atacado sob duas maneiras.

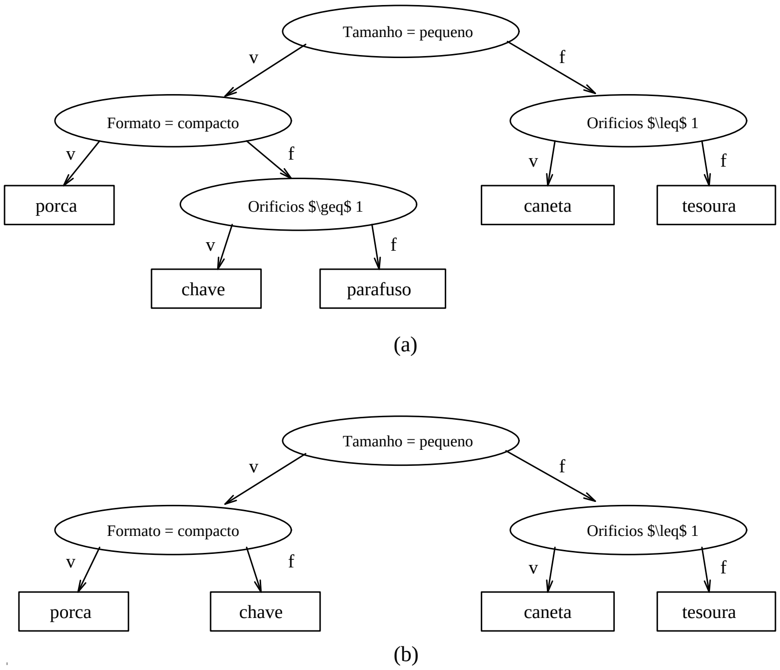


Figura 4.1: (a) Uma árvore binária para o problema da figura 2.3. (b) A árvore podada em um de seus ramos.

Em primeiro lugar, pode-se adotar, durante a construção da árvore de decisão, uma regra para interromper a divisão de um nó t (tornando-o um nó terminal), quando alguma condição especial (diferente daquelas expostas no algoritmo básico) ocorrer. Ao interromper a expansão sob essas condições especiais estaremos, com efeito, realizando uma poda na árvore. Por esse motivo, esse mecanismo é denominado *pré-poda* da árvore de decisão. Alguns critérios possíveis para declarar um nó t como terminal são:

1. Se, para toda divisão s do nó t, $\Delta I(s,t) < \beta$, onde $\beta > 0$ é um parâmetro definido pelo usuário.

2. Se o número de exemplos que incidir sobre t for inferior a um parâmetro n.

3. Se a proporção dos exemplos incidentes no nó em relação ao número total de exemplos em E for inferior a um parâmetro p.

4. Se a estimativa de erro (ou o custo de erro) $r(t)$ naquele nó for menor do que um parâmetro r.

A pré-poda é utilizada pelos sistemas Cal5 [MW94] (o qual será apresentado no próximo capítulo), C4 [QCHL86] e Assistant-86 [CKB87].

A segunda maneira de encontrar uma árvore de tamanho satisfatório é construir a árvore completa e avaliar a confiabilidade de cada uma de suas sub-árvores, podando os sub-ramos considerados não confiáveis. Este mecanismo é denominado *pós-poda* de árvores, e será o objeto de estudo deste capítulo.

Definiremos agora o processo de poda de forma mais precisa. Utilizaremos aqui alguns conceitos básicos apresentados na seção 2.3.

Definição 4.1.1 *Dada uma árvore T e um nó interno $t \in T$, a* pós-poda *(ou simplesmente* poda*) do ramo T_t de T consiste em remover todos os descendentes próprios de t, declarando-o como nó terminal. Rotula-se t conforme a função $classe(t)$*[1]*. A árvore podada desta forma será denotada por $T - T_t$.*

A figura 4.1 ilustra uma poda de árvore.

Definição 4.1.2 *Se T' é obtida de T através de podas sucessivas de ramos, então T' é denominada* sub-árvore podada *de T e é denotada por $T' \preceq T$.*

Ao longo deste capítulo, toda sub-árvore podada T' será chamada simplesmente de sub-árvore de T. Note que T' e T possuem o mesmo nó raíz.

4.2 O Método de Poda por Custo-Complexidade

Em linhas gerais, este método, proposto por [BFOS84] e implementado no algoritmo CART, é constituído de dois estágios. No primeiro estágio, uma seqüência de árvores $T_0, T_1, \ldots, T_K$ é gerada, onde T_0 é a árvore original e T_{k+1} é obtida pela substituição de uma ou mais sub-árvores de T_k por folhas. T_K é uma árvore constituída apenas por uma folha (a raíz da árvore original). No segundo estágio, é selecionada a melhor árvore dessa seqüência, levando-se em consideração o custo estimado dos erros de classificação e a complexidade (medida em número de folhas) de cada uma dessas árvores.

O primeiro passo é construir uma árvore suficientemente grande T_{max}. T_{max} não precisa ser expandida de forma exaustiva, isto é, até que todas as folhas sejam puras[2]. Se começarmos o processo com a maior árvore possível T'_{max} ou com uma árvore menor, mas suficientemente grande T_{max}, o processo de poda produz as mesmas sub-árvores podadas, no seguinte sentido: se o processo de poda começa com T'_{max} e produz uma sub-árvore contida em T_{max}, então o processo começando com T_{max} obtém exatamente a mesma sub-árvore.

[1] Descrita na seção 2.4.3.

[2] Um nó puro é aquele que contém exemplos de uma única classe.

Dessa forma, o critério adotado para obter uma árvore suficientemente grande T_{max} especifica um número n e continua o processo de expansão da árvore até que cada folha seja pura, contenha apenas vetores de medidas idênticos ou contenha um número de exemplos menor do que N_{min}.

A idéia principal da poda por custo complexidade é a que segue:

Definição 4.2.1 *Para qualquer subárvore $T \preceq T_{max}$, defina sua complexidade como $|\overline{T}|$, o número de folhas em T. Seja $\alpha > 0$ um número real denominado o* parâmetro de complexidade *e defina a* medida de custo-complexidade *$R_\alpha(T)$ como*

$$R_\alpha(T) = R(T) + \alpha|\overline{T}|.$$

$R_\alpha(T)$ é uma combinação linear entre o custo de erro da árvore e sua complexidade. O problema central do método é encontrar, para cada valor de α, a sub-árvore $T(\alpha) \preceq T_{max}$ que minimiza $R_\alpha(T)$, isto é,

$$T(\alpha) = \arg\min_{T \preceq T_{max}} R_\alpha(T).$$

O parâmetro α pode ser visto como um custo por folha; assim, se α for pequeno, a penalização por haver muitas folhas será pequena e $T(\alpha)$ será grande. À medida que a penalidade α por folha aumenta, a sub-árvore $T(\alpha)$ passa a ter um número menor de nós terminais até que, para um valor suficientemente grande de α, $T(\alpha)$ consistirá apenas do nó raíz e a árvore T_{max} terá sido completamente podada.

Embora α seja contínuo, existe um número finito de sub-árvores de T_{max} havendo, portanto, um número finito de sub-árvores $T(\alpha), \alpha \in [0, +\infty[$. O que ocorre é que, se $T(\alpha)$ é a árvore que minimiza $R_\alpha(T)$ para um certo valor de α, então essa mesma árvore continua minimizando $R_\alpha(T)$ à medida que α aumenta, até que um ponto de ruptura α' seja atingido, onde uma nova árvore $T(\alpha')$ (com menos folhas) se torna a árvore que minimiza $R'_\alpha(T)$ e continua sendo ótima até o próximo ponto de ruptura α'', e assim sucessivamente. Esse processo gera uma seqüência finita de sub-árvores $T_1, T_2, T_3, \ldots$ com número de folhas progressivamente menor.

Detalharemos agora um algoritmo eficiente para a obtenção dessa seqüência de sub-árvores (sem necessidade de buscar exaustivamente entre todas as possíveis sub-árvores para encontrar aquela que minimize $R_\alpha(T)$).

4.2.1 Obtenção das sub-árvores

Definição 4.2.2 *A* árvore minimal *$T(\alpha)$ para o parâmetro de complexidade α é definida pelas condições:*

- $R_\alpha(T(\alpha)) = \min_{T \preceq T_{max}} R_\alpha(T)$;
- *se* $R_\alpha(T) = R_\alpha(T(\alpha))$, *então* $T(\alpha) \preceq T$.

Inicialmente, devemos obter a sub-árvore $T_1 = T(0)$ a partir de T_{max}, ou seja, devemos encontrar a menor das sub-árvores de T_{max} que minimizam $R(T)$. A proposição a seguir tem como conseqüência o fato de toda sub-árvore T de T_{max} satisfazer $R(T) \geq R(T_{max})$; logo, T_1 será a menor sub-árvore de T_{max} satisfazendo

$$R(T_1) = R(T_{max}).$$

Proposição 4.2.3 *Seja T uma árvore binária. Dada uma folha $t \in T$, para qualquer divisão de t em t_v e t_f tem-se*

$$R(t) \geq R(t_v) + R(t_f).$$

Para encontrar T_1, execute o seguinte procedimento: seja t um nó interno de T, e sejam t_v e t_f os filhos de t. Se $R(t) = R(t_v) + R(t_f)$, então pode o ramo T_t, substituindo-o pelo nó t. Repita este processo até que já não seja mais possível nenhuma poda.

Acabamos de obter a sub-árvore $T_1 = T(\alpha_1)$, onde $\alpha_1 = 0$. Mostraremos agora como obter, a partir de uma sub-árvore $T_k = T(\alpha_k)$, a sub-árvore $T_{k+1} = T(\alpha_{k+1})$ (onde $\alpha_{k+1} > \alpha_k$), de tal forma que T_k permaneça uma árvore minimal para todo α no intervalo $[\alpha_k, \alpha_{k+1})$. Ou seja, para $\alpha_1 \leq \alpha < \alpha_{k+1},\ T_k = T(\alpha)$.

Para todo nó interno t de T_k e seu respectivo ramo $T_{k,t}$, defina

$$R_\alpha(t) = R(t) + \alpha \tag{4.1}$$
$$R_\alpha(T_{k,t}) = R(T_{k,t}) + \alpha|\overline{T_{k,t}}|. \tag{4.2}$$

Note que $R_{\alpha k}(T_{k,t}) < R_{\alpha k}(t)$ para todo nó interno de T_k, pois caso contrário T_k não seria minimal. Suponha agora que α comece a subir de forma contínua. Enquanto $R_\alpha(T_{k,t}) < R_\alpha(t)$ para cada t, teremos ainda $T_k = T(\alpha)$. No instante em que

$$R_\alpha(T_{k,t}) = R_\alpha(t) \tag{4.3}$$

para algum nó t, teremos então $R_\alpha(T_k) = R_\alpha(T_k - T_{k,t})$ e $T_i - T_{i,t}$ conterá menos folhas do que T_i; logo, $T_i \neq T(\alpha)$. Nesse momento, o ramo $T_{k,t}$ deverá ser podado.

As equações 4.1 e 4.2 fornecem o valor de α para o qual a igualdade 4.3 ocorre:

$$\alpha = \frac{R(t) - R(T_{k,t})}{|\overline{T_{k,t}}| - 1}.$$

O ponto crítico α_{k+1} será o menor valor de α para o qual a igualdade 4.3 pode ocorrer. Ou seja,

$$\alpha_{k+1} = \min_{t \in \overline{T_k}^C} \frac{R(t) - R(T_{k,t})}{|\overline{T_{k,t}}| - 1}$$

onde $\overline{T_k}^C$ é o conjunto de nós internos de T_k.

Para obter a árvore T_{k+1}, execute o seguinte procedimento: seja t um nó interno de T_k; se $R_{\alpha k+1}(t) = R_{\alpha k+1}(T_{k,t})$, pode o ramo $T_{k,t}$, substituindo-o pelo nó t. Repita este procedimento até que não seja mais possível nenhuma poda. A árvore resultante será T_{k+1}.

O procedimento acima gera uma seqüência de sub-árvores de T_{max}, $T_1 \succ \ldots \succ T_K$, onde $T_k = T(\alpha_k)$, $\alpha_1 = 0$. A sub-árvore T_K será constituída somente pela raíz $\{t_1\}$ de T_{max}.

4.2.2 Escolha da melhor sub-árvore

Uma vez obtida a seqüência decrescente de sub-árvores $T_1 \succ T_2 \succ \ldots \succ T_K \equiv \{t_1\}$, o estágio final do método de poda por custo-complexidade é escolher a melhor dessas sub-árvores. O critério para essa decisão é baseado na precisão de classificação e na complexidade de cada sub-árvore.

Inicialmente, deve-se encontrar estatisticamente uma boa estimativa de erro para cada uma das árvores. Para encontrar essa estimativa, não podemos simplesmente utilizar os mesmos exemplos que haviam sido empregados para a contrução da árvore, sob pena de tal estimativa de erro ser demasiadamente otimista. Por exemplo, suponha que a estimativa de erro empregada seja $R(T)$, conforme definido na subseção 2.4.3, usando o próprio conjunto de treinamento para obter $R(T)$. Pela proposição 4.2.3, $R(T)$ será menor na medida em que a árvore T tiver mais folhas; logo, a sub-árvore T_1 será sempre favorecida na escolha.

Para calcularmos a estimativa de erros das árvores, utilizaremos um *conjunto de testes* E_A, que consiste de um conjunto de instâncias cujas classes sejam conhecidas e que não tenham sido empregadas durante a construção da árvore T_{max}.

Seja T_k uma sub-árvore da seqüência. Submeta a árvore T_k ao conjunto de testes E_A, ou seja, utilize T_k para classificar cada uma das instâncias de E_A. Como a classe real de cada objeto é conhecida, é possível obter uma estimativa do custo de erro da árvore T_k (denotada por $R^C(T_k)$), conforme veremos a seguir.

Denote por N^A a cardinalidade do conjunto E_A. Sejam N_j^A o número de instâncias da classe j em E_A e N_{ij}^A o número de instâncias de classe j em E_A que tenham sido classificados por T_k como classe i. Então, a probabilidade estimada de um objeto de classe j ser classificado por T_k como classe i será

$$Q(i|j) = N_{ij}^A / N_j^A.$$

Denote por $R(j)$ o custo esperado no erro de classificação dos objetos de classe j. $R(j)$ será dado por

$$R(j) = \sum_{i=1}^{J} C(i,j) Q(i|j)$$

onde $C(i,j)$ é o custo de erro definido na subseção 2.4.3.

Finalmente, seja $\pi(j)$ a probabilidade *a priori* de um objeto qualquer de E_A ser de classe j. A estimativa do custo da árvore T_k é dada por,

$$R^C(T_k) = \sum_{j=1}^{J} R(j)\pi(j).$$

Após calculada a estimativa de custo $R^C(T_k)$ para cada sub-árvore T_k da seqüência, pode-se simplesmente escolher a sub-árvore

$$T_{k1} = \arg \min_{1 \leq k \leq K} R^C(T_k).$$

4.3 O Método de Poda por Erro Mínimo

Este método, proposto por [Nib87], tenta encontrar a sub-árvore T que, teoricamente, minimiza a taxa esperada de erro de classificação. Em nosso trabalho, procuramos detalhar o método, para sua melhor compreensão.

O primeiro passo será calcular o erro esperado de classificação em um certo nó t. Por questão de simplicidade, suponha inicialmente que o domínio possua somente duas classes 1 e 2, com probabilidades de ocorrência (desconhecidas) p_1 e p_2 (onde $p_2 = 1 - p_1$). Suponha que n exemplos incidam sobre o nó t, dentre os quais n_1 pertencem à classe 1 e n_2 $(= n - n_1)$ pertencem à classe 2. Podemos supor, sem perda de generalidade, que $n_1 \geq n_2$. Deveremos responder à seguinte questão: qual será o erro esperado se atribuírmos a classe 1 a um novo objeto incidente em t? Denotando por $f_p(A|B)$ a função de densidade da probabilidade do evento A dado o evento B, o erro esperado será

$$E(t) = \int_0^1 (1-x) f_p(p_1 = x | n_1) dx.$$

Note que

$$f_p(p_1 = x | n_1) = \frac{p(n_1 | p_1 = x) f_p(p_1 = x)}{p(n_1)}$$

e que, pela fórmula de Bayes,

$$p(n_1) = \int_0^1 p(n_1|p_1{=}x)dx.$$

Assim,

$$E(t) = \int_0^1 \frac{(1-x)p(n_1|p_1{=}x)f_p(p_1{=}x)}{\int_0^1 p(n_1|p_1{=}x)f_p(p_1{=}x)dx}dx.$$

Assuma que p_1 possui uma distribuição *a priori* tal que $f_p(p_1{=}x) = 1$.

$$E(t) = \frac{\int_0^1 (i-x)p(n_1|p_1{=}x)dx}{\int_0^1 p(n_1|p_1{=}x)dx}.$$

Agora, $p(n_1|p_1{=}x) = \binom{n}{n_1} x^{n_1}(1-x)^{n-n_1}$; logo,

$$E(t) = \frac{\int_0^1 (i-x)x^{n_1}(1-x)^{n-n_1}dx}{\int_0^1 x^{n_1}(1-x)^{n-n_1}dx}. \tag{4.4}$$

Note que

$$\begin{aligned}
\int_0^1 x^a(1-x)^b dx &= \int_0^1 \sum_{i=0}^{b} \binom{b}{i} (-1)^i x^{a+i} dx \\
&= \sum_{i=0}^{b} (-1)^i \binom{b}{i} \int_0^1 x^{a+i} dx \\
&= \sum_{i=0}^{b} \frac{(-1)^i \binom{b}{i}}{a+i+1} \\
&= \frac{n!}{\prod_{i=0}^{b} a+i+1}.
\end{aligned}$$

Logo, da equação 4.4 resulta

$$E(t) = \frac{\sum_{i=0}^{n-n_1+1} \frac{(-1)^i \binom{n-n_1+1}{i}}{n_1+i+2}}{\sum_{i=0}^{n-n_1} \frac{(-1)^i \binom{n-n_1}{i}}{n_1+i+1}} = \frac{n-n_1+1}{n+2}.$$

O cálculo acima pode ser generalizado para domínios com múltiplas classes. Denote por J o número de classes e por n_j o número de exemplos de classe j incidentes sobre t. Seja

$$i^* = \arg \max_{j \in \{1,2,\ldots,J\}} n_j.$$

Supondo que a distribuição de probabilidade *a priori* para cada classe seja uniforme (com valor constante $1/J$), se definirmos $classe(t) = i^*$, o erro esperado no nó t será

$$E(t) = \frac{n - n_{i^*} + J - 1}{n + J}.$$

Defina agora a função $Erro(t)$ como o erro esperado de classificação no ramo T_t. Esta função será calculada como segue:

- se t for um nó folha, $Erro(t) = E(t)$;
- se t for um nó não-terminal, denote por t_v e t_f os filhos de t; sendo n_v o número de exemplos incidentes sobre t_v, defina $p_v = n_v/n$ (analogamente para n_f e p_f); então, o erro esperado sobre o ramo T_t será

$$Erro(t) = p_v Erro(t_v) + p_f Erro(t_f).$$

Os erros esperados em cada nó t e em cada ramo T_t sugerem o seguinte algoritmo: Seja T uma árvore não podada. Partindo das folhas e subindo em direção à raíz, calcule para cada nó t os erros esperados $E(t)$ e $Erro(t)$. Se $E(t) \leq Erro(t)$, execute a poda do ramo T_t, mantendo apenas o nó t; defina $classe(t) = \arg\max_i n_i$ e atualize $Erro(t) = E(t)$.

Após a execução deste procedimento para todos os nós (na ordem citada, isto é, partindo-se dos nós inferiores da árvore até a raíz), teremos obtido uma sub-árvore $T' \preceq T$ cujo erro de classificação esperado será mínimo (em relação ao conjunto de treinamento E). Isto é facilmente verificado por indução na altura da árvore.

Teoricamente, este método de poda é o ideal, uma vez que minimiza o erro global esperado e não necessita de um conjunto separado para testes. Entretanto, possui algumas desvantagens. A primeira delas é que a premissa de todas as classes serem equiprováveis raramente é verdadeira (basta notar que os nós inferiores das árvores privilegiam certas classes em detrimento de outras). Além disso, através de testes empíricos, [Min89] concluiu que o número de classes afeta drasticamente o tamanho da árvore podada, levando a resultados instáveis.

4.4 Considerações Finais

Todo algoritmo TDIDT precisa buscar um equilíbrio adequado para o nível de especialização da árvore construída. Se a árvore resultante for excessivamente genérica (pequena demais), poucos atributos estarão sendo utilizados e muita informação importante poderá ser perdida, comprometendo assim a precisão na classificação. Por outro lado, uma árvore excessivamente especializada (muito grande) poderá estar empregando atributos com baixo poder preditivo, prejudicando a precisão na classificação de novos objetos. Em domínios muito ruidosos, esse problema se torna ainda mais crítico.

Para adequar o tamanho das árvores, são possíveis duas abordagens: a primeira é interromper a expansão da árvore quando certas condições são satisfeitas (esse processo é denominado *pré-poda* da árvore); a segunda é expandir o máximo possível a árvore, e realizar a poda de alguns de seus ramos inferiores (esse processo é denominado *pós-poda*, ou simplesmente *poda*).

Neste capítulo, foram apresentados dois métodos de poda.

O primeiro, denominado poda por custo-complexidade, procura gerar uma árvore com a melhor relação *precisão x número de folhas*. Este método necessita de um conjunto de testes separado do conjunto de treinamento, o que pode representar uma desvantagem quando os exemplos disponíveis são escassos.

O segundo método, denominado poda por erro mínimo, busca gerar uma sub-árvore com a menor taxa estimada de erro. Embora não necessite de um conjunto separado de testes, assume a hipótese de que as classes sejam equiprováveis, o que raramente é verdadeiro.

O programa NewId, o único dos programas analisados em nosso trabalho a realizar pós-poda das árvores, possui um método de poda diferente dos aqui descritos, o qual será apresentado na seção 5.2.

Capítulo 5

Os Programas Utilizados

5.1 Considerações Iniciais

Em nosso trabalho, realizamos testes com três programas da família TDIDT para avaliar sua utilidade como ferramentas de suporte à decisão no mercado de ações. Foram avaliados os programas NewId, Cal5 e Real.

Este capítulo apresenta as descrições dos programas estudados. Para sua melhor compreensão, faremos uma breve revisão da estrutura geral dos algoritmos TDIDT (uma apresentação mais completa da família TDIDT se encontra no capítulo 2).

Algoritmo TDIDT:

- Seja t um nó terminal da árvore que se deseja construir. Seja E o conjunto de treinamento incidente sobre t. Se E satisfizer à condição de parada $P(E)$, rotule t conforme a regra $classe(t)$.

- Se E não satisfizer à condição de parada $P(E)$:

  1. calcule os valores da função $score(E, a_i)$, e escolha o atributo a_n^* que maximiza esta função. Rotule o nó t com a_n^*.
  2. Categorize o atributo a_n^* através da função $categ(E, a_n^*)$, e crie um ramo correspondente a cada intervalo gerado.
  3. Particione o conjunto de treinamento E de acordo com os ramos obtidos, e aplique recursivamente o algoritmo para cada nó filho de t.
  4. Ajuste o tamanho da árvore resultante, podando alguns ramos inferiores se necessário.

Todos os algoritmos apresentados neste capítulo possuem a estrutura acima. Os componentes particulares de cada algoritmo são:

- O critério de parada, $P(E)$;
- A função de atribuição de classe a um nó terminal, $classe(t)$;
- A função de escolha do melhor atributo, $score(E, a_i)$;
- A função de categorização de atributos numéricos, $categ(E, a_i)$;
- O critério de poda dos ramos inferiores da árvore.

Uma vez construída a árvore, o Real e o NewId não realizam a poda dos ramos inferiores. Nestes algoritmos, o controle do tamanho da árvore é realizado pelo critério de parada.

5.2 O NewID

O NewID [Bos90] foi implementado pelo Turing Institute, Escócia. É um algoritmo capaz de tratar, além de atributos numéricos, também classes numéricas (contínuas). Os demais algoritmos vistos neste capítulo somente lidam com classes categóricas. É o único dos três algoritmos que realiza a pós-poda.

Os parâmetros desse algoritmo, descritos mais detalhadamente nas seções que seguem, são:

fator de poda ϕ: através desse parâmetro, o algoritmo decide pela poda ou não de alguns ramos inferiores da árvore gerada; $0 \leq \phi < 100$.

coeficiente de variância ν: é utilizado para interromper a expansão da árvore quando as classes são do tipo ordenado; $0 < \nu \leq 100$.

5.2.1 A atribuição de classes aos nós terminais

Para classes categóricas, a regra para rotulação de um nó terminal t é atribuir a classe de maior freqüencia no nó.

Para classes do tipo ordenado, a regra é atribuir a média ponderada dos valores das classes que aparecem no nó.

5.2.2 A condição de parada

Para classes categóricas, o NewID interrompe a expansão da árvore em um nó quando as classes dos exemplos sobre esse nó são idênticos, ou quando os valores dos atributos forem idênticos para todos os exemplos.

Para classes do tipo numérico, o algoritmo considera t como um nó terminal quando

$$\sigma(E_t) \leq \frac{1}{\nu\sigma(E)}$$

onde E é o conjunto de treinamento original, E_t é o conjunto de exemplos incidentes sobre t, $\sigma(S)$ é o desvio-padrão das classes dos exemplos de S, e ν é um fator de poda fornecido pelo usuário, já mencionado no início da seção.

5.2.3 Medida para avaliação dos atributos ("score")

Para classes do tipo categórico, o programa utiliza a medida de entropia, descrita no capítulo 3.

Para classes do tipo ordenado, o programa tenta encontrar partições que minimizem as variâncias dos valores das classes nos nós descendentes:Para cada atributo a_n, selecione a bipartição $s_n^* = (I_{n1}^*, I_{n2}^*)$ tal que

$$s_n^* = \arg\min_{sn} \sigma(E_{t1}) + \sigma(E_{t2}),$$

onde $\sigma(S)$ é a variância das classes no conjunto S e E_{t1}, E_{t2} são os sub-conjuntos obtidos de E_t pela partição s_n.

5.2.4 O método de poda

Para os domínios com classes categóricas, o NewID implementa o seguinte método de poda descrito a seguir. O programa necessita de um conjunto de testes separados E_A e um fator de poda $\phi \in [0, 100]$.

Sejam j^* a classe de maior freqüência entre os exemplos do conjunto original incidentes sobre t, p_T a proporção de exemplos de E_A classificados corretamente por T_t, p_t a proporção de exemplos de E_A classificados corretamente por t se a este nó fosse atribuída a classe j^*.

Se $p_T > p_t\sigma/100$, mantenha o ramo T_t; caso contrário, pode o ramo e atribua a classe j^* a t.

5.3 O Cal5

A meta do Cal5[MW94] é construir árvores de decisão onde, para cada nó terminal t, pelo menos uma classe j possa ser decidida com probabilidade

$$P(j|t) \geq S \tag{5.1}$$

sendo $P(j|t)$ a probabilidade da classe j no nó t. A desigualdade 5.1 será estimada dentro do nível de confiança $1 - \alpha$. Tanto S quanto α são parâmetros que devem ser fornecidos pelo usuário. O método para a estimação de $P(j|t)$ será visto nas seções que seguem.

A condição de parada e a função de atribuição de classes a nós terminais serão descritas juntamente com a função de categorização.

5.3.1 Escolha do melhor atributo

Uma vez definido que um nó t precisa ser expandido, a escolha do melhor atributo para a expansão de t obedece ao seguinte procedimento: cada atributo a_n é discretizado em intervalos $I_{n1}, \ldots, I_{nK}$, pelo método que apresentaremos na próxima seção. Após a discretização de todos os atributos, será selecionado aquele que minimizar a entropia resultante[1].

5.3.2 Discretização dos atributos numéricos

Seja a_n um atributo numérico a ser discretizado. O primeiro passo para sua discretização é ordenar o conjunto E_t de exemplos que incidem sobre t, em ordem crescente dos valores do atributo a_n. O segundo passo será formar recursivamente os intervalos sobre esse atributo, coletando os exemplos em ordem crescente, até que se possa decidir se é possível ou não atribuir uma classe aos exemplos desse intervalo, dentro do nível de confiança $1 - \alpha$.

Seja I o intervalo corrente contendo n exemplos, dentre os quais n_j possuem classe j. Seja $P(j|I)$ a probabilidade de ocorrência de exemplos de classe j no intervalo I. Uma estimativa de $P(j|I)$ pode ser fornecida por $p(j|I) = n_j/n$.

Testaremos a hipótese

H_1: "Existe uma classe j em I que satisfaz

$$P(j|I) \geq S."$$

contra a hipótese

[1] Ver capítulo 3

H_2: "Para todas as classes j que ocorrem em I, a desigualdada

$$P(j|I) < S$$

vale dentro do nível de confiança $1 - \alpha$."

A partir do nível de confiança dado, é calculado um intervalo de confiança $[d_1, d_2]$ para $P(j|I)$. Este intervalo é obtido através da desigualdade de Tschebyscheff, usando-se o estimador $p(j|I)$ e assumindo uma distribuição de Bernoulli para as classes j:

$$d_{1,2}(j) = \frac{2\alpha n_j + 1}{2\alpha n + 2} \pm \frac{1}{2\alpha n + 2}\sqrt{4\alpha n_j \left(1 - \frac{n_j}{n}\right) + 1}.$$

A partir desse intervalo de confiança, as hipóteses H_1 e H_2 são testadas das seguinte maneira:

H_1: Existe uma classe j ocorrendo em I tal que

$$d_1(j) > S.$$

H_2: Para todas as classes j que ocorrem em I, vale

$$d_2(j) < S.$$

As seguintes meta-decisões são tomadas neste ponto:

1. Se existe uma classe j que satisfaz H_1, então j é a classe que domina em I. O intervalo I é fechado e rotulado com a classe j.

2. Se para todas as classes que ocorrem em I a hipótese H_2 é verdadeira, então nenhuma classe domina em I. O intervalo também é fechado e o próximo intervalo é iniciado.

3. Se não ocorre nem 1 nem 2, não há como decidir se existe ou não uma classe que domine em I. O próximo exemplo é incluído no intervalo corrente, calculando-se novamente o intervalo de confiança $[d_1, d_2]$ e testando-se as hipóteses H_1 e H_2. Se não houver mais exemplos a serem incluídos, o intervalo será rotulado com a classe de maior freqüência.

Após a expansão dos intervalos, poderá haver concatenações de intervalos adjacentes. Intervalos I_{nk}, I_{nk+1} rotulados com a mesma classe serão concatenados. Os intervalos resultantes dessa concatenação constituirão os nós terminais da árvore.

A mesma regra se aplica a classes adjacentes onde nenhuma classe domina (condição 2), e que permanecerem com classes idênticas após aplicada a seguinte heurística: Cal5 elimina uma classe j dentro de um intervalo I_{nk}, se valer a desigualdade

$$d_2(j) < \frac{1}{n_I},$$

onde n_I é o número total de classes distintas que ocorrem em I. Os intervalos resultantes desse tipo de concatenação formarão os nós internos, os quais deverão ser expandidos recursivamente.

5.4 O Real

O Real (Real-Valued Attribute Learning Algorithm)[SNL96] foi desenvolvido conjuntamente com o Prof. Júlio M. Stern, durante o desenvolvimento de nossa dissertação.

Os procedimentos desse algoritmo estão baseados em uma função de perda e uma função de convicção, descritas a seguir.

5.4.1 Função de convicção

Seja t um nó folha de classe j com n exemplos, dentre os quais m são classificados com erro e $(n-m)$ são classificados corretamente. Seja q a probabilidade de um exemplo ser erroneamente classificado em t, $p = 1 - q$ a probabilidade de classificação correta, e suponha que q possui uma distribuição Bayesiana

$$D(c) = Pr(q \leq c) = Pr(p \geq 1 - c).$$

Definimos a medida de convicção: $100 * (1 - cm)\%$, onde

$$cm = \min c | Pr(q \leq c) \geq 1 - g(c)$$

e $g(c)$ é uma bijeção convexa de $[0,1]$ em si mesmo. A convexidade de $g(c)$ nos torna “cautelosos ao fazer afirmações fortes”. Esta característica do algoritmo resultou de necessidades reais manifestadas por analistas de mercado com os quais interagimos.

Na implementação atual do Real:

- $g(c) = c^r$, onde $r \geq 1.0$ é um parâmetro de convexidade fornecido pelo usuário;

- $D(c)$ é a função beta incompleta derivada da distribuição de Bernoulli:

$$\begin{aligned} B(n,m,q) &= comb(n,m) * q^m * p^{n-m} \\ D(c,n,m) &= \int_{q=0}^{c} B(n,m,q) / \int_{q=0}^{1} B(n,m,q) \\ &= betainc(c, m+1, n-m+1). \end{aligned}$$

Com estas escolhas, cm é dada por

$$\begin{aligned} cm(n,m,r) &= c | f(c) = 0 \\ f(c) &= 1 - g(c) - D(c,n,m) \\ &= 1 - c^r - betainc(c, m+1, n-m+1). \end{aligned}$$

5.4.2 Função de avaliação dos atributos

Para selecionar o melhor atributo que expandirá o nó t, o Real discretiza cada atributo a_n em intervalos $I_1, \ldots, I_K$, pelo método que apresentaremos a seguir. Após a discretização de todos os atributos, será selecionado aquele que minimizar a função de perda, definida por

$$loss = \sum_k n_k * cm(n_k, m_k, r)$$

onde n_k é o número de exemplos no intervalo I_k e m_k é o número de exemplos classificados erroneamente em I_k.

5.4.3 Função de categorização de atributos

O primeiro passo do procedimento de discretização, para um atributo selecionado, é ordenar os exemplos do nó t em ordem crescente dos valores do atributo, e então agrupar os exemplos adjacentes de mesma classe.

Nos passos subseqüentes, agruparemos os intervalos de maneira a diminuir a perda global dentro do nó. O ganho em agrupar H intervalos adjacentes $I_{k+1}, I_{k+2}, \ldots, I_{k+H}$ é o decréscimo relativo na função de perda

$$gain(k,h) = \sum_h loss(n_h, k_h, r) - loss(n, h, r)$$

onde $n = \sum_h n_h$ e k é o número de exemplos em minoria no novo intervalo.

A cada passo, são concatenados os intervalos com ganho máximo. O procedimento de discretização pára quando não há mais agrupamentos com ganho positivo.

Cada intervalo obtido pelo procedimento acima constitui um novo nó, que deverá ser recursivamente expandido.

Após a discretização descrita, podem ocorrer nós adjacentes que poderiam ser melhor expandidos se fossem agrupados, ao invés de serem expandidos isoladamente. Isso porque pode haver outros atributos que consigam discriminar bem os exemplos contidos nesses nós, melhorando assim a qualidade da árvore gerada. Por essa razão, são reagrupados todos os intervalos (e seus respectivos nós) adjacentes que não satisfazem $cm < crv$, onde $crv \in [0, 1]$ é um grau de convicção a ser fornecido pelo usuário. Para prevenir um loop infinito, a função de perda associada ao novo intervalo é a soma das funções de perda dos intervalos a serem reagrupados. Nas folhas, este reagrupamento é desfeito.

5.4.4 Condição de parada e atribuição de classes

A expansão de um nó é interrompida quando não houver nenhum atributo cuja discretização diminua a função de perda por um fator $\epsilon > 0$. O nó é então rotulado pela classe majoritária.

O grau de convicção crv funciona também como um critério de parada do algoritmo (quanto maior o valor de crv, menor a árvore obtida).

5.5 Considerações Finais

Neste capítulo, apresentamos os três programas que foram avaliados em nosso trabalho. Os programas apresentados possuem o mesmo algoritmo básico, diferindo entre si nos seguintes aspectos: o critério de parada, a atribuição de classe a um nó terminal, o critério de escolha do melhor atributo para partição de um nó, a função de categorização de atributos numéricos e o critério de poda dos ramos inferiores da árvore.

O NewId é um algoritmo que binariza os atributos numéricos. Para a escolha do melhor atributo, utiliza a medida de entropia. Após a expansão da árvore, o algoritmo realiza a poda dos ramos inferiores, utilizando para isso um conjunto separado de testes.

O Cal5 e o Real se diferenciam do NewId por apresentarem um critério mais complexo de categorização dos atributos numéricos, permitindo que os mesmos sejam particionados em mais do que dois subconjuntos. Conseqüentemente, esses algoritmos podem gerar árvores com menor profundidade e, portanto, mais eficientes computacionalmente. Nem o Cal5 nem o Real possuem procedimentos de pós-poda.

Capítulo 6

Aplicação dos Algoritmos TDIDT ao Mercado de Ações

6.1 Considerações Iniciais

Vimos nos capítulos anteriores alguns algoritmos de construção de árvores da família TDIDT. Uma vez que nosso interesse era o de avaliar o desempenho desses algoritmos no mercado de ações, foram realizados alguns testes com três programas disponíveis (Cal5, NewID e Real, apresentados no capítulo 5). Neste capítulo, apresentaremos a metodologia empregada para a aplicação dos algoritmos no apoio à tomada de decisão para compra e venda de papéis. Ao final do capítulo, serão apresentados os resultados obtidos.

O processo de testes compreende basicamente duas fases. A primeira consiste em formar um conjunto de exemplos aos quais os algoritmos possam ser aplicados. A segunda etapa consiste na construção das árvores de decisão e posterior teste de precisão destas árvores.

Os exemplos são obtidos a partir de observações diárias de preços de alguns papéis com alta liquidez e alto volume de negociação na BOVESPA. Além das cotações dos papéis, são incorporados alguns indicadores técnicos, calculados diretamente a partir dessas cotações. Esses indicadores passarão a ser adotados como atributos para os exemplos. O critério de classificação de cada exemplo é o seguinte: estabelece-se uma certa estratégia de compra e venda do papel, a qual é aplicada para cada um dos exemplos. Se a aplicação desta estratégia sobre o exemplo resultar numa certa taxa mínima de lucro, o exemplo será classificado com bem-sucedido; caso contrário, será classificado como mal-sucedido.

6.2 O Cálculo dos Indicadores

Para a realização dos testes, foram selecionados dois papéis de alta liquidez e grande volume de negociação na BOVESPA. Para os papéis selecionados, foram obtidas observações diárias de preços e de volume negociado. As bases de dados com as observações nos foram gentilmente cedidas pelo Prof. Jacob Zimbarg Sobrinho.

As informações disponíveis para cada dia observado são:

- H_t - preço máximo do papel no dia t;
- L_t - preço mínimo do papel no dia t;
- O_t - preço de abertura no dia t;
- C_t - preço de fechamento no dia t;
- M_t - preço médio no dia t;
- V_t - volume negociado no dia t.

Os preços são dados em dólar comercial (venda), o que atenua os efeitos da desvalorização da moeda nacional sobre a variação dos mesmos.

Além dos dados observados, são necessários também alguns atributos que descrevam o comportamento passado dos preços. Isso porque nosso objetivo, ao adotar algoritmos de classificação, é correlacionar o comportamento passado da cotação de um papel com seu comportamento futuro. Em outras palavras, a pergunta que fazemos é: até que ponto o conhecimento do comportamento de preços num certo período nos permite “prever” a tendência desses preços no período subseqüente?

A alternativa para capturar o comportamento dos preços dentro de certos períodos foi calcular alguns parâmetros numéricos a partir das séries temporais dos mesmos. Esses parâmetros são os chamados indicadores técnicos (daqui por diante denominados apenas como indicadores). Um indicador é uma função de uma ou mais variáveis observadas que tem por objetivo auxiliar a tomada de decisão quanto a comprar ou vender papéis. Um exemplo de indicador é

$$H_t(r) = \max\{H_t, H_{t-1}, \ldots, H_{t-r}\},$$

ou seja, o preço máximo atingido por um papel nos últimos r+1 dias. Uma boa referência sobre os indicadores técnicos pode ser encontrada em [CM88].

Um total de 34 indicadores foram selecionados e implementados por Celma Ribeiro. Os indicadores foram incorporados à base de dados e foram utilizados - juntamente com as observações - como atributos para os exemplos.

6.3 A Estratégia de Operação Considerada

Os exemplos foram classificados levando-se em conta uma certa *estratégia de operação*. Em linhas gerais, uma estratégia de operação pode ser descrita como uma política para realização de compras e vendas de papéis, de acordo com critérios bem estabelecidos.

Procuramos adotar uma estratégia que fosse relativamente simples e que oferecesse um baixo risco, isto é, que limitasse a perda do investidor. A estratégia básica adotada foi a seguinte:

CompraVenda(t, l, p, d, c) : Dados os preços de abertura de um papel num certo período t, $t+1$, ..., $t+d$, compre o papel no dia t e venda-o nas seguintes condições:

quando seu preço de abertura tiver subido $l\%$ em relação ao dia t;
quando seu preço de abertura tiver caído $p\%$ em relação ao dia t;
quando tiverem transcorrido d dias sem que nenhuma das condições acima tenham ocorrido.

Os parâmetros l, p e d têm a finalidade de limitar os riscos do investidor; por essa razão, são comumente denominados *travas*.

Além das travas, incorporamos também um quarto parâmetro: o custo percentual da operação no mercado, denotado por c. Esse parâmetro indica se a estratégia foi bem-sucedida: se o rendimento obtido com a aplicação da estratégia for maior do que c, isso indica que houve um lucro real e, portanto, a estratégia foi bem-sucedida. Caso contrário, a aplicação da estratégia foi mal-sucedida, pois seu rendimento foi inferior ao custo da operação.

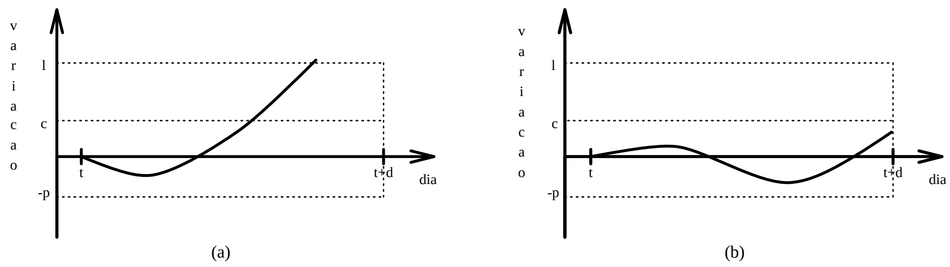


Figura 6.1: Dois exemplos de aplicação da estratégia CompraVenda(t, l, p, d, c).

Os gráficos apresentados na figura 6.1 ilustram a aplicação da estratégia acima sobre duas situações. As travas l, $-p$, d e o custo c são apresentados pelas linhas pontilhadas paralelas aos eixos.

O papel foi comprado no dia t, sendo seu preço observado nos dias subseqüentes. A linha contínua do gráfico representa a variação no preço do papel antes de sua venda. No momento em que uma das travas foi atingida, o papel foi vendido. Na situação descrita pelo gráfico (a), a aplicação da estratégia foi bem-sucedida, enquanto na situação descrita em (b) a estretégia foi mal-sucedida.

6.4 Obtendo Exemplos para os Sistemas

Nesta seção veremos como foram construídos os conjuntos de exemplos, tanto para treinamento quanto para testes dos sistemas. Na aplicação dos testes, cada um dos papéis foi considerado isoladamente, isto é, cada conjunto de exemplos foi construído utilizando-se um único papel.

Após obtidas as observações dos preços (ver seção 6.2), o primeiro passo foi "limpar" a base de dados, eliminando-se os dias sem observação. Todavia, dada a alta liquidez dos papéis, tivemos poucos "missing values". O segundo passo foi o cálculo dos indicadores técnicos descritos ($H_t(r_1), RSI_t(r_2)$, etc), formando com esses indicadores e com os dados observados no dia t o vetor de atributos. Para o cálculo dos indicadores em cada dia t, foi considerado o período $t-5$, $t-4$, $\ldots$, $t-1$, t.

A terceira etapa foi a atribuição de uma classe $Classe(t)$ a cada exemplo, isto é, a cada dia t. Foram observados os dias posteriores a t, testando-se a aplicação da estratégia CompraVenda(t,l,p,d,c). Sendo $rendimento(t)$ o rendimento obtido com a venda do papel segundo a aplicação da estratégia CompraVenda(t,l,p,d,c), $Classe(t)$ foi definida como

$$Classe(t) = \begin{cases} \text{SUCESSO} & \text{se } rendimento(t) > c \\ \text{FRACASSO} & \text{se } rendimento(t) \leq c \end{cases}$$

Os parâmetros considerados em nosso trabalho para a aplicação da estratégia CompraVenda(t,l,p,d,c)foram $l=3, p=1, d=5, c=1$.

A figura 6.2 representa um exemplo, formado pelo vetor de atributos e pela classe (SUCESSO ou FRACASSO).

Observações					Indicadores					$Classe(t)$
H_t	L_t	O_t	...	V_t	$H_t(r)$	$L_t(r)$	$RSI_t(r)$	$\%K_t(r)$	...	C_t

Figura 6.2: Composição de um exemplo: vetor de atributos + classe.

A figura 6.3 fornece um esquema cronológico para a obtenção de um exemplo. Note-se que cada exemplo representa, em última análise, o comportamento do preço do papel num certo perído (anterior e posterior ao dia t). Numa tentativa de manter a independência entre os exemplos, fizemos com que a distância entre dois exemplos consecutivos fosse de $d+1$ dias, ou seja, o número de dias necessário para a análise da estratégia

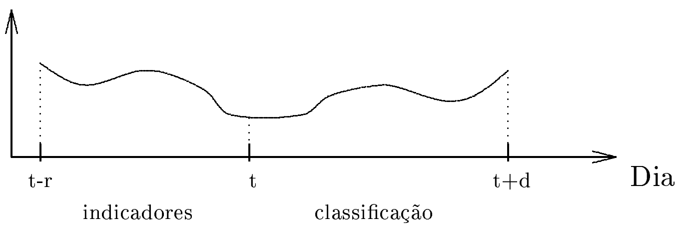


Figura 6.3: Esquema cronológico da obtenção de um exemplo.

CompraVenda(t, l, p, d, c). Dessa forma, evitamos o "overlap" entre exemplos (ver figura 6.4).

Como conseqüência do distanciamento entre os exemplos, o tamanho do conjunto para treinamento e testes ficou reduzido a aproximadamente 1/6 do tamanho original da base disponível. Isso nos obrigou a adotar a estratégia de cross-validation, a qual será discutida ainda neste capítulo.

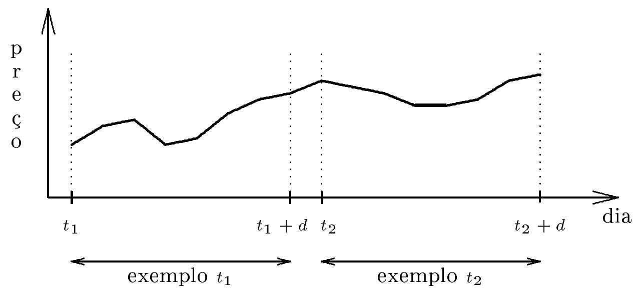


Figura 6.4: Dois exemplos consecutivos.

6.5 A Obtenção das Matrizes de Classificação

Após construído o conjunto de exemplos, este conjunto foi submetido aos algoritmos disponíveis, sendo os resultados aferidos e comparados. Nesta seção discutimos o critério para avaliação dos algoritmos.

Para a avaliação dos algoritmos, era necessário particionar o conjunto original de exemplos em dois subconjuntos: o primeiro seria o *conjunto de treinamento*, destinado à construção das árvores, e o segundo seria o *conjunto de testes*, destinado à aferição da qualidade das árvores geradas. Esse particionamento deveria ser realizado aleatoriamente, mantendo-se uma proporção aproximada de 70% dos exemplos no conjunto de treinamento e 30% dos exemplos no conjunto de testes [1].

[1] Na verdade, utilizamos uma forma de particionamento diferente, a ser descrita na próxima seção

Após obtidos os conjuntos de treinamento e de testes (denotados aqui por E_T e E_A, respectivamente), esses conjuntos foram submetidos aos algoritmos. Uma vez que cada algoritmo tem seu próprio conjunto de parâmetros P (por exemplo, o Cal5 possui como parâmetros o fator de decisão S e o nível de confiança $1 - \alpha$), o qual altera a precisão e tamanho da árvore gerada, foram realizadas diversas iterações com cada algoritmo, dentro de uma gama de valores permitidos para seus parâmetros, buscando com isso otimizar a precisão das árvores geradas.

Em cada iteração, uma árvore era gerada (através do conjunto E_T), à qual era submetido em seguida o conjunto E_A. Cada exemplo de E_A era classificado como SUCESSO, FRACASSO, ou simplesmente não era classificado, situação que ocorria quando o algoritmo nao tinha convicção suficiente para classificá-lo. Uma vez que jah conhecíamos a classificação real deste exemplo (pelo processo descrito em 6.4), podíamos dizer se o mesmo tinha sido ou não classificado corretamente pela árvore. Dessa forma, após a classificação dos exemplos de E_A pela árvore, obtínhamos uma *matriz de classificação*, que apresentava um resumo sobre a classificação dos exemplos. A figura 6.5 apresenta uma matriz de classificação típica.

	classe atribuída pela árvore	
classe real	FRACASSO	SUCESSO
FRACASSO	$m_{1,1}$	$m_{1,2}$
SUCESSO	$m_{2,1}$	$m_{2,2}$

Figura 6.5: Uma matriz de classificação típica.

Numa matriz de classificação, cada linha representa as classes verdadeiras dos exemplos, e cada coluna representa a classe prevista pela árvore de decisão. Por exemplo, o elemento $m_{1,2}$ da matriz descrita na figura representa o número de exemplos de classe FRACASSO que foram classificados pela árvore como SUCESSO. Os elementos $m_{1,1}$ e $m_{2,2}$ representam as quantidades de exemplos classificados corretamente.

Uma vez que cada matriz de classificação é determinada por um algoritmo Ψ, um conjunto de parâmetros P, um conjunto de treinamento E_T e um conjunto de testes E_A, denotaremos as matrizes de classificação por $M(\Psi, P, E_T, E_A)$.

6.6 Avaliação das Árvores Geradas

Para a avaliação das árvores geradas, foram definidas algumas funções de avaliação, cujos valores eram calculados a partir das matrizes de classificação. Algumas dessas funções de avaliação deveriam ser otimizadas, através da variação dos parâmetros dos programas.

Para fixarmos a notação, utilizaremos a definição que segue.

Definição 6.6.1 *Uma* função de avaliação *é uma função $\mathcal{A}$ definida sobre todas as tuplas (Ψ, P, E_T, E_A) cujo valor pode ser calculado a partir da matriz de classificação $M(\Psi, P, E_T, E_A)$, onde Ψ é um algoritmo de construção de regras de classificação, P é um conjunto de parâmetros válidos para Ψ, E_T é um conjunto de treinamento para Ψ, E_A é um conjunto de testes para Ψ.*

Em nosso trabalho, consideramos $\mathcal{A}(\Psi, P, E_T, E_A)$ como uma função objetivo a ser otimizada sobre uma família (finita) $\mathcal{P}(\Psi)$ de conjuntos de parâmetros válidos para Ψ. O conjunto $P \in \mathcal{P}(\Psi)$ que otimiza $\mathcal{A}$ será denotado por P^*.

Consideramos inicialmente três funções-objetivo:

- o erro global da árvore:

$$erro_{global}(\Psi, P, E_T, E_A) = \frac{m_{1,2} + m_{2,1}}{m_{1,1} + m_{1,2} + m_{2,1} + m_{2,2}}.$$

- a taxa de aplicações mal-sucedidas:

$$erro_{aplic}(\Psi, P, E_T, E_A) = \begin{cases} 1 & \text{caso } m_{1,2} = m_{2,2} = 0 \\ m_{1,2}/(m_{1,2} + m_{2,2}) & \text{caso contrário} \end{cases}$$

- a taxa de aproveitamento de oportunidades:

$$aprov(\Psi, P, E_T, E_A) = m_{2,2}/(m_{2,1} + m_{2,2}).$$

Note que apenas minimizar a função $erro_{aplic}$ nos levaria a construir árvores muito "conservadoras" que decidiriam por uma aplicação somente em condições muito favoráveis. Por outro lado, apenas maximizar a função *aprov* nos levaria a construir árvores muito "agressivas", que para não deixar de aproveitar nenhuma oportunidade decidiriam por aplicações de alto risco. Por essa razão, adotamos também a função de mérito abaixo:

$$merito(\Psi, P, E_T, E_A) = c * m_{2,2} - p * m_{1,2}.$$

onde

c	é o lucro mínimo definido para a estratégia CompraVenda(t, l, p, d, c)
p	é o prejuízo máximo permitido para a estratégia CompraVenda(t, l, p, d, c)

Essa função nos fornece uma noção da taxa de lucro (mínima) esperada sobre o conjunto de testes. Isso porque, segundo os parâmetros definidos para CompraVenda(t, l, p, d, c), cada aplicação mal-sucedida incorre num prejuízo máximo de $p\%$; cada aplicação bem-sucedida propicia um rendimento mínimo de $c\%$. Subtraídas as aplicações mal-sucedidas daquelas bem-sucedidas (cada qual ponderada pelo respectivo coeficiente), o que permanece é o lucro (mínimo) de $merito(\Psi, P, E_T, E_A)\%$.

A função *merito* é uma maneira intuitiva de combinar o erro em aplicações ($apl_e rr$) e a taxa de aproveitamento ($aprov$), pois para a maximização da mesma são procuradas matrizes de classificação com altos valores para $m_{2,2}$ (maior aproveitamento) e com baixos valores para $m_{1,2}$ (menores erros em aplicações).

As duas funções a serem otimizadas em nosso trabalho são $erro_{glob}$ e *merito*.

6.7 A Técnica de Cross-Validation

No início dos testes das árvores, nos deparamos com o fato do conjunto de exemplos obtido para cada papel ser demasiado pequeno para ser particionado em subconjuntos de treinamento e de testes. Tal partição acarretaria problemas tanto na obtenção das árvores quanto na obtenção da matriz de confusão. A solução adotada foi a técnica de *cross-validation*.

A técnica de *cross-validation* (comumente denominada também *V-fold cross-validation*) [Hjo94] consiste, basicamente, no procedimento descrito a seguir.

Divida aleatoriamente o conjunto original de exemplos E em V subconjuntos $E_1, E_2, \ldots, E_V$ de forma que

$$|E_v| = \left\lfloor \frac{|E|}{V} \right\rfloor, v = 1, 2, \ldots, V$$

ou seja, cada subconjunto deve ter exatamente o mesmo tamanho (para isso pode ser necessário desprezar alguns exemplos de E). Em nosso trabalho, adotamos o valor de V como 10.

Para cada $v \in \{1, \ldots, V\}$, separe o subconjunto E_v para testes e agrupe todos os demais subconjuntos para formar o conjunto de treinamento E_T^v, ou seja,

$$E_T^v = \bigcup_{i \neq v} E_i,$$

Calculando em seguida o valor da função $\mathcal{A}(\Psi, P, E_T^v, E_v)$.

Agora defina as *funções de erro global e de mérito cross-validation* ($erro_{glob}^{cv}$ e $merito^{cv}$) como

$$erro_{glob}^{cv}(\Psi, P, E^{cv}) = \frac{1}{V} \sum_{v=1}^{V} merito(\Psi, P, E_T^v, E_v)$$

$$merito^{cv}(\Psi, P, E^{cv}) \quad = \quad \frac{1}{V}\sum_{v=1}^{V} merito(\Psi, P, E_T^v, E_v)$$

onde

$$E^{cv} = \bigcup_{v=1}^{V} E_v.$$

Nosso objetivo passa a ser então encontrar, para cada algoritmo, os parâmetros P^* que otimizem $erro_{glob}^{cv}(\Psi, P, E^{cv})$ e $merito^{cv}(\Psi, P, E^{cv})$. A comparação efetiva dos programas se dará através da comparação entre os valores otimos de $erro_{glob}^{cv}$ e $merito^{cv}$.

Em nossos testes, também calculamos os desvios-padrões de $erro_{glob}(\Psi, P, E_T^v, E_v)$ e $merito(\Psi, P, E_T^v, E_v)$, com o objetivo de aferir quão sensíveis são as árvores com relação às instâncias:

$$\sigma = \sqrt{\frac{1}{n}\sum_{v=1}^{V} \left(\mathcal{A}(\Psi, P, E_T^v, E_v) - \mathcal{A}^{CV}(\Psi, P, E^{CV})\right)^2}.$$

As funções $erro_{aplic}^{cv}$e $aprov^{cv}$, bem como seus respectivos desvios-padrões, são obtidos de forma análoga.

Pode-se perceber que a grande vantagem da técnica de cross-validation é que praticamente todo o conjunto de exemplos gerados é utilizado, tanto para a construção quanto para o teste das árvores. Os resultados obtidos ficam menos suscetíveis à maneira como o conjunto original de exemplos foi particionado.

6.8 Resultados Obtidos

Estudamos nas seções anteriores a metodologia empregada para a aplicação dos algoritmos TDIDT para o mercado de ações. Foram apresentadas uma função de erro global $erro_{glob}^{cv}(\Psi, P, E_T^{cv})$ e uma função de mérito $merito^{cv}(\Psi, P, E_T^{cv})$ a serem otimizadas, através da variação dos parâmetros dos programas.

Definidos os critérios, realizamos os testes sobre os papéis Telebrás PN e Petrobrás PN, por sua alta liquidez e grande volume de negociação no mercado. Nesta seção, apresentaremos os resultados obtidos com a aplicação dos algoritmos Real, Cal5 e NewID sobre esses papéis.

A base de dados referente às ações Telebrás possui observações diárias, que vão de 19/06/89 a 23/02/96, num total de 1.736 registros. Eliminando-se os dias sem negociação do papel ("missing values"), obtemos um total de 1.609 registros. o conjunto de exemplos obtidos possui 1/6 desse tamanho, isto é, 267 registros. Destes, aproximadamente 46% são de classe SUCESSO.

A base de dados referente às ações Petrobrás possui observações de 02/01/86 a 23/02/96, num total de 2.638 registros. Eliminando-se os dias sem negociação do papel, obtemos 2.459 registros. O conjunto final de exemplos possui 410 registros, dentre os quais 40% são de classe SUCESSO.

Para a realização da técnica de cross-validation, dividimos cada conjunto de exemplos em 10 sub-conjuntos de mesmo tamanho.

6.8.1 Otimizando as funções de erro global e de mérito

Após realizados alguns testes preliminares sobre a sensibilidade dos algoritmos em relação a seus parâmetros, e considerando o intervalo natural em que estes parâmetros são definidos (ver cap 5), foram estabelecidas as seguintes malhas de discretização:

NewID: Para o fator de poda ϕ, foram utilizados todos os valores

$$\phi \in \{0\%, 2\%, 4\%, \ldots, 28\%, 30\%\}.$$

Cal5: Foram adotados todos os pares

$$(S, \alpha) \in \{0.05, 0.10, 0.15, \ldots, 0.90\} \times \{0.05, 0.10, 0.15, \ldots, 0.90\}.$$

Real: Foram adotados todos os pares

$$(r, crv) \in \{0.5, 1, 1.5, 2, \ldots, 4\} \times \{0.1, 0.15, 0.2, \ldots, 0.45\}.$$

Telebrás PN

Algoritmo	P^*	$erro^{cv}_{glob}$	$merito^{cv}$	$erro^{cv}_{aplic}$	$aprov^{cv}$
Real	$r = 3.5, crv = 0.4$	0.32 ± 0.08	3.8 ± 1.9	0.22 ± 0.12	0.44 ± 0.12
Cal5	$S = 0.3, \alpha = 0.1$	0.34 ± 0.07	3.3 ± 1.5	0.35 ± 0.09	0.63 ± 0.10
NewID	$\phi = 6\%$	0.35 ± 0.09	2.9 ± 2.3	$.25 \pm 0.09$	0.45 ± 0.22

Petrobrás PN

Algoritmo	P^*	$erro^{cv}_{glob}$	$merito^{cv}$	$erro^{cv}_{aplic}$	$aprov^{cv}$
Real	$r = 2, crv = 0.3$	0.30 ± 0.05	4.3 ± 2.3	0.24 ± 0.10	0.39 ± 0.16
Cal5	$S = 0.4, \alpha = 0.2$	0.31 ± 0.05	3.8 ± 2.4	0.23 ± 0.15	0.35 ± 0.16
NewID	$\phi = 8\%$	0.33 ± 0.05	2.9 ± 2.6	0.42 ± 0.05	0.28 ± 0.19

Tabela 6.1: Valores ótimos das funções $erro^{cv}_{glob}$ e $merito^{cv}$, com os respectivos valores das funções $erro^{cv}_{aplic}$ e $aprov^{cv}$.

Para cada conjunto de dados e cada algoritmo analisado, os parâmetros que minimizaram o erro global e a função de mérito foram os mesmos. Na tabela 6.1 são apresentados

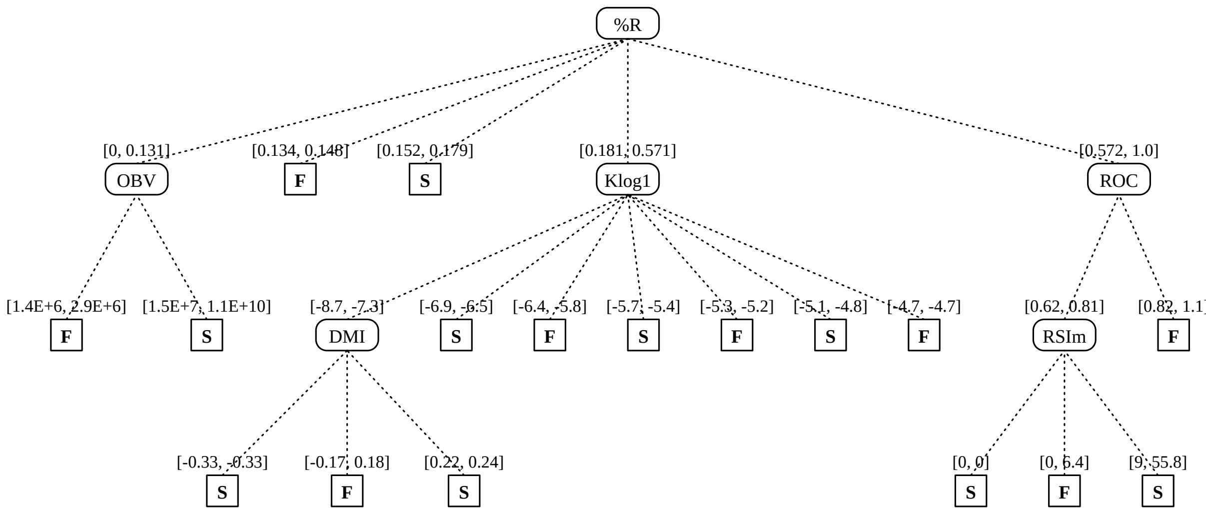


Figura 6.6: Árvore gerada pelo programa Real sobre o conjunto de treinamento Telebrás PN, usando os parâmetros $r = 3.5$ e $crv = 0.4$.

os parâmetros ótimos para as funções $erro_{glob}^{cv}$ e $merito^{cv}$ obtidos para cada programa, sobre as bases de dados de *Telebrás PN* e *Petrobrás PN*. São apresentados os valores ótimos dessas funções, com os respectivos valores de $erro_{aplic}^{cv}$e $aprov^{cv}$.

Na figura 6.6, vemos uma árvore gerada pelo programa Real sobre uma das partições do conjunto de treinamento da Telebrás PN. As folhas são rotuladas com as classes S (sucesso) e F (fracasso). Cada nó interno é rotulado por um indicador técnico (os indicadores não estão descritos, por estarem fora do escopo de nosso trabalho). Os indicadores mais próximos à raiz são considerados os mais relevantes, isto é, com maior poder preditivo. Cada ramo é rotulado com um intervalo do indicador correspondente. Por exemplo, o nó raíz, rotulado pelo indicador $\%R$, dá origem aos intervalos reais $[0, 0.131]$, $[0.134, 0.148], \ldots, [0.572, 1.0]$.

A árvore de decisão gerada pode ser empregada como uma ferramenta ágil no apoio à tomada de decisão: suponha que um analista de mercado resolva lançar mão da árvore de decisão da figura 6.6 para decidir se deve ou não aplicar a estratégia de operação pré-definida para as ações Telebrás PN, num certo dia t. Então basta comparar os valores dos indicadores no dia t com os intervalos constantes na árvore. Por exemplo, se $\%R = 0.3$ e $Klog1 = -5.5$, então a árvore sugere que a estratégia pode ser aplicada com grande possibilidade de ganho.

Embora os intervalos gerados para cada indicador não fossem contíguos, em nosso trabalho extendemos tais intervalos para que todo o espaço de atributo pudesse ser considerado, aumentando assim o aproveitamento de oportunidades.

6.8.2 Análise de sensibilidade

Considerando que o Real obteve os melhores resultados sobre as funções $erro^{cv}_{glob}$ e $merito^{cv}$, apresentamos a tabela 6.2, que descreve a variação nos valores das funções $erro^{cv}_{glob}$, $merito^{cv}$,$erro^{cv}_{aplic}$e $aprov^{cv}$nos pontos vizinhos ao ponto ótimo $P^* = (r^*, crv^*)$. Esta tabela nos dá uma noção do grau de sensibilidade do Real com relação a seus parâmetros.

Programa Real

Telebrás PN

r \ crv	0.35	$crv^* = 0.40$	0.45
3.0	$err_{glob} = 0.33 \pm 0.08$ $mer = 3.5 \pm 2.1$ $err_{apl} = 0.18 \pm 0.15$ $aprov = 0.38 \pm 0.15$	$err_{glob} = 0.33 \pm 0.13$ $mer = 3.5 \pm 3.0$ $err_{apl} = 0.26 \pm 0.18$ $aprov = 0.47 \pm 0.16$	$err_{glob} = 0.32 \pm 0.07$ $mer = 3.7 \pm 2.5$ $err_{apl} = 0.30 \pm 0.14$ $aprov = 0.51 \pm 0.15$
$r^* = 3.5$	$err_{glob} = 0.33 \pm 0.09$ $mer = 3.5 \pm 2.2$ $err_{apl} = 0.14 \pm 0.16$ $aprov = 0.34 \pm 0.13$	$err_{glob} = 0.32 \pm 0.08$ $mer = 3.8 \pm 1.9$ $err_{apl} = 0.22 \pm 0.12$ $aprov = 0.44 \pm 0.12$	$err_{glob} = 0.33 \pm 0.11$ $mer = 3.5 \pm 3.1$ $err_{apl} = 0.31 \pm 0.16$ $aprov = 0.51 \pm 0.15$
4.0	$err_{glob} = 0.34 \pm 0.10$ $mer = 3.2 \pm 2.6$ $err_{apl} = 0.27 \pm 0.29$ $aprov = 0.34 \pm 0.18$	$err_{glob} = 0.33 \pm 0.10$ $mer = 3.5 \pm 2.2$ $err_{apl} = 0.24 \pm 0.12$ $aprov = 0.41 \pm 0.15$	$err_{glob} = 0.34 \pm 0.12$ $mer = 3.2 \pm 3.0$ $err_{apl} = 0.27 \pm 0.20$ $aprov = 0.45 \pm 0.20$

Petrobrás PN

r \ crv	0.25	$crv^* = 0.3$	0.35
1.5	$err_{glob} = 0.34 \pm 0.07$ $mer = 2.7 \pm 2.9$ $err_{apl} = 0.30 \pm 0.30$ $aprov = 0.24 \pm 0.16$	$err_{glob} = 0.35 \pm 0.07$ $mer = 2.2 \pm 2.7$ $err_{apl} = 0.36 \pm 0.17$ $aprov = 0.35 \pm 0.16$	$err_{glob} = 0.36 \pm 0.09$ $mer = 1.8 \pm 3.4$ $err_{apl} = 0.40 \pm 0.18$ $aprov = 0.43 \pm 0.16$
$r^* = 2.0$	$err_{glob} = 0.33 \pm 0.06$ $mer = 3.0 \pm 2.7$ $err_{apl} = 0.35 \pm 0.35$ $aprov = 0.23 \pm 0.18$	$err_{glob} = 0.30 \pm 0.05$ $mer = 4.3 \pm 2.3$ $err_{apl} = 0.24 \pm 0.10$ $aprov = 0.39 \pm 0.16$	$err_{glob} = 0.33 \pm 0.05$ $mer = 3.1 \pm 2.5$ $err_{apl} = 0.36 \pm 0.09$ $aprov = 0.45 \pm 0.13$
2.5	$err_{glob} = 0.40 \pm 0.04$ $mer = 0.1 \pm 0.3$ $err_{apl} = 0.94 \pm 0.17$ $aprov = 0.02 \pm 0.06$	$err_{glob} = 0.32 \pm 0.08$ $mer = 3.2 \pm 3.2$ $err_{apl} = 0.36 \pm 0.35$ $aprov = 0.26 \pm 0.19$	$err_{glob} = 0.32 \pm 0.08$ $mer = 3.4 \pm 3.1$ $err_{apl} = 0.26 \pm 0.19$ $aprov = 0.32 \pm 0.14$

Tabela 6.2: Programa Real - Valores das funções nos pontos vizinhos a P^*.

Note que, para o papel *Telebrás PN*, os valores da funcão $merito^{cv}$nos pontos vizinhos ao ponto de ótimo P^* são bastante semelhantes, e apresentam uma pequena queda em relação a P^*. Isso mostra que a região de P^* é bastante "suave", e portanto a malha está bem definida e o ponto de ótimo bem localizado.

Para o papel *Petrobrás PN*, percebemos uma variação mais abrupta nos valores. Isso implica que poderíamos obter melhores resultados realizando testes sobre esses dados com uma malha mais fina, isto é, com intervalos menores entre os valores dos parâmetros. De qualquer forma, os resultados são surpreendentemente bons.

6.9 Considerações Finais

Neste capítulo, apresentamos uma metodologia para a adoção dos algoritmos TDIDT no apoio à tomada de decisões no mercado de capitais. Também apresentamos os resultados obtidos a partir de alguns testes realizados.

A partir de uma série histórica de preços de um papel, calcula-se para cada exemplo um conjunto de indicadores técnicos, os quais serão utilizados como atributos. A classificação do exemplo é binária, de acordo com uma certa estratégia de operação pré-definida: se a aplicação da estratégia sobre o exemplo resultar num lucro mínimo, o exemplo é rotulado como bem-sucedido; caso contrário, é rotulado como mal-sucedido.

Além do conjunto de treinamento, foram definidas algumas funções a serem otimizadas através dos parâmetros dos algoritmos. Para maior confiabilidade dos resultados, foi empregada a técnica de cross-validation.

Os resultados obtidos a partir dos testes realizados sobre os papéis Telebrás PN e Petrobrás PN foram em média bastante satisfatórios. Pudemos também perceber que as árvores geradas pelos programas Real e Cal5 possuem poucos níveis, o que as torna eficientes e de fácil interpretação - uma vez que a quantidade de indicadores selecionados por esses algoritmos foi relativamente pequena.

Conclusão

Em nosso trabalho, apresentamos os algoritmos da família TDIDT (Top-Down Induction of Decision Trees), os quais, a partir de um conjunto de exemplos (descritos através de seus atributos), constroem recursivamente as árvores de decisão. Foram apresentados diversos métodos de discretização e de escolha de atributos, regras de parada e métodos de poda.

Foram propostas técnicas para utilização desses algoritmos no suporte à escolha de estratégias de operação no mercado financeiro. Foram avaliados três algoritmos dessa família, sobre dois papéis de alta liquidez no mercado.

A partir dos resultados obtidos nos testes, pudemos concluir que:

- Os algoritmos TDIDT se mostraram muito úteis como ferramentas de suporte à decisão no mercado de capitais. Na verdade, foi surpreendente o desempenho obtido com os papéis analisados, onde foram obtidas taxas de aproveitamento e taxas de erro em aplicações da ordem de 40% e 20%, respectivamente.

- O escopo desse trabalho não incluiu a análise e interpretação dos indicadores utilizados e, portanto, também não incluiu a interpretação das árvores geradas. Todavia, contatos informais com analistas de mercado que utilizam estes indicadores mostraram que as árvores são de fácil interpretação. Pelo fato das árvores formarem regras de decisão baseadas exclusivamente nos indicadores, esses analistas se mostraram à vontade quanto à utilização das mesmas: utilizando sua experiência com tais indicadores, poderiam aceitar uma sugestão (quando esta fosse baseada em indicadores mais confiáveis) ou descartá-la (quando esta fosse baseada em indicadores de comportamento anômalo).

Através deste trabalho, são vislumbradas outras linhas para futuro desenvolvimento:

Mercado intra-day: O mercado intra-day (onde as estratégias são concretizadas no mesmo dia) fornece um campo muito vasto, dadas suas características:

- é mais agressivo e necessita de decisões rápidas; por essa razão, possui grande potencial de ganhos;

- a massa de dados disponíveis é muito maior, permitindo a formação de conjuntos de treinamento mais satisfatórios;
- necessita de respostas em "tempo real", exigindo portanto implementações mais eficientes.

Hipótese de eficiência de mercado: Segundo essa hipótese, os mercados são imprevisíveis. Gostaríamos de analisar com mais cuidado esta hipótese à luz dos resultados obtidos com o uso que fizemos das árvores de classificação.

Estratégias de operação: Poderá ser avaliado o desempenho das árvores frente a estratégias mais complexas, como por exemplo estratégias multi-período.

Referências Bibliográficas

[AB92] M. Anthony and N. Biggs. *Computational Learning Theory*. Cambridge University Press, 1992.

[AHU83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley Publishing Company, 1983.

[Ban90] S. P. Banks. *Signal Processing, Image Processing and Pattern Recognition*. Prentice Hall, 1990.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International, CA, 1984.

[Bos90] R. Boswell. *Manual for NewID version 4.1*. The Turing Institute, January 1990. TI/P2154/RAB/4/2.3.

[CKB87] B. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: A knowledge-elicitation tool for sophisticated users. In I. Bratko & N.Lavrac, editor, *Progress in Machine Learning*, pages 31–45. Sigma Press, England, 1987.

[CM88] R. W. Colby and A. Meyers, T. *The Encyclopedia of Technical Market Indicators*. Dow Jones-Irwin, Homewood, Illinois 60430, 1988.

[CN87] P. Clark and T. Nibblet. Induction in noisy domains. In I. Bratko & N.Lavrac, editor, *Progress in Machine Learning*, pages 11–30. Sigma Press, England, 1987.

[dBdV85] Comissão Nacional de Bolsas de Valores. Introdução ao mercado de ações, 1985.

[Fis35] R. A. Fisher. The logic of inductive inference. *Journal of the Royal Statistical Society*, 98:39–54, 1935.

[Hjo94] J. S. Urban Hjorth. *Computer Intensive Statistical Methods*, chapter 3. Chapman & Hall, 1994.

[Khi53] Khinchin. The entropy concept in probability theory. *Uspekhi Matematicheskikh Nauk*, VIII(3):3–20, 1953.

[LM71] R. J. Light and B. H. Margolin. An analysis of variance for categorical data. *Journal of the American Statistical Association*, 66(335):534–544, 1971.

[MG63] A. Mood and F. Graybill. *Introduction to the Theory of Statistics*. McGraw-Hill Book Company, Inc, second edition, 1963.

[Min87] J. Mingers. Expert systems - rule induction with statistical data. *J. Opl. Res. Soc*, 38(1):39–47, 1987.

[Min89] J. Mingers. An empirical comparison of pruning methods for decision-tree induction. *Machine Learning*, 4:227–243, 1989.

[MST94] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[MW94] W. Müller and F. Wysotzki. Automatic construction of decision trees for classification. *Annals of Operations Research*, 52:231–247, 1994.

[Nib87] T. Nibblet. Constructing decision trees in noisy domains. In I. Bratko & N.Lavrac, editor, *Progress in Machine Learning*, pages 67–78. Sigma Press, England, 1987.

[Nic94] M. C. Nicoletti. Ampliando os limites do aprendizado indutivo de máquina através das abordagens construtiva e relacional. Tese de Doutorado, Universidade de São Paulo - Instituto de Física de São Carlos, 1994.

[QCHL86] J.R. Quinlan, P. J. Compton, K. A. Horn, and L. Lazarus. Inductive knowledge acquisition: A case study. In *Proceedings of the 2nd Australian Conference on Applications of ES*, pages 183–203, Sydney, 1986.

[Qui79] J. R. Quinlan. Discovering rules by induction from large collection of examples. In D. Michie, editor, *Expert Systems in the Microelectronic Age*. Edinburgh University Press, 1979.

[SNL96] J.M. Stern, F. Nakano, and M.S. Lauretto. Real: Real attribute learning for strategic market operation. Technical Report RT-MAC-9606, Computer Science Department, University of São Paulo, August 1996.

[Utg89] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[Yat34] B. A. Yates. Contingency tables involving small numbers and the χ^2 test. *Journal of the Royal Statistical Society*, Suppl. 1:217–235, 1934.