Universidade Federal do Ceará, Departamento de Estatística e Matemática Aplicada
Introdução à Mineração de Dados (CC0101), período 2019.2
Professor: Tibérius O. Bonates (`tb@ufc.br`).

Notas de Aula sobre o Classificador de Árvore de Decisão
(Não distribua este material. Ele é para uso exclusivo na disciplina identificada neste documento.)

1 Introdução

Árvore de Decisão (AD) é um algoritmo frequentement utilizado para tarefas de classificação, embora também possa ser utilizado para o caso de regressão. O classificador resultante pode ser representado por meio de uma estrutura de árvore, que representa regiões disjuntas do espaço onde os exemplos estão inseridos. Os algoritmos para construção de árvores diferem principalment em relação à maneira como essas regiões são determinadas e ao critério de término do algoritmo.

Conforme veremos, este tipo de classificador é apropriado para situações em que:

- desejamos obter uma hipótese que tenha uma forma disjuntiva (seja formada por uma disjunção de condições);
- o conjunto de treinamento contém erros (tanto na atribuição de classe quanto nos valores das características);
- o conjunto de treinamento contém exemplos incompletos, isto é, com dados faltantes.

2 Definições e convenções

Chamaremos de *espaço de características* (*feature space*) o espaço no qual os dados de treinamento e de teste estão imersos. No caso do conjunto de dados "Iris2D", utilizado anteriormente, o espaço de características pode ser considerado o $\mathbb{R}^2$. Já o conjunto "vote" está no espaço $\{0,1\}^{16}$. Se algumas características forem nominais ou tiverem seus valores limitados, o espaço de características pode ter uma estrutura mais complicada, como nos seguintes casos:

(a) Algumas características possuem intervalo limitado de valores. No caso do conjunto "Iris2D", por exemplo, existem limitações práticas sobre os valores da característica *petal length* (a primeira característica do conjunto), pois aquela característica tem seu valor dado em centrímentos e não existem, naquele conjunto, flores com pétalas maiores do que oito centrímentos. Assim, é possível termos um espaço de características do tipo $[0,8] \times \mathbb{R}$.

(b) Algumas características possuem valores numéricos, enquanto outras possuem valores nominais ou discretos. Como exemplo, podemos considerar uma característica chamada *temperatura*, com possíveis valores no conjunto {`baixa`, `média`, `alta`}. Desta forma, podemos ter um espaço de características dado por $\{0,1\} \times \{$`baixa`, `média`, `alta`$\} \times \mathbb{R}$, ou com estruturas mais complicadas.

Felizmente, as árvores de decisão lidam diretamente com dados deste tipo, sem exigir que as características sejam modificadas.

3 Funcionamento

A ideia essencial do algoritmo de AD consiste na divisão do espaço de características em duas ou mais regiões disjuntas, com o intuito de criar um classificador "simples" para cada região.

Na maioria dos algoritmos, esse classificador simples é realmente muito simples: o classificador constante, que classifica todas as observações como pertencentes a uma classe específica. Isso significa que o algoritmo associa a cada uma das regiões que ele cria uma das classes do conjunto de

treinamento.

Mais formalmente, dado o conjunto de treinamento $S = \{(x^i, y^i), i = 1, \ldots, m\}$, o algoritmo dividirá o espaço de características em m regiões, $R_1, \ldots, R_m$, com $R_i \cap R_j = \emptyset$, para todo $i \neq j$. Além disso, ele associará a cada região uma das classes existentes no conjunto de treinamento, por meio da seguinte função de classificação $c : \{R_1, \ldots, R_m\} \rightarrow \{y : \exists(x^i, y^i) \in S,\ y^i = y\}$.

3.1 Divisões

Os algoritmos de aprendizado de AD geralmente utilizam divisões bem simples do espaço de características. Para características numéricas, a divisão costuma ser paralela aos eixos desse espaço. Isso significa que serão criadas divisões do tipo $x_1 \leq 5$ versus $x_1 > 5$.

Para características nominais que possuam uma noção de ordem entre os valores possíveis, a ideia é semelhante: por exemplo, no caso da característica *temperatura*, poderíamos ter a divisão $x_2 \leq$ `média` versus $x_2 >$ `média`.

Caso a característica assuma valores nominais, mas sem uma noção de ordem entre eles, a divisão acontece por meio de teste de igualdade versus desigualdade: se x_3 é uma característica que assume apenas os valores `azul`, `verde` e `vermelho`, então uma possível divisão seria $x_3 =$ `verde` versus $x_3 \neq$ `verde`.

Formas alternativas de divisão

Outros tipos de divisão podem ser usados, e alguns deles podem ser obtidos apenas por meio do acréscimo de novas características artificiais aos dados, tal como características que envolvam o produto ou a combinação linear de algumas das características originais:

$$2x_1 + 3x_2 \leq 10 \quad \text{versus} \quad 2x_1 + 3x_2 > 10,$$
$$x_1 \cdot x_2 \leq 25 \quad \text{versus} \quad x_1 \cdot x_2 > 25.$$

Também é possível termos divisões que resultam em mais de duas partes, Por exemplo, se x_3 é uma característica que assume os valores `azul`, `verde` e `vermelho`, então poderíamos ter uma divisão feita sobre x_3 que particionaria os exemplos em 3 grupos: $x_3 =$ `azul`, $x_3 =$ `verde`, e $x_3 =$ `vermelho`.

A mesma partição obtida com um teste que resulte em 3 ou mais grupos pode ser obtida também por intermédio de uma sequência de testes que resultem em apenas dois grupos, cada. Por essa razão, muitas vezes as decrições dos algoritmos de AD se limitam ao caso de testes que resultem em dois grupos. As árvores deste tipo são chamadas de *binárias*.

As divisões de uma AD são feitas de forma sucessiva/cumulativa. Por simplicidade, vamos assumir que cada divisão irá particionar uma região do espaço de características em exatamente duas novas regiões. Esse processo dá origem a uma estrutura de árvore, em que cada nó (ou vértice) é associado a uma divisão e possui exatamente dois ramos partindo dele. O conjunto de todas as possíveis árvores é, portanto, o **espaço de hipóteses do classificador árvore de decisão**.

Assim, após a primeira divisão, temos o espaço de características dividido em exatamente duas regiões. Cada uma dessas regiões, por sua vez, pode ser dividida, e o processo pode se repetir muitas

vezes para cada (sub)região resultante. O conjunto de treinamento é particionado de acordo com cada nova divisão: os exemplos são divididos conforme pertençam a uma ou outra região do espaço de características.

Como o objetivo das divisões é criar regiões que possam ser facilmente classificáveis, a escolha do teste a ser usado para a divisão é baseada na informação de classe dos exemplos da amostra que pertencem àquela região. Também é com base na classe dos exemplos que pertencem a uma determinada região que o algoritmo decide não subdividir mais a região. A Figura 1 mostra um exemplo de árvore de decisão para o conjunto de dados da Tabela 1 que envolve a escolha entre jogar tênis (*Yes*) e não jogar tênis (*No*), com base nas características *Aparência Geral* (*Outlook*), *Vento* (*Wind*) e *Umidade* (*Humidity*).

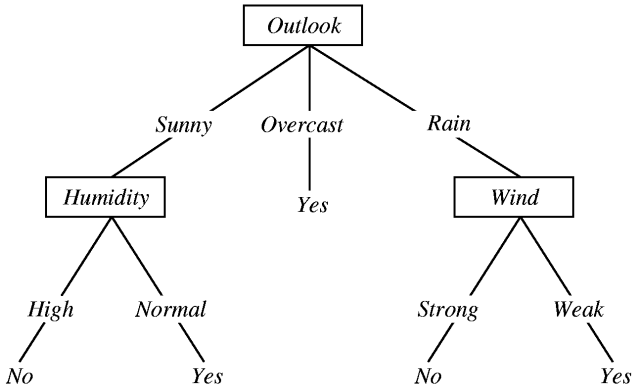

Figura 1: Exemplo de árvore de decisão. *Sunny*, *Overcast* e *Rain* são os valores possíveis da característica *Outlook*, enquanto *High* e *Normal* são aqueles da característica *Humidity*, e *Strong* e *Weak* são aqueles da característica *Wind*.

3.2 Classificação

Uma vez que a árvore tenha sido construída para o conjunto de treinamento, a classificação de um novo exemplo $\mathbf{x}$ pode ser feita seguindo-se os testes da árvore, a partir de sua raiz, até que se chegue a algumas de suas extremidades, que chamaremos de *folhas*. Sabemos que cada folha da árvore está associada a uma das regiões definidas pelo algoritmo. Seja $R(\mathbf{x})$ a região associada à folha em que chegamos ao seguir os testes da árvore utilizando o exemplo $\mathbf{x}$. A árvore classifica o exemplo $\mathbf{x}$ por meio do valor de $c(R(\mathbf{x}))$, dado pela função de classificação c, que associa a cada folha da árvore uma das classes do conjunto de treinamento.

Perceba que a árvore da Figura 1 é consistente com as observações da tabela 1, que constituem o conjunto de exemplo de treinamento. Perceba, também, que não é necessário utilizar todas características ao longo de cada caminho da raiz da árvore até uma folha. De fato, até mesmo o número de características utilizadas ao longo de cada um desses caminhos pode ser diferente, como no caso do caminho com *Outlook=Overcast*, que envolve apenas um teste, enquanto os caminhos para as demais folhas envolvem dois testes.

É interessante perceber que, uma vez construída a árvore, **os dados do conjunto de treinamento não são mais necessários para o propósito de classificar novos exemplos**. Isso difere do que acontecia no classificador de vizinhos mais próximos (K-NN).

Outlook	Humidity	Wind	Play Tennis
Sunny	High	Weak	No
Sunny	High	Strong	No
Overcast	High	Weak	Yes
Rain	High	Weak	Yes
Rain	Normal	Weak	Yes
Rain	Normal	Strong	No
Overcast	Normal	Strong	Yes
Sunny	Normal	Weak	Yes
Sunny	Normal	Strong	Yes
Overcast	High	Strong	Yes
Overcast	Normal	Weak	Yes
Rain	High	Strong	No

Tabela 1: Conjunto de dados "Play Tennis".

4 Um algoritmo genérico para construir uma AD

O algoritmo 1 mostra um pseudocódigo para construção de uma AD. O algoritmo não especifica certos detalhes de implementação, como a escolha da melhor divisão e o critério para interromper a divisão no nó atual. Estes detalhes serão discutidos adiante. Algoritmos clássicos como ID3, C4.5 e CART encaixam-se nesse esquema genérico. De início, o algoritmo 1 é chamado como `CriarArvore(`S`)`, isto é, ele recebe como entrada toda a amostra. Em momentos subsequentes, o algoritmo é chamado recursivamente, recebendo como entrada subconjuntos de exemplos da amostra.

Algorithm 1: `CriarArvore(`D`)`: – Pseudocódigo genérico para construção de uma AD.

ENTRADA: conjunto de exemplos $D = \{(\mathbf{x}^i, y^i) : i = 1, \ldots, t\}$
SAÍDA : Árvore de decisão A.

1 Inicializar árvore A com apenas um nó
2 **if** D contém quantidade suficiente de observações de classes distintas **then**
3 Escolher "melhor" teste para divisão de D, resultando na partição

$$D = D_1 \cup \cdots \cup D_k,\ D_i \cap D_j = \emptyset,\ \forall i, j = 1, \ldots, k,\ i \neq j,\ k \geq 2$$

4 **for** $i = 1, \ldots, k$ **do**
5 $A_i \leftarrow$ `CriarArvore(`D_i`)`
6 Fazer A_i ser descendente de A (subárvore de A)
7 **else**
8 Etiquetar A como nó terminal (folha), associado à classe majoritária em D
9 Retornar A

4.1 Quando parar?

No passo 2 do Algoritmo 1, verificamos se o conjunto D é suficientemente heterogêneo em relação às classes das observações que o compõem. Esta verificação geralmente consiste em detectar se as observações do conjunto D são unicamente, ou predominantemente, de uma mesma classe. Em caso positivo, não há necessidade de dividir o conjunto e o algoritmo pára após atribuir uma classificação

constante a este nó terminal. A classificação atribuída é aquela da classe mais frequente dentre as observações em D (passo 8). Em caso negativo, o algoritmo realiza uma partição de D em dois ou mais subconjuntos, que irão constituir subárvores da árvore A.

Com esse tipo de critério de parada, é possível mostrar que o algoritmo termina em um número finito de passos, desde que o conjunto de treinamento S não possua observações idênticas com classes distintas. Essa condição é importante porque, dados dois exemplos distintos, sempre existe um teste que é satisfeito por um dos exemplos e não é satisfeito pelo outro. Isso implica na existência de pelo menos um teste no passo 3 para qualquer subconjunto da amostra original S:

Proposição 1. *Dado um conjunto de exemplos $D = \{(\mathbf{x}^i, y^i) : i = 1, \ldots, m\}$, se $y^i = y^j$ para todos os pares $i, j \in \{1, \ldots, m\}$ com $i \neq j$ e $\mathbf{x}^i = \mathbf{x}^j$,* `CriarArvore(`$D$`)` *termina em um número finito de passos.*

Note que esta afirmação independe de como a divisão é selecionada no passo 3.

4.2 Como detectar divisões?

O procedimento para elencar as possíveis divisões é realizado separadamente para cada característica. Se a característica for nominal, o procedimento costuma ser exaustivo: se $v_1, \ldots, v_p$ são os valores que a característica X_j assume em D, geramos todos os testes que envolvem X_j e seus valores em S. Estes testes costumam ser de dois tipos: (i) binário: "$X_j = v_k$ versus $X_j \neq v_k$", para $1 \leq k \leq p$, ou (ii) p-ário (*multiway*): $X_j = v_1$ versus $X_j = v_2$ versus $\cdots$ versus $X_j = v_p$.

No caso de uma característica X_j numérica (ou nominal, mas com uma ordem total sobre seu domínio), as observações em S podem ser ordenadas em função do valor de X_j. Sejam $v_1 < v_2 < \cdots < v_p$ os valores que X_j assume em D. Para cada par de valores consecutivos $v_i < v_{i+1}$, verificamos se existem observações $\mathbf{x}^r$ e $\mathbf{x}^s$ em D tais que:

(i) $X_j = v_i$ em $\mathbf{x}^r$, isto é, $\mathbf{x}^r_j = v_i$;

(ii) $X_j = v_{i+1}$ em $\mathbf{x}^s$, isto é, $\mathbf{x}^s_j = v_{i+1}$;

(iii) $y^r \neq y^s$.

Em outras palavras, os valores v_i e v_{i+1} da característica X_j podem acontecer associados a classes distintas. Em tal situação, o valor v_i é considerado como parte de um teste; especificamente, o teste "$X_j \leq v_i$ versus $X_j > v_i$".

Como exemplo, considere o conjunto de dados da Tabela 2, na qual a primeira coluna fornece o índice de cada observação, as 3 colunas seguintes fornecem os valores das 3 caraterísticas, seguidos da classe da observação. Para enumerar os valores candidatos a divisão da característica X_3, por exemplo, obtemos primeiramente a lista ordenada de valores que X_3 assume em S: 1.0, 2.0, 2.8, 3.7, 3.8, 5.2. O valor 1.0 não é um candidato porque, embora seja possível satisfazer as condições (i) e (ii) acima, não é possível satisfazer simultaneamente a condição (iii). Mais especificamente, o valor 1.0 de X_3 acontece apenas na observação 5 do conjunto S, enquanto que o valor 2.0 em X_3 acontece apenas na observação 6, satisfazendo, portanto, as condições (i) e (ii). No entanto, ambas as observações são da classe 2, violando a condição (iii). Portanto, não é possível encontrar o par de observações $(\mathbf{x}^r, \mathbf{x}^s)$ requerido.

O valor 2.0, por sua vez, é candidato a participar de uma divisão, já que efetivamente distingue entre a observação 6 e a observação 1. Em outras palavras, a observação 6 satisfaz a condição (i), a

observação 1 satisfaz a condição (ii), e estas duas observações pertencem a classes diferentes (satisfazendo a condição (iii)). A Tabela 3 mostra todos os valores candidatos para divisão no início do processo de construção da árvore de decisão.

Perceba que alguns destes valores poderiam deixar de ser candidatos, caso estivéssemos utilizando um subconjunto próprio de S. De fato, é isso que ocorre quando uma árvore de decisão está sendo construída: os testes levados em consideração para dividir um dado nó da árvore são apenas aqueles que são candidatos válidos para o subconjunto particular de S associado àquele nó.

Obs.	X_1	X_2	X_3	classe
1	3.5	3.8	2.8	1
2	2.6	1.6	5.2	1
3	1.0	2.3	3.7	1
4	3.5	1.6	3.8	2
5	2.3	2.1	1.0	2
6	4.2	1.8	2.0	2

Tabela 2: Conjunto de treinamento S, com três características numéricas e duas classes.

Características	X_1	X_2	X_3
Valores candidatos	1.0	1.6	2.0
	2.3	2.1	3.7
	2.6		3.8
	3.5		

Tabela 3: Valores candidatos para divisão de cada característica dos dados da Tabela 2.

4.3 Como escolher a divisão a ser realizada?

No passo 3 do Algoritmo 1, uma "melhor" partição de D é detectada e as partes $D_1, \ldots, D_k$ correspondentes à partição são construídas. Este passo pode consistir apenas na detecção de uma característica apenas, caso ela seja nominal, mas pode também consistir na detecção de uma característica e de um valor de limiar para a divisão, caso a característica seja numérica. De qualquer forma, o importante para este passo é que nenhuma outra divisão seja mais "vantajosa" para o conjunto D do que a divisão escolhida, de acordo com o critério utilizado.

É natural que o critério utilizado para selecionar uma divisão vantajosa represente alguma noção de "pureza" ou "homogeneidade" das partes resultantes. Em um cenário ideal, gostaríamos de escolher uma divisão em que nenhuma das partes resultantes, $D_1, \ldots, D_k$, contivesse observações de duas classes distintas. No caso geral, não podemos assumir que existe uma divisão deste tipo. Na ausência de uma divisão assim, alguma forma de comparar divisões deve ser utilizada.

Um critério simples seria calcular o *erro de classificação*, que é o percentual das observações em cada D_i que não pretencem à classe c_i^* mais frequente em D_i:

$$EC(D_i) = \frac{1}{|D_i|} \sum_{(\mathbf{x}^j, y^j) \in D_i} \mathbf{I}(y^j, c_i^*),$$

onde $\mathbf{I}(a, b) = 1$, se $a = b$, e $\mathbf{I}(a, b) = 0$, caso contrário. Aqui, o erro total da partição seria dado pela soma dos erros das partes, $EC(D_1, \ldots, D_k) = \sum_{i=1}^{k} EC(D_i)$.

Um critério muito utilizado, chamado de *Ganho de Informação*, baseia-se na noção de entropia de Shannon, da teoria da informação. Se D_i possui t classes, $c_1, \ldots, c_t$, e os percentuais de observações de cada uma das classes são $p_1, \ldots, p_t$, respectivamente, então a entropia de D_i é dada por

$$H(D_i) = -\sum_{k=1}^{t} p_k \log_2(p_k),$$

com a convenção de que $0 \log_2(0) = 0$. Note que, se tivermos duas classes com $p_1 = p_2 = \frac{1}{2}$, então temos entropia igual a 1, enquanto que, se tivermos $p_1 = 1$ e $p_2 = 0$, a entropia é igual a zero. Com t classes, o valor máximo da entropia é $\log_2(t)$, que é atingido quando temos todas as classes com igual percentual, $\frac{1}{t}$, de participação em S_i.

Com base na medida de entropia, podemos calcular o *ganho* resultante de uma divisão de D que resulta nas partes $D_1, \ldots, D_k$ como

$$G(D, (D_1, \ldots, D_K)) = H(D) - \sum_{i=1}^{k} \frac{|D_i|}{|D|} H(D_i).$$

Desejamos, portanto, encontrar uma divisão que maximize o ganho, o que é equivalente a minimizar a soma ponderada das entropias das partes – ou seja, desejamos que a partição resultante tenha baixa entropia nas partes. No cenário ideal, teríamos entropia nula em cada uma das partes e o ganho máximo teórico.

Como exemplo, considere o conjunto de dados "Play Tennis" da Tabela 1. Calculando a entropia para o conjunto inteiro de exemplos, temos

$$H(S) = -\frac{4}{12}\log_2\left(\frac{4}{12}\right) - \frac{8}{12}\log_2\left(\frac{8}{12}\right) \approx 0.9183,$$

uma vez que temos apenas duas classes, com percentuais $\frac{4}{12}$ e $\frac{8}{12}$.

Outro critério muito utilizado no lugar da entropia é o índice Gini. Dada uma parte D_i de D, o critério Gini para D_i corresponde à probabilidade de atribuirmos uma classe errada a um exemplo selecionado aleatoriamente a partir de D_i, caso a classe a ser atribuída seja selecionada aleatoriamente, de acordo a distribuição de classes presentes em D_i. Se denotarmos por p_c a fração de exemplos de D_i que pretencem à classe c e por $C(D_i)$ o conjunto das classes presentes em D_i, podemos escrever:

$$G(D_i) = \sum_{c \in C(D_i)} p_c \sum_{\substack{d \in C(D_i) \\ d \neq c}} p_d = \sum_{c \in C(D_i)} p_c(1 - p_c) = \sum_{c \in C(D_i)} p_c - \sum_{c \in C(D_i)} p_c^2 = 1 - \sum_{c \in C(D_i)} p_c^2.$$

Na segunda e quarta igualdades acima, utilizamos o fato de que $\sum_{c \in C(D_i)} p_c = 1$. Uma vantagem do índice de Gini em relação à entropia é a velocidade de cálculo, já que ele não exige que sejam calculados logaritmos.

5 Outros Assuntos

5.1 Evitando Super-Aprendizado

Seja $\mathcal{H}$ o conjunto de hipóteses (classificadores) nos quais estamos interessados (em nosso caso, o conjunto de todas as árvores de decisão que podem ser construídas para um dado conjunto de treinamento S)[1]. Na fase de aprendizado, estamos interessados em selecionar um classificador (árvore) que se ajuste ao conjunto de treinamento S, isto é, que cometa poucos erros (ou nenhum erro) na classificação de exemplos do conjunto S.

Sejam $h \in \mathcal{H}$ o classificador escolhido ao final da fase de treinamento, e T um conjunto de exemplos de teste. Definimos $\text{erro}_S(h)$ como o número de observações em S que h classifica incorretamente e $\text{erro}_T(h)$, de forma similar. Dizemos que houve super-aprendizado (*overfitting*) durante a etapa de aprendizado que selecionou h se existir $h' \in \mathcal{H}$, satisfazendo $\text{erro}_S(h') \geq \text{erro}_S(h)$, mas tal que $\text{erro}_T(h') < \text{erro}_T(h)$. Em outras palavras, h' erra tanto ou mais do que h no conjunto de treinamento, mas comete menos erros do que h no conjunto de teste. Em geral, não conhecemos o conjunto de teste T que iremos encontrar ao usar o classificador na prática. Mas, há formas de estimar empiricamente se h tende a cometer mais erros na classificação de observações desconhecidas do que classificadores alternativos cujos erros em S são maiores ou iguais a $\text{erro}_S(h)$.

No caso de ADs, é possível ter duas ou mais árvores distintas que classificam incorretamente o mesmo número (e até o mesmo conjunto) de observações do conjunto de treinamento. Estas mesmas árvores podem classificar de maneira diferente observações do conjunto de teste. Logo, é possível ocorrer super-aprendizado no contexto de ADs. Em particular, desde que não existam duas observações $(\mathbf{x}^1, y^1)$, $(\mathbf{x}^2, y^2)$ no conjunto de treinamento S, com $\mathbf{x}^1 = \mathbf{x}^2$ e $y^1 \neq y^2$, é sempre possível construir uma árvore h com $\text{erro}_S(h) = 0$, isto é, uma árvore que classifica todas as observações de S corretamente. Para isso, é preciso dividir suficientemente os dados, mesmo que isso exija construir subregiões que contenham poucos exemplos de treinamento. Uma árvore com esta propriedade é candidata a sofrer do problema de super-aprendizado, pois o ajuste perfeito à amostra que compõe o conjunto de treinamento pode resultar em baixa precisão na classificação de exemplos provenientes da população geral – isto é, a árvore teria pouca capacidade de generalizar o conhecimento extraído de S.

Árvores que possuem uma ou mais subregiões associadas a poucas observações do conjunto de treinamento costumam ser resultado de super-aprendizado. Algumas razões simples para isso são:

(i) Subregiões deste tipo podem ser criadas como resultado de ruído/imprecisão aleatório(a) nos dados de treinamento. Esta imprecisão pode acarretar a ocorrência de uma observação que esteja muito próxima a observações de outra classe, obrigando a criação de uma sequência de divisões específicas para classificar corretamente aquela observação. Por ser causada por imprecisão aleatória na coleta dos dados, é de se esperar que a subregião resultante tenha baixa taxa de acerto na classificação de observações do conjunto de teste.

(ii) Exemplos do conjunto de treinamento que estejam próximos à superfície de separação entre duas ou mais classes naturalmente tendem a induzir uma divisão fina dos dados. Infelizmente, a superfície de separação pode ser muito complexa, e o conjunto de treinamento pode não conter exemplos suficientes para possibilitar a detecção fiel desta superfície. Isso causaria subregiões associadas a uma determinada classe como uma consequência da amostra que compõe o conjunto de treinamento, e não de como a superfície de separação realmente é. Na presença

[1] Se o conjunto de treinamento contiver características numéricas, temos um conjunto infinito de árvores de decisão.

de mais dados, estas subregiões em questão poderiam ter sido associadas a outras classes, ou poderiam ter sido subdivididas, ou talvez tais subregiões nem mesmo existissem na partição induzida pelo processo de criação da árvore.

(iii) É possível que as características disponíveis sejam insuficientes para detectar a superfície de separação dos dados por meio de uma árvore de decisão. Neste caso, é de se esperar que a árvore especifique subregiões pequenas para conseguir classificar corretamente os dados de treinamento.

Em cada um dos casos descritos acima, é razoável supor que uma divisão **menos detalhada** dos dados — isto é, uma divisão que **não** exigisse subregiões contendo somente observações de uma mesma classe — acarretasse **menos** erros na classificação de exemplos do conjunto de teste, mesmo que esse efeito fosse acompanhado de um aumento do número de erros na classificação de exemplos do conjunto de treinamento. Sendo este o caso, teríamos evitado o problema de super-aprendizado.

No contexto de árvores de decisão, existem duas ideias principais para atingir esse objetivo: a *pré-poda*, que consiste em evitar que divisões muito específicas aconteçam, é realizada por meio da interrupção prematura do processo de divisão quando se constrói a árvore; a *pós-poda*, que é um pós-processamento da árvore construída no intuito de remover alguns de seus ramos, removendo aquelas partes da árvore que correspondem a divisões que se mostram muito particulares ao conjunto de treinamento e menos representativas da tendência geral dos dados.

Nem todas as implementações de árvores de decisão possuem todos os recursos de prevenção de super-aprendizado descritos aqui.

5.1.1 Pré-poda

Um procedimento comumente empregado para pré-poda é o de limitar o número mínimo de observações em um nó da árvore: a divisão de cada subconjunto de dados é realizada enquanto todas as partes resultantes possuam pelo menos um número predefinido de observações. O nó não é dividido, caso não exista nenhuma divisão que resulte em partes que possuam, cada uma, um número mínimo de observações do conjunto de treinamento.

Essa ideia efetivamente incorpora o princípio de evitar uma divisão muito específica do conjunto de treinamento, discutido anteriormente. A árvore resultante tende a ser menos profunda do que aquela gerada no processo em que não ocorre poda. O processamento adicional é muito pequeno e requer pouca informação adicional armazenada em cada nó da árvore.

Outra ideia de pré-poda é limitar a altura que a árvore pode ter. Um nó não é dividido se ele estiver suficiente distante da raiz da árvore.

Esta distância máxima, assim como o número mínimo de observações em cada nó, são valores definidos de antemão, como hiperparâmetros do algoritmo.

5.1.2 Pós-poda

A pós-poda é aplicada após a construção usual de uma AD. A ideia é pós-processar a árvore, removendo ramos indesejáveis: ramos que podem ser responsáveis, ou parcialmente responsáveis, pelos erros que a árvore comete (ou cometerá) no conjunto de teste.

Antes de construir a árvore inicial, os **dados de treinamento** são divididos aleatoriamente em duas partes, uma das quais fará o papel do conjunto de treinamento propriamente dito, e outra que será chamada de *conjunto de poda*. Tipicamente, reserva-se um terço dos dados originais de treinamento para o conjunto de poda. Para conjuntos de treinamento com poucas observações, percentuais menores podem ser utilizados para definir o conjunto de poda. Com base no conjunto de poda, os algoritmos de pós-poda fazem estimativas do acerto que a árvore teria em um conjunto de teste desconhecido.

A técnica de poda por redução de erro (*reduced-error pruning*, em inglês, ou apenas REP) é uma das mais utilizadas. Ela consiste em simular a "contração" de cada nó não-folha da árvore. A contração de um nó não-folha N consiste em descartar suas subárvores e substituí-lo por um nó folha N', isto é, sem descendentes. Este novo nó folha N' passa a ser associado à classe predominante dentre as observações do conjunto de treinamento que alcançam o nó N durante a fase de treinamento. (Se houver empate entre classes, o desempate pode ser feito com base na classe mais frequente no conjunto de treinamento inteiro.)

A simulação desta contração é feita para verificar se ela resultaria em melhoria na classificação do conjunto de poda. Para tanto, verificam-se dois valores:

- A quantidade de observações do conjunto de poda que são classificadas incorretamente nas subárvores de N. Em outras palavras, a quantidade de observações do conjunto de poda que alcançam o nó N na árvore original, e que são classificadas incorretamente nas folhas das subárvores que descendem de N. Chamaremos este erro de *erro estático*.
- A quantidade de erros que acontecem em N' durante a classificação do conjunto de poda, isto é, quantas observações do conjunto de poda alcançam o nó N', ao serem classificadas pela árvore modificada, e são de uma classe diferente daquela à qual o nó N' está associado. Chamaremos este erro de *erro da contração*.

Se o erro estático for maior do que o erro da contração, então vale a pena realizar a contração em questão. Caso contrário, o nó N e suas subárvores são mantidos.

Os nós não-folhas são inspecionados sequencialmente, **no sentido das folhas para a raiz da árvore**. Se uma contração for vantajosa, ela é aplicada imediatamente na árvore e a avaliação de outros nós não-folhas a seguir deve ser feita já levando em conta as contrações realizadas até o momento.

A árvore resultante pode ser estritamente menor (em número de nós) do que a árvore original. Além disso, a árvore resultante não possui nenhum nó não-folha que possa ser contraído sem prejudicar a precisão da classificação do conjunto de poda, e comete um número de erros no conjunto de poda que é menor ou igual ao número de erros cometidos pela árvore original. Logo, este processo pode evitar o super-aprendizado.

Apesar de computacionalmente mais cara, esta técnica é mais utilizada do que a de pré-poda e é considerada mais efetiva em reduzir o efeito negativo do problema de super-aprendizado. O algoritmo de REP resulta em árvores menos profundas do que as árvores originais e com menos divisões pequenas que capturam nuances específicas ao conjunto de treinamento.

5.2 Dados Faltantes

Dizemos que uma observação $\mathbf{x}$ possui dado faltante para a característica X_j se o valor de X_j em $\mathbf{x}$ é desconhecido, e denotamos isso por $\mathbf{x}_j = ?$. Isso pode acontecer por várias razões, incluindo

o fato do valor coletado ter sido corrompido e registrado como um valor inválido (tal como uma idade negativa), ou simplesmente o fato de que o valor da característica não estava definido para a observação em questão (como no caso de um paciente que não realizou determinado exame).

5.2.1 Fase de Treinamento

No caso do conjunto de treinamento possuir dados faltantes, há uma alternativa simples para o procedimento de construção da árvore. Ao elencar testes com a característica X_j, as observações que não possuem valor em X_j simplesmente não são levadas em conta pelo algoritmo para o propósito de determinar valores de interesse para os testes.

Ao realizar a divisão dos dados S em função de certo teste que envolve a característica X_j, a noção de partição é deixada de lado, e o seguinte procedimento é adotado:

- Observações que possuem valor de X_j são particionadas conforme o processo usual;
- Observações que possuem valor faltante em X_j estarão presentes em todas as partes $D_1, \ldots, D_k$.

Desta forma, o teste é efetivamente ignorado, no que diz respeito à divisão daquelas observações.

5.2.2 Fase de Teste

De forma semelhante, ao lidar com a classificação de uma observação que não possui valor em X_j, os testes envolvendo X_j são ignorados, e a observação segue por mais de um caminho paralelamente ao longo da árvore. Esse processo levará a observação a mais de uma folha. Portanto, algum tipo de critério de desempate deve ser aplicado, caso as folhas estejam associadas a duas ou mais classes distintas. Uma das ideias comuns é considerar a "pureza" ou homogeneidade do subconjunto de treinamento associado àquela folha, e escolher a classe associada à folha com maior grau de pureza.

Como uma alternativa para a implementação deste processo, pode-se considerar o seguinte procedimento, para o caso de características com domínio nominal ou discreto e limitado. Em vez de seguir paralelamente por mais de um ramo da árvore, o programa pode transformar a observação em questão em $q = t_{j_1} \times t_{j_2} \times \cdots \times t_{j_k}$ observações diferentes, onde $X_{j_1}, X_{j_2}, \ldots, X_{j_k}$ são as características com valores faltantes na observação, e t_j é o número de valores no domínio da característica X_j. Ou seja, os dados faltantes são preenchidos com todas as q tuplas de valores possíveis. Assim, a árvore pode ser usada para classificar cada uma das q observações e o programa fornece como classificação da observação original uma classe obtida a partir destas q classificações (por exemplo, uma classe majoritária entre as q classificações).

Quando a quantidade de dados de treinamento é abundante, uma alternativa ainda mais simples é não utilizar observações com dados faltantes durante a fase de treinamento.

5.3 Melhorando a Eficiência

Alguns algoritmos não utilizam todos os dados para construir a árvore, com o intuito de evitar muitos cálculos. O algoritmo ID3, em particular, trabalha com uma "janela" de dados: um subconjunto de observações do conjunto de treinamento, selecionadas aleatoriamente. O algoritmo cria uma árvore de decisão **consistente** para tal janela, como se a janela fosse o próprio conjunto de treinamento. Em seguida, o algoritmo avalia o erro desta árvore na classificação das observações do

conjunto de treinamento que não pertencem à janela. Caso este erro seja não-nulo, um subconjunto das observações incorretamente classificadas é selecionado aleatoriamente para fazer parte da janela (isto é, a janela é aumentada) e o processo é repetido, até que uma árvore seja construída com a propriedade de classificar corretamente todo o conjunto de treinamento. Esta árvore é o classificador final.

Classificação com custos sobre as características

Algumas tarefas de classificação levam em conta um custo associado ao uso de cada característica. Esses custos podem representar custos de obter informações sobre exemplos da vida real. Por exemplo, o custo (ou o risco) associado a um determinado exame. Árvores diferentes podem ter o mesmo poder preditivo e custos diferentes; podemos optar por usar uma ou outra com base neste custo.

Em um cenário deste tipo, deseja-se construir um classificador que tenha, simultaneamente, boa precisão de classificação e baixo custo. Os algoritmos de construção de ADs podem ser adaptados para este caso com facilidade, por meio de algum tipo de penalização ou desconto relacionado à escolha de certa característica para participar de um teste. O custo total da árvore seria dado pela soma dos custos de cada uso de cada característica. Outra opção é a formulação da escolha da árvore explicitamente como um problema de otimização e a aplicação de alguma técnica apropriada. Você teria alguma proposta de como fazer isso?