Árvores de Decisão

Marcelo S. Lauretto

EACH-USP - marcelolauretto@usp.br
Novembro/2010

1 Definições

1.1 Aprendizado Supervisionado

Denotaremos por U o conjunto universo de objetos, ou seja, o conjunto de todos os objetos que o aprendiz pode encontrar.

Denotaremos por $C = \{c_1, c_2 \dots c_J\}$ um conjunto (finito) de classes (ou grupos), e assumimos que cada objeto do conjunto universo pertence a uma única classe c_j.

Assumimos que a representação dos objetos se dá através de um conjunto de características ou *atributos* previamente definido,
$A = a_1, a_2, \dots, a_m$. Ou seja, cada objeto é descrito por um *vetor de características* (ou *vetor de atributos)* $\mathbf{x} = [x_1, x_2, \dots, x_m]$, onde x_i é o valor do atributo a_i, $i = 1, \dots, m$.

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.

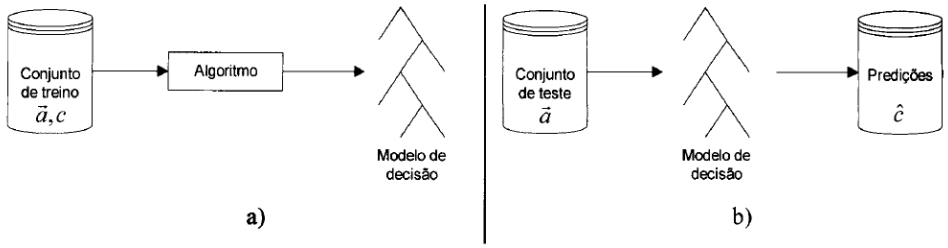
O conjunto universo U é usualmente denominado *espaço de atributos*, e é composto por todas os possíveis vetores de atributos $\mathbf{x}$ sobre o conjunto de atributos A.

Todo método de aprendizado supervisionado necessita de conjunto de exemplos E, tambem denominado *conjunto de treinamento*. O conjunto de treinamento E é um subconjunto de U composto por vetores de características $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \dots \mathbf{x}^{(n)}$ e suas respectivas classes $c^{(1)}, c^{(2)} \dots c^{(n)}$.

O problema de aprendizado supervisionado é encontrar, a partir do conjunto de treinamento E, uma função (*classificador*)

$$\widehat{f} : U \longrightarrow C$$

que associe a cada vetor de atributos $\mathbf{x}$ uma classe $\widehat{f}(\mathbf{x})$.



Fase de treinamento e fase de aplicação. Fonte: Pimenta (2004)

(Definição informal:) *Árvores de decisão* (ou *árvores de classificação*) são modelos de aprendizado supervisionado que representam regras de decisão baseadas nos valores dos atributos.

1.2 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 1.1 *Uma* árvore *pode ser definida recursivamente da seguinte maneira:*

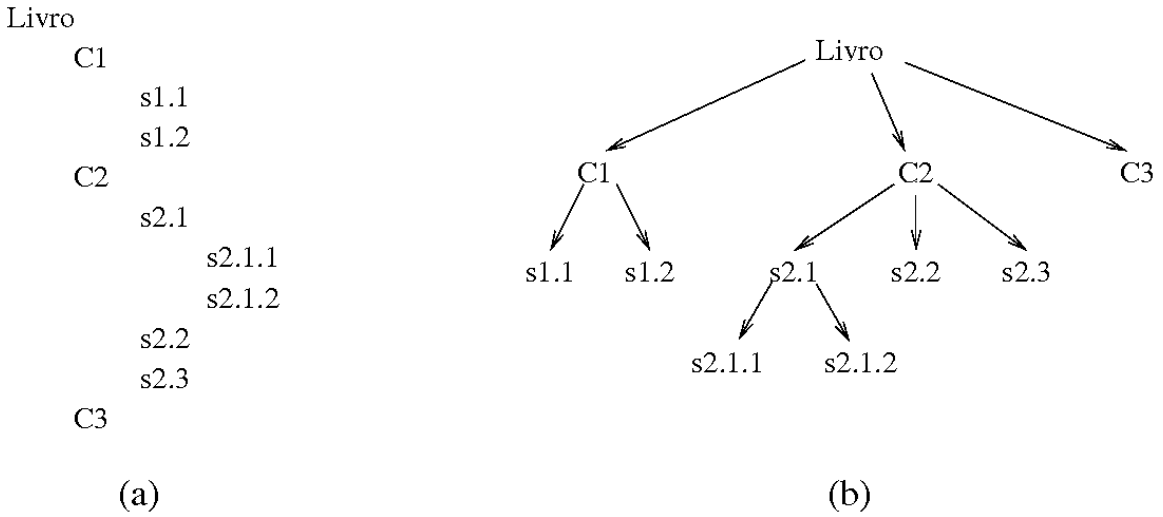
1. *Um único nó é uma árvore. Este nó é também a raíz da árvore.*
2. *Suponha que t seja um nó e $T_1, T_2, \ldots, T_k$ sejam árvores com raízes $t_1, t_2, \ldots, t_k$, respectivamente. Podemos construir uma nova árvore transformando t no* pai *dos nós $t_1, t_2, \ldots, t_k$. Nessa árvore, t será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

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.

Se existe um caminho do nó a ao nó b, então a é um *ancestral* de b e b é um *descendente* de a. Todo nó é ancestral e descendente de si mesmo.

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.

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 $\overline{T}$.



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

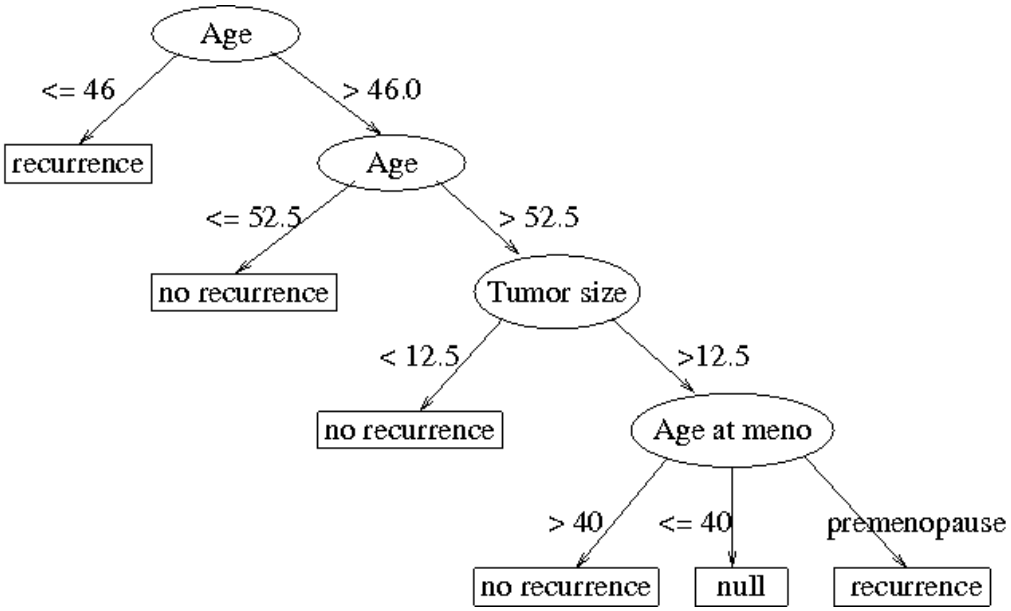
2 Árvores de Decisão

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.

Uma árvore de decisão possui a seguinte estrutura típica:

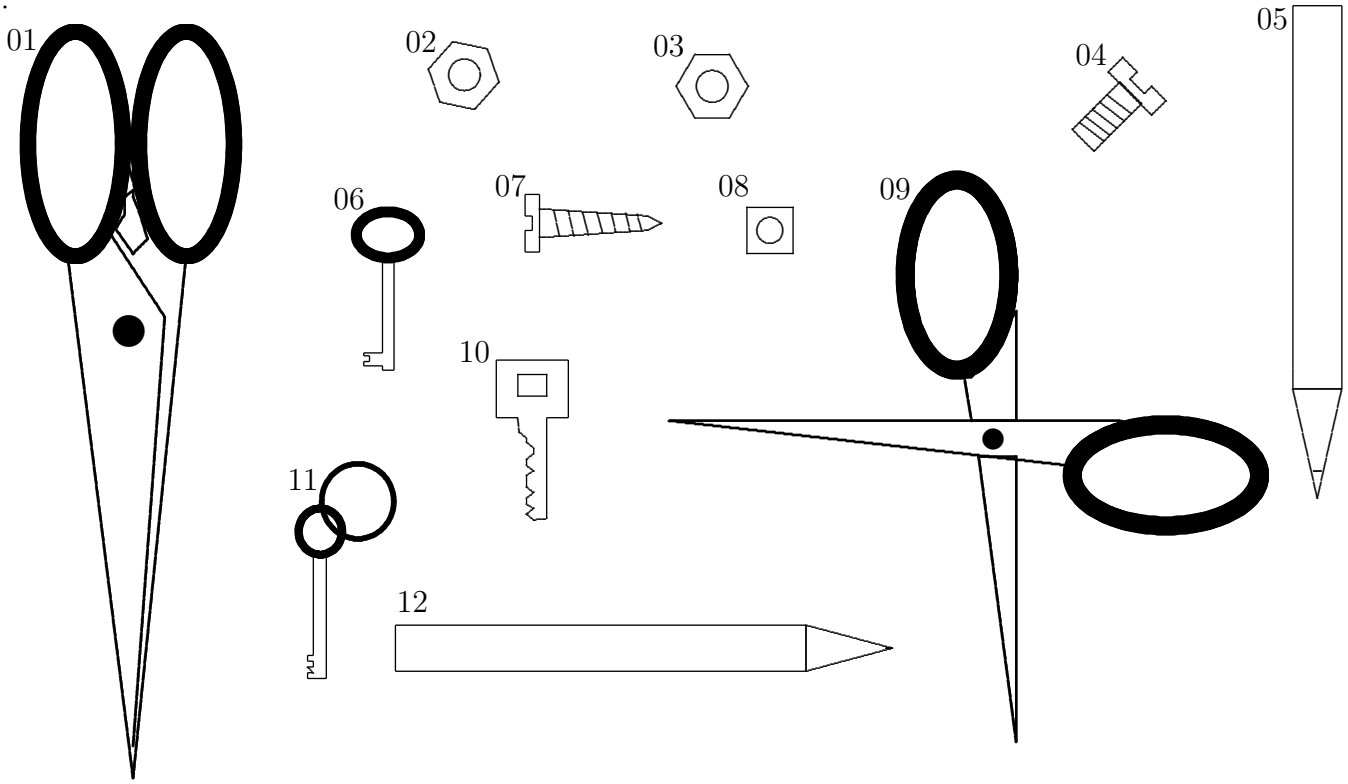
- Nós internos são rotulados com atributos;
- Folhas são rotuladas com classes;
- Ramos são rotulados com valores (atributos categóricos) ou com intervalos (atributos numéricos).



Exemplo: Predição de recorrência de câncer de mama

2.1 Exemplo de construção de uma árvore

Considere os objetos da figura abaixo. Esses objetos, 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 processamento de imagens.



(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)

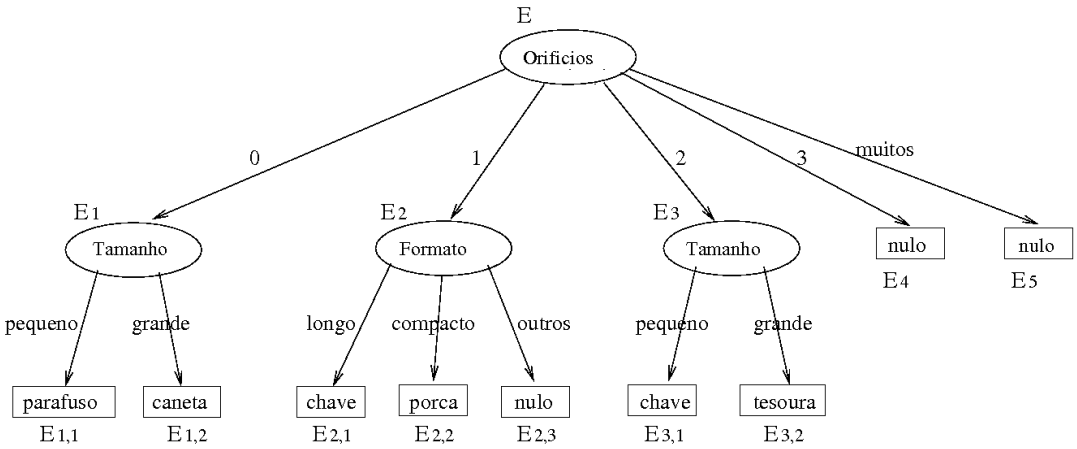
O programa pode extrair alguns valores de atributos de cada imagem, que no nosso caso são: o tamanho, o formato e o número de orifícios de um objeto. Os possíveis valores desses atributos são:

- *Tamanho*: *pequeno*, *grande*

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

Assim, cada objeto é representado através de seu respectivo vetor de atributo (item b da figura).

A figura abaixo mostra uma árvore de decisão induzida a partir dos exemplos acima.



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*. Em nosso caso, os subconjuntos de E são:

$$\begin{aligned} E_1 &= \{X \in E \text{ tq } X_{Orifícios} = 0\} \\ E_2 &= \{X \in E \text{ tq } X_{Orifícios} = 1\} \\ E_3 &= \{X \in E \text{ tq } X_{Orifícios} = 2\} \\ E_4 &= \{X \in E \text{ tq } X_{Orifícios} = 3\} \\ E_5 &= \{X \in E \text{ tq } X_{Orifícios} = muitos\} \end{aligned}$$

- 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.

Por exemplo, se tivermos um objeto com vetor de atributos

$$(Tamanho = grande, Formato = longo, Orifícios = 0),$$

a aplicação da árvore sobre esse objeto levará ao nó $E_{1,2}$ e o mesmo será, portanto, classificado como uma caneta.

3 Algoritmo de construção

Idéia geral do algoritmo:

- Expansão da árvore, através de sucessivas partições do conjunto de treinamento, até que a condição de parada seja satisfeita;
- Eliminação de algumas partes inferiores (*poda*) da árvore, através de reagrupamentos dos subconjuntos da partição.

Algoritmo

Dados um conjunto de aprendizado E
uma condição de parada $P(E)$
uma função de avaliação $score(E, atributo)$
uma função de classificação $classe(t)$
uma função de categorização $categ(E, atributo)$

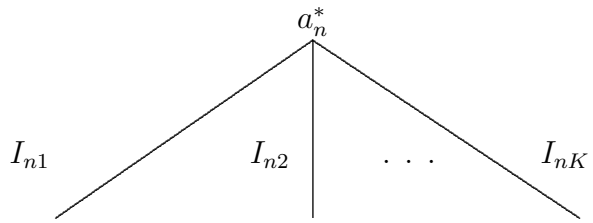
Se o conjunto E satisfaz a condição de parada $P(E)$,

então a árvore resultante é um único nó t rotulado 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. sejam $I_n1, I_{n2}, \ldots, I_{nK}$ os intervalos gerados segundo $categ(E, a_n^*)$;
   crie o nó e os ramos



3. particione E em K sub-conjuntos $E_1, E_2, \ldots, E_K$ segundo os intervalos
   (ou valores) de a_n na árvore de decisão

4. aplique o algoritmo recursivamente para cada um dos subconjuntos E_i

5. após a expansão da árvore ter terminado, ajuste seu tamanho,
   eliminando alguns ramos T_t se necessário.

Fim

Os principais elementos do algoritmo são:

- 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.

  Casos mais simples: 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.

- Regra $classe(t)$: função que atribui um rótulo de classe a um nó t.

  Critério mais simples: classificação por maioria.

- Função $score(a_i)$: é utilizada para tentar identificar o atributo mais relevante existente sobre E, isto é, o atributo com maior poder discriminante das classes.

- Método de categorização de atributos $categ(E, a)$: Atributos numéricos precisam ser "discretizados" em 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.

- Critério de poda: Após a expansão da árvore, pode ocorrer que os subconjuntos de E correspondentes aos nós folhas sejam muito pequenos e com grande efeito de *overfitting* (ajuste muito preciso aos dados de treinamento, porém com baixa precisão na classificação de novos exemplos).

  É necessário então eliminar-se alguns dos ramos inferiores da árvore, de forma a atenuar o efeito dos ruídos (informações inúteis para a classificação) e de forma a manter na árvore apenas regras com mais alto poder de discriminação das classes.

  Esse processo é denominado *pós-poda* (ou simplesmente *poda*) das árvores.

4 A regra de atribuição de classes

Denotaremos por $p(j|t)$ 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.

4.1 Critério de minimização do erro esperado

O critério mais simples de atribuição é adotar 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:

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

4.2 Critério de minimização do custo do erro de classificação

Breiman et al. (1984) sugere um critério de atribuição que minimiza o custo de erro de classificação no nó.

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

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

Se um objeto de classe desconhecida incidente sobre um nó t, for classificado com 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:

$$r(t) = \min_i \sum_j C(i, j)p(j|t). \tag{2}$$

Observação: O critério de escolha pela maior probabilidade estimada é um caso particular do critério de minimização de custos.

4.3 Rotulação dos nós pelas probabilidades das classes

Para classificação de um novo objeto, pode ser desejável que a árvore T forneça, ao invés de sua classe prevista j, as probabilidades estimadas desse objeto pertencer a cada uma das classes $1, 2, \ldots, J$.

Exemplo: diagnóstico médico.

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, vide Michie et al. (1994).

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

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

4.4 Custo total de erro de classificação na árvore

Uma vez apresentadas as regras de atribuição de classe e as respectivas estimativas de erro $r(t)$, é útil definirmos os custos estimados de erros de cada nó e o custo de erro da árvore T.

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.

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.

5 Escolha do Atributo Ótimo

Uma vez definidas as possíveis divisões para os nós da árvore (critérios de discretização), é necessário definir um critério de avaliação que escolha a melhor divisão para cada nó. De forma geral, 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 5.1 *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 $p_1, p_2, \ldots, p_J$.*

As restrições acima retratam as noções intuitivas de impureza: de fato, a impureza de um nó é:

- máxima quando todas as classes estão igualmente presentes no nó
- mínima quando o nó contém apenas uma classe.

Definição 5.2 *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 que construímos 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 Breiman et al. (1984), mostra que a aplicação sucessiva do critério acima gera uma árvore com impureza global mínima.

Teorema 5.3 *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

$$\begin{aligned} \Delta I(s,t) &= I(t) - I(t_v) - I(t_f) \\ &= p(t)imp(t) - p(t_v)imp(t_v) - p(t_f)imp(t_f) \end{aligned} \quad (3)$$

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

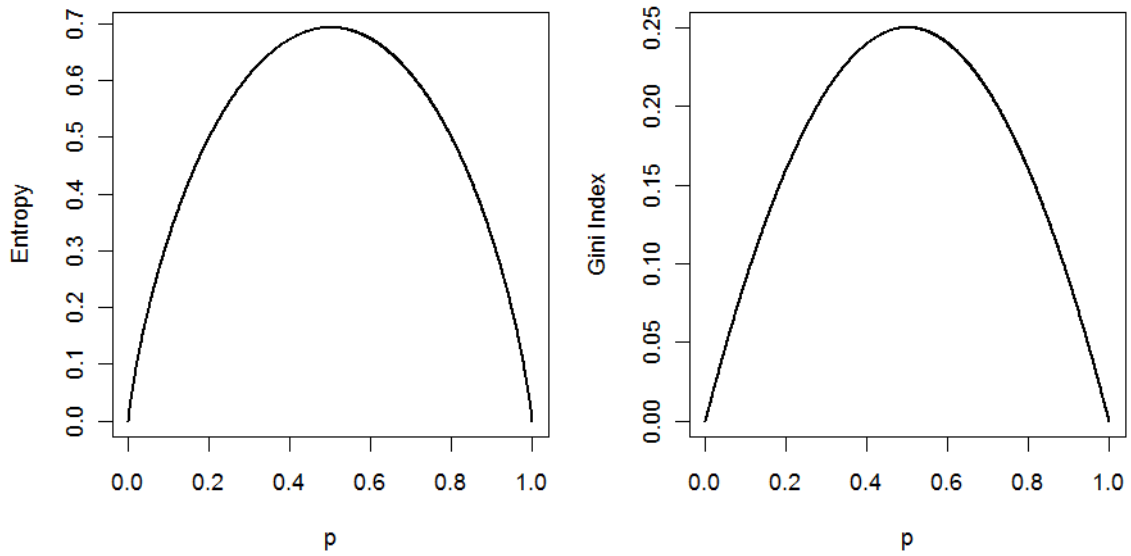
$$p_v = p(t_v)/p(t) \ , \ \ p_f = p(t_f)/p(t)$$

podemos reescrever a expressão 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 seqüência de passos de minimização da impureza global da árvore.

□



Forma da curva de Medida de Entropia e do Índice de Gini para 2 classes

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.

5.1 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. Aqui apresentamos a abordagem adotada por Khinchin (1953), 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 $p_i = 0$.

Khinchin (1953) 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.

Note que 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; resta mostrar 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,

$$\mathcal{E}(AB) = \mathcal{E}(A) + \mathcal{E}(B). \tag{4}$$

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

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

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

$$\mathcal{E}(AB) = \mathcal{E}(A) + \mathcal{E}_A(B). \tag{5}$$

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 5 permanece válida. É fácil também ver que a igualdade 5 é equivalente à igualdade 4 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

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

□

5.2 Índice Gini para custos unitários

O índice Gini, implementado no sistema CART (Breiman et al. (1984)), foi criado fundamentalmente como uma medida de variância para dados categóricos (Light & Margolin (1971)). Inicialmente será apresentada a versão do índice para custos unitários de erro.

O índice de diversidade Gini possui a forma

$$\mathcal{G}(p_1, p_2, \ldots, p_J) = \sum_{i \neq j} p_i p_j \tag{6}$$

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 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

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

Segunda interpretação: 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 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

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

Pode-se verificar que o índice de 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.

□

5.3 Índice Gini para custos variáveis

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

$$\mathcal{G}(p(1|t), p(2|t), \ldots, p(J|t)) = \sum_{i,j} C(i, j)p(i|t)p(j|t). \tag{7}$$

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.

Essa abordagem somente é adequada para casos em que a função de custo é aproximadamente simétrica, ou seja, quando $C(i, j)$ e $C(j, i)$ não são significativamente diferentes.

Apresentamos agora a segunda extensão para o índice Gini, que 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

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

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

$$C'(i, j)\pi'(j) = C(i, j)\pi(j). \tag{8}$$

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 8. 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 $\{\pi'(j)\}$.

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*

$$\pi'(j) = \frac{C(j)\pi(j)}{\sum_j C(j)\pi(j)}. \tag{9}$$

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 9;
3. Construa a árvore usando custos unitários e usando a distribuição $\{\pi'(j)\}$; utilize o índice Gini original.

6 A Poda de Árvores de Decisão

Todo algoritmo de construção de árvores de decisão tenta encontrar associações entre os atributos dos objetos e a classificação dos mesmos. A força dessas associações varia de acordo com o número de exemplos que suportam ou negam a relação observada. Algumas dessas associações refletirão características genuínas do domínio (a partir das quais se infere uma relação causal ou associativa, vide Clark & Nibblet (1987)). Outras serão encontradas ao acaso, devido a efeitos aleatórios ou à escolha particular dos exemplos presentes no sistema. Geralmente, encontrar as associações verdadeiras requer um número maior de exemplos para se evitar a interferência dos 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.

O problema então é encontrar o tamanho "correto" da árvore.

Duas abordagens:

- Pré-poda: consiste em estabelecer regras de parada sob certas condições, tais como:
    1. Se, para toda divisão s do nó t, $\Delta I(s,t) < \beta$, onde $\beta > 0$ é um parâmetro definido pelo usuário.

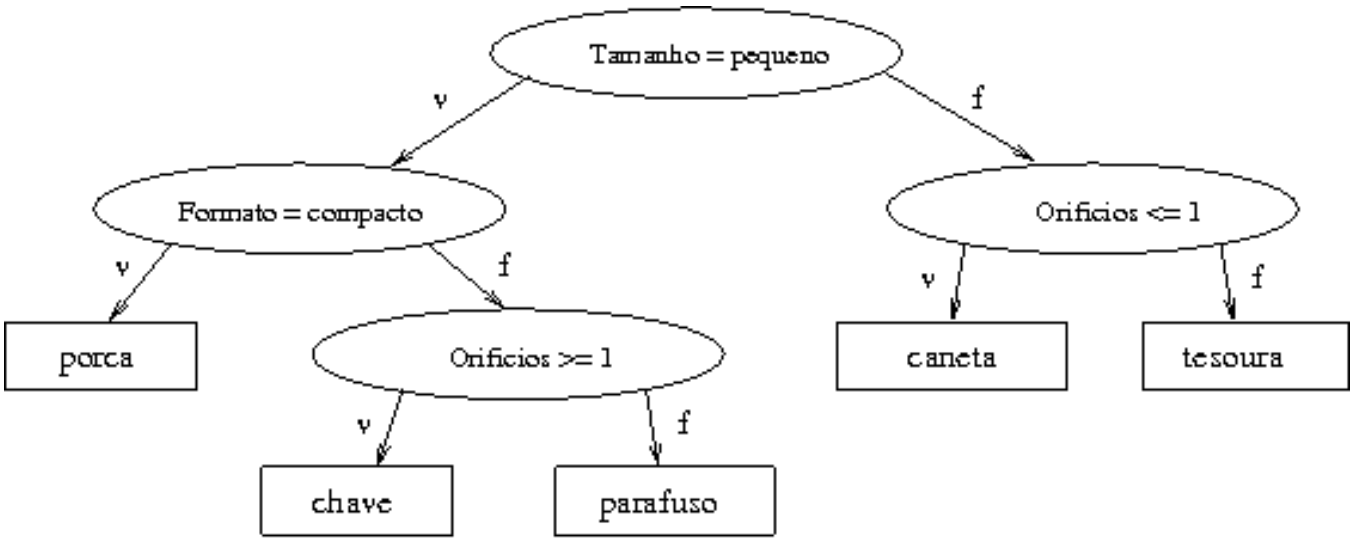
2. Se o número de exemplos que incidir sobre t for inferior a um parâmetro n.
3. Se a proporção dos exemplos incidentes no nó em relação ao número total de exemplos em E for inferior a um parâmetro p.
4. Se a estimativa de erro (ou o custo de erro) $r(t)$ naquele nó for menor do que um parâmetro r.

Alguns dos algoritmos que realizam a pré-poda são o REAL (Lauretto (1996)), o Cal5 (Müller & Wysotzki (1994)) , o C4 (Quinlan et al. (1986)) e o Assistant-86 (Cestnik et al. (1987)).

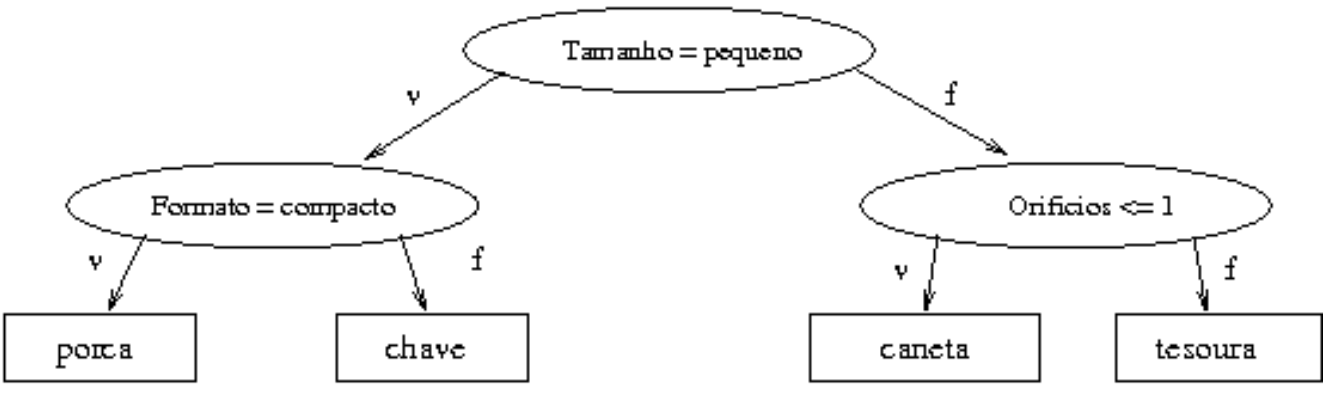
- Pós-poda: consistem em construir a árvore completa e avaliar a confiabilidade de cada uma de suas sub-árvores, podando os sub-ramos considerados não confiáveis.

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 $classe(t)$ (definida anteriormente). A árvore podada desta forma será denotada por $T - T_t$.

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 $T' \preceq T$.



(a)



(b)

Árvore original (esq) e árvore podada (dir)

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

6.1 Método de poda por custo-complexidade

Em linhas gerais, este método, proposto por Breiman et al. (1984) 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, basta ser suficientemente grande. Para isso, basta estabelecer um número mínimo de exemplos por folha N_{min} e adotá-lo como critério de parada.

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

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: Note que o tamanho 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 $R_\alpha(T)$).

Obtenção das Sub-árvores

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 6.1 *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

$$\begin{aligned} R_\alpha(t) &= R(t) + \alpha & (10)\\ R_\alpha(T_{k,t}) &= R(T_{k,t}) + \alpha|\overline{T_{k,t}}|. & (11) \end{aligned}$$

Note que $R_{\alpha(}T_{k,t}) < R_{\alpha k}(t)$ 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

$$R_\alpha(T_{k,t}) = R_\alpha(t) \tag{12}$$

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 10 e 11 fornecem o valor de α para o qual a igualdade 12 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 12 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}.

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 utilizar os mesmos exemplos que haviam sido empregados para a contrução da árvore, sob pena de tal estimativa de erro ser demasiadamente otimista (T_max será sempre favorecida na escolha).

Para calcularmos a estimativa de erros das árvores, utiliza-se 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.

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.

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

$$T_k^* = \arg \min_{1 \le k \le K} R^C(T_k).$$

7 Algoritmo Real

O Real - Real Valued Attribute Learning Algorithm - foi desenvolvido conjuntamente com o Prof. Júlio M. Stern, durante o desenvolvimento de minha dissertação, vide Lauretto (1996), Stern et al. (1998).

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

7.1 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. Consideramos que q possui uma distribuição

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

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.

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

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

7.2 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.

7.3 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.

7.4 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).

8 Random Forests

Uma floresta aleatória é um conjunto de árvores de classificação, cada qual construída a partir de um subconjunto aleatório do conjunto de treinamento. Esse conjunto de árvores resulta em um preditor agregado, que pode ser usado para a predição da classe de novos objetos através de um sistema de votação.

8.1 *Bagging Predictors*

Bootstrap é uma técnica estatística de propósito geral, amplamente usada para estimar as propriedades de um estimador, para construção de testes de hipóteses, vide Efron et al. (1993).

A idéia principal desta técnica consiste em: dada uma amostra de tamanho N, sorteiam-se amostras aleatórias com reposição de tamanho n (denominadas amostras bootstrap), e para cada amostra bootstraps estima(m)-se o(s) parâmetro(s) de interesse.

O termo "Bagging" foi usado pela primeira vez em Breiman (1996) como acrônimo do procedimento "**b**oostrap **agg**regat**ing**" . *Bagging Predictors* é um método para gerar múltiplas versões de um preditor e usando essas versões obtem-se um preditor agregado Breiman (1996). Os preditores são construídos utilizando amostras bootstraps do conjunto de treinamento, os quais são agregados para formar um preditor.

Cada reamostra é do mesmo tamanho do conjunto original de dados, ou seja, exemplos podem aparecer repetidamente enquanto outros podem não aparecer. Estes exemplos podem ser usados para formar estimativas das taxas de erros em árvores de decisão.

8.2 *Random Forest*

Random forests são uma combinação de árvores preditoras, onde cada árvore é construída usando uma amostra bootstrap dos dados originais. Esse procedimento faz com que, em média, dois terços dos exemplos originais sejam usados na construção da k-ésima árvore. Os exemplos não utilizados na construção são utilizados como conjunto de teste para avaliação do erro.

A construção de uma Random Forest pode ser resumida nos seguintes passos:

- O conjunto de dados é dividido aleatoriamente em um conjunto de teste e um conjunto de treinamento. Em casos reais o conjunto de testes representa aproximadamente 1/3 dos dados.
- A árvore de classificação é construída do conjunto de treinamento. A taxa de erros de classificação é estimada a partir do conjunto de teste.
- A árvore cresce a partir das amostras bootstrap tiradas do conjunto de treinamento. O conjunto original de treinamento é utilizado para selecionar a melhor aub-árvore podada. Este procedimento é repetido uma quantidade k de vezes resultando nas árvores de classificação (de 1 até k).
- Para cada elemento do conjunto de teste, compara-se a classe prevista com a classe verdadeira. A proporção das vezes que a classe estimada difere da verdadeira é a taxa de erro de classificação bagging.
- A divisão aleatória dos dados nos conjuntos teste e treinamento é repetida. O resultado final reportado consiste nas taxas médias de erro sobre as árvores, bem como os respectivos erros-padrão.

Cada árvore é construída como segue:

1. Se o número de exemplos no conjunto de treinamento é n, amostra-se com n exemplos ao acaso - mas com reposição, a partir dos dados originais. Essa amostra será o conjunto de treinamento para construir a árvore.
2. Se há m atributos, um número $m' << m$ é especificado tal que para cada nó, m' variáveis são selecionadas aleatoriamente, e a melhor divisão a partir dessas variáveis é usada para dividir o nó. O valor de m' é constante durante toda a execução.
3. Cada árvore cresce o máximo possível, sem poda.

Foi demonstrado por Breiman (2001), que a taxa de erro de uma floresta de árvores de decisão depende da robustez das árvores individuais na floresta (i.e sua taxa individual de erro) e da correlação entre suas classificações.

8.3 Medidas de Importância dos Atributos

Nos algoritmos de construção de árvores de classificação tradicionais, os atributos mais relevantes para classificação são naturalmente selecionados, graças aos procedimentos de pré e pós poda, vide Lauretto (1996). Dessa forma, a identificação dos atributos com maior poder preditivo é quase imediata. Nas Random Forests, por sua vez, a identificação não é imediata, devido ao grande número de árvores geradas e devido à ausência de procedimentos de poda na construção das árvores.

Por essa razão, adotam-se algumas métricas de avaliação da importância de cada atributo. Breiman (2001) sugere duas medidas, que são descritas a seguir, e que são utilizadas nos testes comparativos de nosso trabalho.

Apresentamos a seguir a notação e formulação das medidas de importância das variáveis. Assumimos que um conjunto de q árvores são geradas a partir das reamostras do conjunto de treinamento.

- O conjunto de árvores geradas é denotado por $K = \{1 \dots q\}$, e $k \in K$ denota a k-ésima árvore.
- O conjunto de atributos é denotado por $A = \{1 \dots m\}$, e $a \in A$ denota o a-ésimo atributo.
- O conjunto de classes é denotado por $C = \{1 \dots h\}$, e $c \in C$ denota a c-ésima classe.
- Cada árvore k é constituída por um conjunto $T_k = \{1 \dots I_k\}$ de nós; $i \in T_k$ denota o i-ésimo nó.
- $r(k, i) : T_k \rightarrow A$ denota o atributo que rotula o i-ésimo nó da k-ésima árvore. Para os nós terminais, define-se $r(k, i) = 0$.
- $T_k(a) \subseteq T_k$ denota o subconjunto dos nós de T_k rotulados pelo atributo a:

$$T_k(a) = \{i \in T_k | r(k, i) = a\}$$

- $n(k, i)$ é o número de exemplos do conjunto de treinamento que incidem sobre o nó i da árvore k.
- $n(k, i, c)$ é o número de exemplos de classe c do conjunto de treinamento que incidem sobre o nó i da árvore k.
- X denota o vetor de valores dos atributos 1...m de um exemplo qualquer:

$$X = (x_1, x_2, \dots, x_m)$$

Critério Baseado no Erro (IE)

Este índice é obtido permutando-se os valores do atributo a entre os exemplos do conjunto de teste e verificando-se o erro resultante. Quanto maior o erro, mais relevante é o atributo.

Seja c_k o número de exemplos classificados corretamente na árvore T_k e

$c'_{k,a}$ o número de exemplos classificados corretamente após a permutação da atributo a

Seja D, o decréscimo do erro do atributo a na árvore k

$$D_{k,a} = \frac{c'_{k,a} - c_k}{c_k}$$

Então, o Índice baseado no erro para o atributo a é:

$$IE_a = \frac{1}{q} \sum_{k \in K} D_{k,a}$$

Critério Baseado no Índice de Gini (IG)

Para escolher a divisão do nó, o índice de Gini é utilizado como segue. Suponha uma divisão candidata do nó, (a, s), que representa uma restrição $x_a \leq s$, onde s é um número real. Suponha que (a, s) divide o nó em dois nós filhos, i_v e i_f, com proporções p_v e p_f dos exemplos originais incidentes em i (i_v corresponde ao nó dos exemplos que obedecem à restrição, e i_f corresponde ao nó dos demais exemplos). A qualidade da divisão de (a, s) é medido pelo decréscimo no índice de Gini:

$$\Delta\mathcal{G}(k, i, a, s) = \mathcal{G}(k, i) - p_v\mathcal{G}(k, i_v) - p_f\mathcal{G}(k, i_f), onde$$

$$\mathcal{G}(k, i) = \sum_{c \in C c \neq d} \frac{n(k, i, c)}{n(k, i)} \times \frac{n(k, i, d)}{n(k, i)}$$

Note-se que cada fração $\frac{n(k,i,c)}{n(k,i)}$ representa a proporção de exemplos do nó i que pertencem a classe c.

Para expandir o nó i, escolhe-se a divisão (a^*, s^*) que maximiza $\Delta\mathcal{G}(k, i, a, s)$.

A medida de importância de cada atributo a em uma Floresta Aleatória pode ser dada pela soma dos índices de Gini de todos os nós rotulados por a:

$$IG_a = \frac{1}{q} \sum_{k \in K} \sum_{i \in T_k(a)} \Delta\mathcal{G}(k, i, a, s^*)$$

.

Referências

Breiman, L. (1996), 'Bagging predictors', *Machine Learning* **24**, 123–140.

Breiman, L. (2001), 'Random forests', *Machine Learning* **45**, 5–32.

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

Cestnik, B., Kononenko, I. & Bratko, I. (1987), Assistant 86: A knowledge-elicitation tool for sophisticated users, *in* I. B. . N.Lavrac, ed., 'Progress in Machine Learning', Sigma Press, England, pp. 31–45.

Clark, P. & Nibblet, T. (1987), Induction in noisy domains, *in* I. B. . N.Lavrac, ed., 'Progress in Machine Learning', Sigma Press, England, pp. 11–30.

Efron, B., Tibshirani, R. & Tibshirani, R. (1993), *An introduction to the bootstrap*, Chapman & Hall.

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

Lauretto, M. S. (1996), Árvores de classificação para escolha de estratégias de operação em mercados de capitais, Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo.

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

Michie, D., Spiegelhalter, D. J. & Taylor, C. C. (1994), *Machine Learning, Neural and Statistical Classification*, Ellis Horwood.

Müller, W. & Wysotzki, F. (1994), 'Automatic construction of decision trees for classification', *Annals of Operations Research* **52**, 231–247.

Pimenta, E. M. C. (2004), Abordagens para decomposição de problemas multi-classe: os códigos de correcção de erros de saída, Master's thesis, Faculdade de Ciências da Universidade do Porto, Porto.

Quinlan, J., Compton, P. J., Horn, K. A. & Lazarus, L. (1986), Inductive knowledge acquisition: A case study, *in* 'Proceedings of the 2nd Australian Conference on Applications of ES', Sydney, pp. 183–203.

Stern, J., Lauretto, M., Ribeiro, C. & Nakano, F. (1998), Real: real attribute learning algorithm, *in* 'Proceedings of the 4th International Conference on Information Systems, Analysis and Synthesis, SCI'98/ISAS'98', Orlando, pp. 315–321.