AdapTree - Proposta de um Algoritmo para Indução de Árvores de Decisão Baseado em Técnicas Adaptativas

Hemerson Pistori
Universidade Católica Dom Bosco, Depto. Engenharia de Computação,
Campo Grande, Brasil, 79117-900
Universidade de São Paulo, Escola Politécnica,
São Paulo, Brasil, 05508-900
pistori@ec.ucdb.br

and

João José Neto
Universidade de São Paulo, Escola Politécnica,
São Paulo, Brasil, 05508-900
joao.jose@poli.usp.br

Abstract

This paper introduces AdapTree, a new algorithm for the induction of decision trees, based on adaptive techniques. One of the main feature of this algorithm is its hybrid approach, which integrates both syntactical and statistical strategies. Some experimental results are also presented indicating that the adaptive approach is useful in the construction of efficient learning algorithms.

Keywords: Machine Learning, Decision Tree Induction, Adaptive Devices.

Resumo

Este artigo apresenta o AdapTree, um novo algoritmo para indução de árvores de decisão utilizando técnicas adaptativas. Uma das principais características deste algoritmo é a utilização conjunta de estratégias sintáticas e estatísticas. Serão apresentados também os primeiros resultados experimentais que indicam que o enfoque adaptativo pode ser utilizado na produção de um novo conjunto de algoritmos eficientes para aprendizagem.

Palavras chave: Aprendizagem de Máquina, Indução de Árvores de Decisão, Dispositivos Adaptativos.

1 Introdução

A indução de árvores de decisão a partir de vetores de atributos é um paradigma bastante explorado na área da aprendizagem de máquina [16]. Na maioria das soluções utilizadas atualmente, como nos algoritmos ID3 e C4.5 [25, 26], a indução da árvore de decisão acontece em uma fase de treinamento distinta da fase de utilização do modelo induzido. Em geral, estes algoritmos constroem a árvore de cima para baixo, utilizando um método guloso para escolher, com base nos exemplos que se aplicam à subárvore corrente, o atributo que será associado à raiz desta subárvore. Grande parte das pesquisas nesta área concentram-se em encontrar novos métodos para comparar atributos e para determinar o momento em que a árvore deve deixar de ser expandida (*prunning*). Existem algumas propostas, como os algoritmos ID5 e ITI [22], que estendem a capacidade de algoritmos tradicionais permitindo que a árvore gerada inicialmente seja re-estruturada durante o aprendizado incremental de novo exemplos.

De maneira geral, os algoritmos de aprendizagem de máquina mais utilizados atualmente, incluindo os conexionistas e evolutivos, não são incrementais. Existem diversas áreas em que um aprendizado incremental eficiente seria importante, como por exemplo, em reconhecimento de escrita e fala, no qual existe a possibilidade de treinamento contínuo do algoritmo. Nestes casos, seria muito interessante que correções propostas pelo usuário pudessem ser rapidamente incorporadas ao sistema de reconhecimento.

Será apresentada neste artigo uma nova forma de resolução para o problema de indução de árvores decisão utilizando conceitos da teoria dos autômatos e de técnicas adaptativas [21]. Este novo enfoque promove o uso de técnicas sintáticas em adição às técnicas estatísticas, visando a construção de modelos que possam ser mais facilmente manipulados, tanto pelo computador, quanto pelo usuário do sistema. As técnicas adaptativas, apresentadas com mais detalhes na seção 3, facilitam a especificação de dispositivos cuja estrutura se altera continuamente, a partir de estímulos recebidos externamente. Além disto, a especificação destes dispositivos dinâmicos pode ser obtida a partir da especificação de dispositivos estáticos, pré-existentes, abrindo assim a possibilidade de reaproveitamento de tecnologias já conhecidas. O Adaptree é uma primeira proposta de algoritmo de aprendizagem baseado em árvores de decisão adaptativas.

O restante deste texto está organizado da seguinte forma: na próxima seção faz-se uma breve revisão sobre árvores de decisão e algoritmos de indução. Em seguida apresenta-se a teoria dos dispositivos adaptativos, bem como um de seus casos particulares: os autômatos adaptativos de estados finitos. O algoritmo criado segundo esta nova técnica e alguns resultados experimentais que investigam a eficiência do mesmo, podem ser encontrados nas seções 4 e 5, respectivamente. Por último, são apresentadas conclusões e sugestões para trabalhos futuros.

2 Indução de Árvores de Decisão

Árvores de decisão (ADs) são mecanismos para a representação de funções discretas sobre múltiplas variáveis (contínuas ou discretas) com características hierárquicas que facilitam a inspeção e utilização das mesmas por seres humanos. Vértices internos de uma árvore de decisão representam testes a serem efetuados sobre alguma variável X, e de cada vértice parte uma aresta para cada possível valor assumido por X. O conjunto imagem da função é representado pelo conjunto formado pelas folhas da árvore. Por exemplo, dados os conjuntos $N = \{0, 1, 2\}$, $L = \{a, b\}$, $C = \{sim, não\}$, a função

$$f : N \times L \to C = \{((0, a), sim), ((0, b), sim), ((1, a), não), ((1, b), não), ((2, a), sim), ((2, b), não)\} \quad (1)$$

pode ser representada por qualquer uma das árvores de decisão apresentadas na figura 1. No caso de variáveis contínuas, os testes são escolhidos de tal forma que particionem o domínio em intervalos (e.g. $X \leq 2.23$ e $X \geq 5.6$). Existem também algumas extensões das árvores de decisão que podem ser utilizadas para a representação de funções contínuas (no contra-domínio), entre as quais as árvores de regressão [24] e as árvores de modelos (*Model Trees*) [12].

Em uma árvore de regressão, as folhas contêm o valor médio da função, dadas às restrições impostas pelos vértices ancestrais. As árvores de modelos, embora menos exploradas na literatura, possuem um poder de expressão maior que as árvores de regressão. Nessas árvores, cada folha é ligada a uma função linear [1] sobre as variáveis do domínio. As árvores de modelos podem constituir uma alternativa interessante para mecanismos de representação de funções contínuas cujas soluções aprendidas são de difícil entendimento, como as redes neurais artificiais [29]. Neste trabalho, no entanto, iremos nos concentrar em árvores de decisão para funções discretas, que ainda são as mais utilizadas.

[1] Recentemente começaram a ser exploradas também a utilização de funções não-lineares [30].

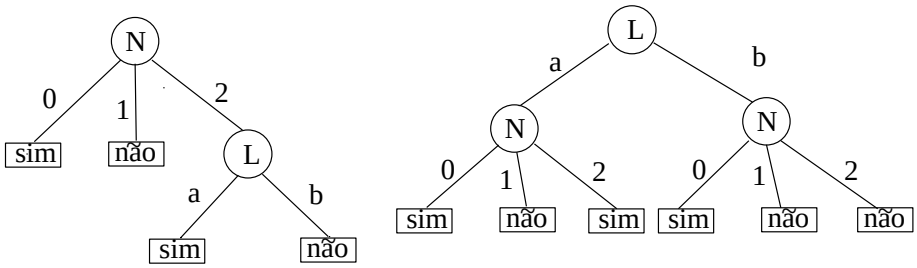

Figura 1: Duas árvores de decisão para uma mesma função

2.1 Algoritmos de Indução de ADs

Um problema importante na área da aprendizagem de máquina consiste em encontrar algoritmos capazes de construir uma AD que represente uma função desconhecida f, dado um subconjunto T, em geral impróprio, de f. A função f é támbem denominada, no jargão da área, *conceito-alvo* e o resultado produzido pelo algoritmo de indução é dito *representação do conceito*. Além disto, cada elemento de T é denominado *vetor de atributos*, ou *exemplo do conceito*, enquanto o subconjunto T constitui o *conjunto de treinamento* do algoritmo. Em geral, o conjunto de treinamento é fornecido como entrada para o algoritmo de indução, que produz como saída uma representação do conceito (e.g. árvore de decisão, rede neural). Esta representação do conceito pode ser posteriormente utilizada para calcular a função f sobre elementos que não estavam presentes no conjunto de treinamento, ou, em outras palavras, para *classificar* novos exemplos do conceito aprendido.

A maioria dos algoritmos de indução de AD segue uma mesma estratégia geral, que consiste em construir a árvore da raiz para as folhas, particionando o conjunto de treinamento de acordo com os testes escolhidos para cada vértice. Cada teste funciona assim como um filtro no conjunto de treinamento, de forma que a decisão sobre quais testes devem ser associados aos vértices próximos às folhas da árvore é geralmente baseada em um conjunto bem menor de exemplos que aqueles próximos à raiz. Por isso, costuma-se dizer que a construção da árvore de decisão segue um princípio de otimização localizada. A estratégia utilizada para comparar atributos (e.g. *information gain* [25], *chi-square statistic* [17]) é um dos fatores que distinguem os diversos algoritmos de indução. Em geral, essa estratégia é construída com o objetivo de tornar mais provável a obtenção da menor árvore de decisão (em número de vértices) capaz de representar corretamente os exemplos recebidos como entrada. A preferência por árvores de decisão mais simples baseia-se no princípio da *Occam Razor* [11], o que sugere que, dentre explicações igualmente satisfatórias, são as mais simples que devem ser escolhidas [2].

Um outro fator distintivo dos diversos algoritmos de indução de AD é a maneira como é determinado o ponto em que o algoritmo deve interromper a expansão da árvore (*prunning*). Árvores muito ramificadas podem apresentar um problema muito conhecido, denominado *overfitting*, que ocorre quando um modelo é tão aderente ao conjunto de treinamento que, por ser demasiadamente específico, resulta inútil para o exemplos externos a este conjunto. Uma das maneiras mais comuns para se verificar a capacidade de generalização do modelo é utilizar, além do conjunto de treinamento, um conjunto de teste que seja independente do conjunto de treinamento, na avaliação da taxa de acerto do modelo inferido.

3 Dispositivos Adaptativos

Uma questão recorrente na área da teoria da computação, principalmente em relação aos formalismos utilizados na representação de problemas e algoritmos, é a busca do equilíbrio entre expressividade e usabilidade[3]. Máquinas de Turing, por exemplo, são formalismos altamente expressivos. No entanto, a utilização direta deste formalismo na solução de problemas reais é muito desconfortável. Por outro lado, máquinas de estados finitos são fáceis de usar, mas possuem um poder de expressão muito restrito, o que prejudica a sua aplicação direta na solução de boa parte dos problemas reais.

Uma alternativa para tratarmos do "dilema" entre expressividade e usabilidade é a utilização de técnicas adaptativas [20, 21], que permitem aumentar a expressividade de um formalismo convencional sem afetar a sua usabilidade. Considerando um formalismo qualquer como sendo um dispositivo computacional, podemos dizer que tal extensão é obtida através da conexão, às regras que definem o dispositivo inicial, de um conjunto de regras que dotam tal dispositivo da capacidade de alterar dinamicamente sua estrutura, sem modificar

[2] Um estudo sistemático e moderno relacionado a este princípio pode ser encontrado em [11].
[3] No sentido normalmente usado pelos Engenheiros de Software.

sua sintaxe e semântica originais. Embora as técnicas adaptativas tenham sido ainda pouco exploradas, diversos exemplos de utilização bem sucedida na solução de alguns problemas reais podem ser encontradas em [1, 5, 4]. Na próxima seção descrevemos um tipo de dispositivo adaptativo que servirá de base para a construção do nosso algoritmo de indução de árvores de decisão, e aproveitaremos também para mostrar com mais detalhes como são utilizadas as técnicas adaptativas.

3.1 Autômatos Adaptativos de Estados Finitos

Um autômato adaptativo de estados finitos (AAF) é um dispositivo adaptativo que estende o poder de expressão dos conhecidos autômatos de estados finitos (AF). De maneira informal, um AAF é um AF capaz de alterar sua estrutura (conjunto de estados e função de transição), durante o processamento de uma cadeia. Na formulação original [19], estas alterações restringem-se a acrescentar ou remover transições e obter informações sobre a configuração do autômato no momento em que são acionadas. Sabe-se que a adição destas caraterísticas expande significativamente a capacidade de expressão de um AF: AAFs são capazes de reconhecer linguagens não-regulares livres de contexto e dependentes de contexto [19]. Na verdade, o poder de um AAF é o mesmo de um Máquina de Turing [15]. Formalmente um AAF pode ser representado por uma octupla $\mathrm{M} = \langle Q, \Sigma, q_0, F, \delta, Q_\infty, \Gamma, \Pi \rangle$ onde os primeiros cinco elementos referem-se ao mecanismo não adaptativo subjacente (um AF) inicial:

$Q \subseteq Q_\infty$ é um subconjunto finito de um conjunto, possivelmente infinito, de estados.

Σ é o alfabeto de entrada, finito e não vazio.

$q_0 \in Q$ é o estado inicial do autômato.

$F \subseteq Q$ é o conjunto de Estados Finais.

$\delta \subseteq (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty)$ é a relação de transição.

Os três últimos elementos de M, responsáveis pela adaptabilidade do sistema, são definidos a seguir:

Q_∞ é um conjunto, possivelmente infinito, de estados.

$\Gamma \subseteq (\{+,-\} \times (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty))$ é o conjunto de ações primitivas que podem ser executadas pelo AAF: incluir (+) ou excluir (-) transições [4].

$\Pi : (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty) \rightarrow \Gamma^*$ é uma função que associa a cada transição em δ uma seqüência de ações adaptativas primitivas. Podemos associar uma seqüência vazia a uma determinada transição para indicar que esta é uma transição "normal", não adaptativa.

Para toda a transição $x \in \delta$ onde $\Pi(x) = \epsilon$, o autômato adaptativo se reduz, funcionalmente, a um simples autômato de estados finitos. Nos outros casos, antes de executar a transição, o AAF deverá efetuar as alterações sugeridas pelas ações adaptativas associadas a mesma [5]. Após executar as ações adaptativas o AAF passa a operar com sua nova estrutura. A noção de *computação* de um AAF leva em conta as possíveis mudanças na sua estrutura, em cada instante. Uma configuração de um AAF é qualquer elemento (q, w, K, Δ) de $Q_\infty \times \Sigma^* \times 2^{Q_\infty} \times 2^{(Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty)}$, onde $q \in Q_\infty$ é o estado corrente, $w \in \Sigma^*$ é a parte da cadeia de entrada que ainda não foi lida, $K \in 2^{Q_\infty}$ é o conjunto atual de estados e $\Delta \in 2^{(Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty)}$ é o conjunto atual de transições. O par ordenado (C_1, C_2) das configurações $C_1 = (q, w, K, \Delta)$ e $C_2 = (q', w', K', \Delta')$ pertence a relação binária $\vdash_M$ (*passo do autômato*) se, e somente se, $w = aw'$ para algum $a \in \Sigma \cup \{\epsilon\}$, $w \in \Sigma^*$, com $\delta(q, a) = q'$ e além disto, as ações adaptativas, $\Pi((q, a, q'))$, associadas à tripla (q, a, q') alteram o conjunto de estados K e a relação Δ, para K' e Δ', respectivamente. Uma cadeia $w \in \Sigma^*$ é *aceita* por uma AAF M se, e somente se, existe um estado $q \in F$ tal que $(q_0, w, Q, \delta) \vdash_M^* (q, \epsilon, K'', \Delta'')$.

3.1.1 Um Exemplo de AAF

Mostraremos agora um AAF M, que implementa um classificador bastante simples, capaz de aprender a partir de exemplos positivos de um determinado conceito-alvo. Este AAF foi inspirado no coletor de nomes, um autômato capaz de tratar identificadores em linguagens de programação [19]. O alfabeto do autômato será definido por $\Sigma = \{a, b, S\}$, onde S (de "Sim" - exemplo positivo) será utilizada como sufixo

[4] Esta definição difere da original pois não torna explicita a operação de consulta. Optamos por não utilizar esta operação em favor de uma especificação mais simples, no entato, suficientemente expressiva para os nossos fins.

[5] Poderiamos permitir também a execução de ações adaptativas depois de cada transição, no entanto, isto não alteraria a capacidade de expressão do AAF [15].

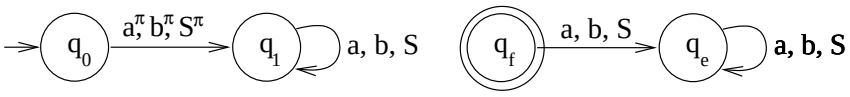

Figura 2: AAF M (O estado q_f é propositalmente inatingível)

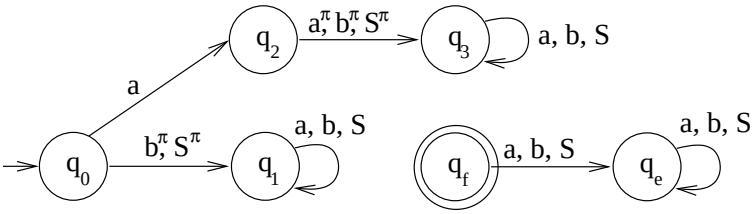

Figura 3: M após a leitura da cadeia a

de cadeias a serem aprendidas. Cadeias que não terminam em S devem ser classificadas (aceitas ou não aceitas). Uma cadeia $w\alpha$ com $\alpha \in \Sigma$ e $\alpha \neq S$, será aceita por M, se e somente se M já tiver recebido como entrada a cadeia wS (ou seja, depois que M tiver recebido um exemplo positivo, indicando que w pertence à linguagem). Por exemplo, dada a seqüência de cadeias ab , abS , ab , aa, o autômato rejeitará a primeira ocorrência de ab, aprenderá ab (ao ler abS), aceitará agora ab e por fim, rejeitará aa (pois ainda não aprendeu aa). A figura 2 mostra a estrutura subjacente do autômato classificador M = $\langle Q, \Sigma, q_0, F, \delta, Q_\infty, \Gamma, \Pi \rangle$ onde $Q = \{q_0, q_1, q_f, q_e\}$, $\Sigma = \{a, b, S\}$, $q_0 = q_0$, $F = \{q_f\}$, δ é formado pelas transições representadas na figura 2, $Q_\infty = q_n, n \in N \cup \{q_i, q_f\}$, $\Gamma = (\{+, -\} \times (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty))$ e a função Π é definida em seguida, com os símbolos q' e q'' usados na equação 3 para indicar não referenciados pelo conjunto de transições corrente.

$$\Pi(q_i, \alpha, q_i) = \epsilon, \forall\ \alpha \in (\Sigma \cup \epsilon) \quad (2)$$

$$\Pi(q_i, \alpha, q_j) = \{(-, q_i, \alpha, q_j), (+, q_i, \alpha, q'), (+, q', \beta, q''), (+, q'', \beta, q'')\}, \forall \alpha \in \{a, b\}, q_i \neq q_j\ e\ \beta \in \Sigma \quad (3)$$

$$\Pi(q_i, S, q_j) = \{(+, q_i, \epsilon, q_f)\}, \forall q_i \neq q_j \quad (4)$$

Na representação gráfica do AAF indicamos com um índice π apenas as transições associadas à ações adaptativas. As figuras 3 e 4 mostram M após a leitura dos símbolos a e S, respectivamente. Note como o autômato, na sua configuração original (figura 2), rejeitaria a cadeia a, mas depois de ler aS, ele passa a aceitá-la.

4 AdapTree - Árvores de Decisão Adaptativas

Mostraremos agora como o problema de indução incremental de árvores de decisão para atributos discretos, apresentado na seção 2, pode ser considerado do ponto de vista da teoria das linguagens. A idéia geral é transformar cada exemplo do conjunto de treinamento em um cadeia e visualizar a árvore de decisão como um tipo especial de autômato adaptativo com o estado inicial representando a raiz da árvore. No caso do algoritmo AdapTree, que será apresentado adiante, foi escolhido como exemplo um autômato adaptativo que opera de forma similar ao classificador mostrado na seção anterior, embora outras estratégias possam ser igualmente adotadas.

O conjunto de treinamento para algoritmos de indução de árvores de decisão pode ser mapeado para um conjunto de cadeias bastante restritas, todas com o mesmo comprimento e formadas por símbolos que nunca se repetem. Formalmente, dado n conjuntos disjuntos $A_1, A_2, ..., A_n$, o conjunto de treinamento é formado pelas cadeias $T = \{w_1, w_2, ..., w_m\}$, onde $|w_i| = n$, $w_i = \alpha_1\alpha_2...\alpha_n$ com $\alpha_j \in A_j$, para $1 \leq j \leq n$ e para $1 \leq i \leq m$. O último símbolo da cadeia é utilizado para representar a classe do exemplo, sendo que $|A_n|$ determina a quantidade de classes do problema e $n-1$ é o total de atributos (variáveis) a serem considerados no processo de aprendizagem (os conjuntos $A_1, A_2, ..., A_n$ representam os domínios - valores permitidos - de cada atributo). Para alternarmos o classificador do modo de aprendizagem para o modo de classificação, basta fornecemos cadeias de tamanho $n-1$ (como fizemos no exemplo da seção anterior).

O dispositivo adaptativo no qual o AdapTree se baseia pode ser visto como um AAF classificador M, estendido para trabalhar com mais de duas classes ($|A_n|$ pode ser maior que 2). Para isto, associaremos a cada estado final um elemento de A_n, que corresponderá a uma das possíveis classes. Ao receber um exemplo de treinamento (cadeia de tamanho n), o AdapTree simplesmente cria um caminho ligando o estado inicial

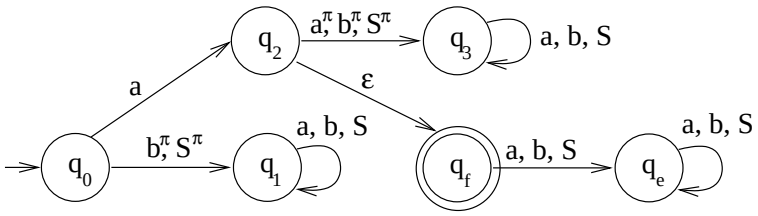

Figura 4: M após a leitura da cadeia aS

do autômato ao estado final correspondente a classe deste exemplo. Durante a classificação (ao receber cadeias de tamanho n-1) o AdapTree irá aceitar (atingir um estado final) e classificar corretamente todas as cadeias (exemplos) que ele tiver aprendido (desde que não haja inconsistências no conjunto de treinamento), e rejeitar as demais. Caso não seja possível determinar, sintaticamente, a classe de uma cadeia de entrada, o AdapTree acionará um mecanismo estatístico que oferecerá como resposta uma estimativa. Este mecanismo será apresentado a seguir.

4.1 O Mecanismo de Inferência Estatística Utilizado pelo AdapTree

Para classificar cadeias que não conduzem o AdapTree a um estado final, baseamo-nos em uma estimativa fundamentada na frequência relativa de cada classe em relação ao total de exemplos recebidos até o momento. Essa estimativa corresponde à probabilidade *a priori* (sem considerar qualquer outra informação) de um exemplo w pertencer a uma determinada classe $a_n \in A_n$, que denominaremos doravante $p(a_n)$. Para ilustrar, imagine que tenhamos recebido, dentro de um conjunto total de 1000 exemplos de treinamento, 800 exemplos pertencentes à classe *viralata*, 150 exemplos à classe *bulldog* e 50 exemplos à classe *boxer*. Na falta de qualquer informação adicional, seria razoável classificar um novo exemplo ainda não classificado como pertencente à classe *viralata*, visto que podemos estimar $p(viralata) = 0.8$.

Na verdade, o AdapTree utiliza um mecanismo um pouco mais sofisticado que este, que procura considerar, no cálculo da classe mais provável, a seqüência de símbolos já lidos pelo autômato. Ou seja, dada uma cadeia $w = a_1 a_2 ... a_m, m < n$ (m é necessariamente menor que n, caso contrário, teríamos uma resposta exata) o AdapTree irá retornar uma estimativa da probabilidade de uma determinada classe a_n dados os atributos $a_1, a_2, ..., a_m$: $p(a_n | a_1, a_2, ..., a_m)$. Esta estimativa pode ser facilmente calculada, computando-se na subárvore iniciada em a_m, o total de exemplos de cada classe sobre o total de exemplos representados nesta subárvore (que são justamente os exemplos cujo prefixo é igual a $a_1 a_2 ... a_m$).

É importante notar que a estimativa acima é altamente dependente da ordem dos atributos, visto que permutações dos mesmos produzirão árvores diferentes (com o primeiro atributo no nivel 0 (raiz), o segundo atributo no nível 1, e assim por diante). Além disto, pode-se perceber que os atributos mais próximos da raiz serão os mais utilizados na classificação de novos exemplos. Isto ocorre porque sempre que o autômato estiver lendo um símbolo $a_j \in A_j$, ele já terá lido os símbolos $a_1, a_2, ..., a_i, i < j$ e $a_i \in A_i$.

Devido à alta sensibilidade do método quanto à ordem dos atributos, o AdapTree utiliza uma estratégia adicional para melhorar sua capacidade de estimar a classe de um exemplo desconhecido, e que consiste em permutar, convenientemente, a ordem inicial dos atributos. Para comparar dois atributos, e decidir qual deles deve ficar mais próximo da raiz, o AdapTree emprega a mesma técnica utilizada pelo Id3: *information gain* [26]. O *information gain* de um atributo qualquer, em relação ao conceito a ser aprendido, mede a redução esperada da entropia associada a distribuição das classes do atributo alvo, quando o conjunto de treinamento é particionado pelos valores de um atributo. A entropia, no contexto da aprendizagem de máquina, mede a homogeneidade de um conjunto de exemplos, sendo mais baixa quando a maioria dos exemplos pertencem apenas a uma das classes e mais alta quando os exemplos estão proporcionalmente distribuidos entre as diversas classes possíveis. Uma redução na entropia de conjunto de exemplos corresponde a um aumento na quantidade de informação em relação as classes representadas por estes exemplos. Um estudo detalhado sobre a relação entre entropia, informação e avaliação de atributos pode ser encontrado em [18].

4.2 Considerações em Relação as Capacidades Incrementais do AdapTree

As estimativas utilizadas pelo AdapTree para ordenar os atributos são calculadas a partir do conjunto de treinamento disponível ao início de sua execução. No entanto, como o AdapTree é um algoritmo incremental, novos exemplos podem ser incorporados durante o processo de classificação. Em geral, quanto maior a diversidade dos dados, melhores serão as estimativas, por isto, o AdapTree busca de tempos em tempos

reordenar sua árvore de decisão caso identifique que algum atributo passou a ser mais significativo que outro. Para isto, o AdapTree recalcula, a cada novo exemplo inserido, o *information gain* de cada um dos atributos. Este cálculo pode ser feito em tempo constante no total de exemplos, utilizando-se estruturas de dados adequadas para acumular as quantidades de exemplos pertencentes a cada classe e para cada um dos atributos. Isto só é possivel por que, diferente do Id3, o AdapTree não particiona o conjunto de exemplos, ou seja, o princípio de otimização é global e não local.

Uma vez que a árvore de decisão mantida pelo AdapTree pode ser mapeada em uma tabela de decisão, com cada nível da árvore correspondendo a uma coluna da tabela, podemos utilizar os métodos númericos descritos em [23] para realizar a reordenação da árvore. Com isto, é possível obter uma nova árvore em um tempo que é independente do tamanho do conjunto de treinamento. É importante ressaltar que grande parte dos algoritmos existentes para a indução incremental de árvores de decisão dependem de operações de reestruturação da árvore, que são lineares sobre o tamanho do conjunto de treinamento [22].

5 Primeiros Experimentos

O algoritmo AdapTree foi implementado a partir da biblioteca Weka [6] (Waikato Environment for Knowledge Analysis), de apoio ao desenvolvimento de programas para aprendizagem de máquina e *data mining* . Esta biblioteca, escrita em Java, além de oferecer implementações de diversos algoritmos bem aceitos de aprendizagem de máquina, permite que módulos comuns a todos os algoritmos sejam reaproveitados na construção de novos algoritmos. O Weka disponibiliza, por exemplo, códigos fonte para a manipulação de vetores de atributos, aplicação de funções estatísticas, aplicações de filtros (discretização de atributos, retirada de *missing-values*, normalização, "embaralhamento",...) entre outros.

Um outro recurso poderoso do Weka é a possibilidade de criação de experimentos para comparação automática do desempenho de diversos algoritmos. Para obter uma primeira impressão do desempenho do AdapTree, utilizamos o Weka para compará-lo com os algoritmos Id3 [26], Naive Bayes [7], C4.5 [26], 5NN [18] (*K-Nearest Neighbour* com K = 5) e Redes Neurais com *backpropagation* [18]. Todos os algoritmos foram utilizados com as opções-padrão disponíveis no Weka e as estatísticas foram obtidas a partir da média aritmética de 10 execuções de avaliação cruzada estratificada de 10 dobras [7], ou seja, cada algoritmo foi executado 100 vezes para cada conjunto de dados. Além disto, os algoritmos foram testados com 5 diferentes conjuntos de dados.

Os conjuntos de dados utilizados são todos de domínio público e extraídos do repositório da Universidade da Califórnia, UCI [2]. O primeiro deles, citado em diversos artigos, nas áreas de computação e também de estatística [10, 27, 3, 14, 8], oferece 150 exemplos de plantas do tipo Iris, classificados a partir de 4 atributos numéricos indicando largura e espessura de suas pétalas e sépalas. O conjunto *Ionosphere* contém 351 exemplos e 34 atributos representando dados de um radar utilizado para detectar a presença de elétrons livres na ionosfera [28]. Temos ainda dois conjuntos relacionados com problemas da área de diagnóstico médico, *hypothyroid* (3772 exemplos, 30 atributos) e *hepatitis* [6, 13] (155 exemplo, 20 atributos), e o último, *glass*, representa um problema de identificação do tipo de uma amostra de vidro. O resultado desta identificação pode ser utilizado como evidência em processos de investigação criminal [9].

A tabela 1 mostra as taxas de acerto e os tempos médios para treinamento (entre parênteses e em segundos) obtidos pelo Weka, para cada um dos algoritmos comparados e em cada conjunto de dados. Estes resultados indicam que não existe diferença significativa (maior que 2%) na taxa de acerto entre o AdapTree e os demais algoritmos nos conjuntos *iris* e *hypothyroid*. No entanto, o *backpropagation* é significativamente superior nos outros conjuntos, embora o AdapTree continue tendo um bom desempenho em relação aos outros algoritmos. Nota-se também que, embora o *backpropagation* apresente uma alta taxa de acerto em relação ao AdapTree, o tempo utilizado para treinamento é mais que 100 vezes maior.

Um resultado importante, mostrado na tabela 1, é a superioridade do AdapTree em relação ao Id3, que é o único dos 5 algoritmos analisados que não possui tratamento especial para atributos numéricos. Tanto o AdapTree, quanto o Id3, só puderam operar sobre os atributos numéricos porque foram aplicados, sobre o conjunto de treinamento, filtros que discretizaram estes atributos. O processo de discretização consistiu apenas em separar os valores contínuos em 5 grupos de igual tamanho o que, certamente coloca estes dois algoritmos em desvantagem em relação aos algoritmos capazes de manipular diretamente estes valores e aproveitar ao máximo a informação ali contida.

Ainda em relação ao tempo de execução, as tabelas 1 e 2 indicam que o AdapTree possui um desempenho comparável ao C4.5, tanto em tempo consumido na fase de treinamento quanto na fase de classificação. Estes tempos foram obtidos a partir da média dos 100 experimentos utilizados para obter as taxas de acertos, e também são gerados automaticamente pelo software Weka. Todos os programas foram executados em uma

[6] Software livre e gratuito disponível em http://www.cs.waikato.ac.nz/ ml/weka/

[7] *10-Fold Stratified Crossvalidation*

Dataset	AdapTree	Id3	NBayes	5NN	C4.5	Backpro
iris	93.92% (0.03s)	89.22% (0.88s)	93.33% (0s)	92.75% (0s)	94.31% (0.01s)	93.08% (1.83s)
ionosphere	86.72% (0.66s)	84.29% (2.09s)	90.76% (0.03s)	89.58% (0.01s)	87.39% (0.48s)	93.84% (74.66s)
hypothyroid	92.22% (5.18s)	90.12% (29.05s)	92.78% (0.17s)	93.28% (0.07s)	93.41% (4.37s)	94.57% (830.67s)
hepatitis	76.6 % (0.1s)	70.75% (1.23s)	84.15% (0s)	81.13% (0s)	80.75% (0.06s)	82.39% (10.5s)
Glass	52.47% (0.07s)	47.95% (5.86s)	52.19% (0.01s)	53.42% (0s)	52.47% (0.11s)	68.95% (10.71s)

Tabela 1: Comparação de taxas de acerto e tempos de execução

mesma plataforma (CPU Pentium III - 700Mhz, 257Mb RAM). A tabela 2 mostra também que o tamanho das árvores de decisão gerada pelo AdapTree, em geral maiores que as geradas pelo Id3 e C4.5, não compromete o tempo de execução do algoritmo na fase de classificação. Apenas a título de ilustração, as tabelas apontam claramente os conhecidos "pontos fracos" do *backpropagation* (alto tempo de treinamento) e do KNN (alto tempo de classificação).

Dataset	AdapTree	Id3	NBayes	5NN	C4.5	Backpro
iris	0s	0s	0.01s	0.05s	0.01s	0.01s
ionosphere	0.01s	0.01s	0.04s	1.3s	0.01s	0.13s
hypothyroid	1.3s	0.11s	0.45s	184.48s	0.07s	1.17s
hepatitis	0s	0s	0.01s	0.18s	0s	0.02s
Glass	0.01s	0s	0.02s	0.37s	0.01s	0.01s

Tabela 2: Comparação de Tempo para Classificação

É importante ressaltar que todos estes algoritmos podem ter seu desempenho melhorado através de ajustes finos em seus parâmetros, a partir da analise dos conjuntos de treinamento. Para tentar evitar qualquer tipo de polarização que viesse a favorecer nosso algoritmo, optamos por não fazer qualquer ajuste fino com base nos conjuntos de treinamento. Os próprios conjuntos de treinamento foram escolhidos ao acaso, a partir de um repositório bastante conhecido e muito utilizado pelos pesquisadores da área de Aprendizagem de Máquina.

6 Conclusões e Trabalhos Futuros

Apresentamos neste trabalho um novo algoritmo para indução de Árvores de Decisão cujo desempenho foi constatado comparável aos principais algoritmos de Aprendizagem de Máquina conhecidos. Um fator importante a ser destacado é o novo enfoque que foi apresentado para a construção deste tipo de programa, utilizando a teoria dos autômatos , enriquecida com a tecnologia adaptativa. Este enfoque cria um elo entre as técnicas usualmente estudadas para a indução de árvores de decisão, e a teoria das linguagens e autômatos. Esperamos que este elo possa ser utilizado no intercâmbio de técnicas entre as duas áreas.

Embora o AdapTree tenha como característica a possibilidade de aprendizagem incremental eficiente, neste trabalho ele não foi comparado com outros algoritmos incrementais. Esta comparação é um dos nossos objetivos futuros, além da análise de outras técnicas para ordenação de atributos, tratamento de atributos numéricos, outras técnicas para "inferência estatística" e a demonstração de resultados relacionado à complexidade em tempo e espaço do algoritmo apresentado.

Referências

[1] B. A. Basseto and J. J. Neto. A stochastic musical composer based on adaptative algorithms. *Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação. SBC-99.*, 3:105–130, Julho 1999.

[2] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[3] B.V. Dasarathy. Nosing around the neighborhood: A new system structure and classification rule for recognition in partially exposed environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1):67–71, 1980.

[4] Ricardo Luis de Azevedo da Rocha. *Um Método de Escolha Automática de Soluções Usando Tecnologia Adaptativa.* PhD thesis, Escola Politécnica, Universidade de Sao Paulo, São Paulo, Brasil, Julho 2000. [in portuguese].

[5] Carlos Eduardo Dantas de Menezes. *Um Método para a Construção de Analisadores Morfológicos, Aplicado à Lingua Portuguesa, Baseado em Autômatos Adaptativos.* PhD thesis, Escola Politécnica, Universidade de Sao Paulo, São Paulo, Brasil, Julho 2000. [in portuguese].

[6] P. Diaconis and B. Efron. Computer-intensive methods in statistics. *Scientific American*, 248, 1983.

[7] Pedro Domingos and Michael J. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *International Conference on Machine Learning*, pages 105–112, 1996.

[8] Cheeseman et alli. Autoclass: A bayesian classification system. *Proceedings of the International Machine Learning Conference*, pages 54–64, 1988.

[9] Ian W. Evett and E. J. Spiehler. Rule induction in forensic science. In *KBS in Goverment*, pages 107–118. Online Publications, 1987.

[10] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7, Part II:179–188, 1936.

[11] M. R. Forster. *The New Science of Simplicity*, pages 83–119. Cambridge University Press, 2001.

[12] Eibe Frank, Yong Wang, Stuart Inglis, Geoffrey Holmes, and Ian H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.

[13] I. Konenenko G. Cestnik and I. Bratko. *Progress in Machine Learning*, chapter Assistant-86: A Knowledge-Elicitation Tool for Sophisticated Users, pages 31–45. Sigma Press, 1987.

[14] G.W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, pages 431–433, May 1972.

[15] M. K. Iwai. *Um Formalismo Gramatical Adaptativo para Linguagens Dependentes de Contexto.* PhD thesis, Escola Politécnica, Universidade de São Paulo, São Paulo, Brasil, 2000. [in portuguese].

[16] R. Kothari and M. Dong. *Pattern Recognition: From Classical to Modern Approaches*, chapter 6, Decision Trees For Classification: A Review and Some New Results, pages 169–184. World Scientific, 2001.

[17] J. Mingers. Expert systems - rule induction with statistical data. *Journal of the Operational Research Society*, 38:39–47, 1987.

[18] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[19] J. J. Neto. *Contribuição a Métodologia de Construção de Compiladores.* PhD thesis, Escola Politécnica, Universidade de São Paulo, São Paulo, Brasil, 1993. [in portuguese].

[20] J. J. Neto. Solving complex problems efficiently with adaptative automata. *Conference on Implementation and Application of Automata - CIAA 2000*, July 2000.

[21] J. J. Neto. Adaptative rule-driven devices - general formulation anda case study. *Conference on Implementation and Application of Automata - CIAA 2001*, July 2001.

[22] Neil C. Berkman Paul E. Utgoff and Jeffery A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, October 1997.

[23] W. Del Picchia. *Métodos numéricos para a resolução de problemas lógicos.* Edgar Blucher, 1993.

[24] J. R. Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[25] J. R. Quinlan. *C4.5 Programs for Machine Learning.* Morgan Kaufmann, 1993.

[26] J. R. Quinlan. Learning decision tree classifiers. *ACM Computing Surveys*, 28(1), 1996.

[27] R.O.Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, 1973.

[28] S. P. Wing et alli V. G. Sigillito. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262–266, 1989.

[29] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 2000.

[30] Z. H. Zhou and Z.Q.Chen. Hybrid decision tree. *Knowledge-Based Systems*, 0(0):in press, 2002.