HTILDE-RT: UM ALGORITMO DE APRENDIZADO DE ÁRVORES DE REGRESSÃO DE LÓGICA DE PRIMEIRA ORDEM PARA FLUXOS DE DADOS RELACIONAIS

Glauber Marcius Cardoso Menezes

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Gerson Zaverucha

Rio de Janeiro
Dezembro de 2011

HTILDE-RT: UM ALGORITMO DE APRENDIZADO DE ÁRVORES DE REGRESSÃO DE LÓGICA DE PRIMEIRA ORDEM PARA FLUXOS DE DADOS RELACIONAIS

Glauber Marcius Cardoso Menezes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Gerson Zaverucha, Ph.D.

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof[a]. Bianca Zadrozny, Ph.D.

Prof[a]. Marley Maria Bernardes Rebuzzi Vellasco, Ph.D.

Prof. Adriano Alonso Veloso, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2011

Cardoso Menezes, Glauber Marcius

HTILDE-RT: UM ALGORITMO DE APRENDIZADO DE ÁRVORES DE REGRESSÃO DE LÓGICA DE PRIMEIRA ORDEM PARA FLUXOS DE DADOS RELACIONAIS/Glauber Marcius Cardoso Menezes. – Rio de Janeiro: UFRJ/COPPE, 2011.

XIV, 91 p.: il.; 29, 7cm.

Orientador: Gerson Zaverucha

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2011.

Referências Bibliográficas: p. 88 – 91.

1. Aprendizado de máquina. 2. Programação em lógica indutiva. 3. Regressão. 4. Escalabilidade de algoritmos. 5. Mineração em grandes bases de dados. 6. Fluxos de dados. I. Zaverucha, Gerson. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meu pais, Nelson (in memoriam) e Joana, sem os quais nenhuma de minhas realizações seria possível. Tudo que há de bom em mim tem sua contribuição.

Agradecimentos

Agradeço primeiramente aos meus pais, Joana D'arc de Oliveira Menezes e Nelson Cardoso Menezes, por todo carinho, atenção e dedicação investidos em minha criação e por todos os seus ensinamentos, os quais foram fundamentais para minha formação moral e intelectual. Sem eles, eu nada seria.

À minha namorada Sarah Mendes, por todo o carinho e apoio, compartilhando de perto minhas derrotas e vitórias, meus erros e acertos.

Aos meus amigos de longa data: André Lima, Aprígio Bertholdo, Diego Abrantes, João Carlos da Rosa, Leandro Lessa, Maximiliano Iabrudi, Rafael Abrantes, Renan Marcolino, Vinícius Ferreira, William de Carvalho e Yuri Villar; por estarem ao meu lado ontem, hoje e sempre.

Aos amigos conquistados graças à minha passagem pela UFRJ: Alexandre Rabelo, Alexandre Sardinha, Carlos Wagner, Douglas Cardoso, Fellipe Duarte, Lúcio Paiva, Paulo Braz, Rafael Espírito Santo e Rodrigo Barbosa; por serem uma inesgotável fonte de sabedoria e companheirismo.

Ao amigo Elias Bareinboim, por ter influenciado positivamente na minha decisão de adentrar no mestrado desta instituição.

Ao grande amigo Dr. Celso Lugão da Veiga, por ter acompanhado de perto esta minha caminhada, zelando por meu bem-estar e transferindo-me conhecimentos valiosíssimos para o resto de minha vida.

Ao meu recém-amigo Dr. Gustavo Coutinho Medeiros de Andrade, por estar cuidando de minha saúde com maestria, por seus valiosos conhecimentos e sua influência impactante em minha vida.

Aos amigos e colegas de orientação: Cristiano Pitangui e Juliana Bernardes, pela excelente interação que tivemos; e, em particular, a Ana Duboc e Aline Paes por terem me ajudado inúmeras vezes mais proximamente.

A Carina Lopes, por ter sido tão solícita e atenciosa, transmitindo-me seus conhecimentos e esclarecendo-me incontáveis dúvidas sobre o sistema ACE e os algoritmos TILDE e HTILDE.

Ao meu orientador, professor Gerson Zaverucha, pelos conhecimentos a mim transferidos e por acreditar em mim, dedicando-me seu tempo e paciência em momentos em que duvidei de mim mesmo.

A Hendrik Blockeel, criador do TILDE, e Jan Ramon, por fornecerem o código fonte do sistema ACE e permitirem que este trabalho pudesse basear-se em seu sistema.

Agradeço especialmente: a Jan Struyf que, mesmo não mais estando no ambiente acadêmico, foi extremamente solícito, ajudando-me a resolver problemas relacionados ao sistema ACE; e a Celine Vens, por fornecer os datasets sintéticos, bem como explicações pertinentes aos mesmos.

Agradeço ainda a Celine Vens, Daan Fierens, Jan Ramon e Hendrik Blockeel pela ajuda com dúvidas acerca do funcionamento de elementos do sistema.

À equipe de funcionários do PESC: Cláudia, Gutierrez, Roberto Rodrigues, Solange, Sônia Regina e demais, por mostrarem-se tão prestativos e atenciosos em todos os momentos que os solicitei.

Ao excelente corpo docente do PESC e, em particular, aos professores com os quais cursei as disciplinas que escolhi.

Agradeço à CAPES e ao CNPq pela ajuda financeira.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

HTILDE-RT: UM ALGORITMO DE APRENDIZADO DE ÁRVORES DE REGRESSÃO DE LÓGICA DE PRIMEIRA ORDEM PARA FLUXOS DE DADOS RELACIONAIS

Glauber Marcius Cardoso Menezes

Dezembro/2011

Orientador: Gerson Zaverucha

Programa: Engenharia de Sistemas e Computação

Atualmente, organizações modernas armazenam seus dados sob a forma de bancos de dados relacionais que crescem mais rapidamente que a capacidade de hardware. Entretanto, a extração de informação tornou-se uma tarefa crucial para a sobrevivência das corporações. Neste trabalho, propomos o algoritmo HTILDE-RT, um algoritmo incremental escalável para aprender de forma eficiente árvores de regressão de lógica de primeira ordem em fluxos de dados relacionais. O HTILDE-RT é baseado no sistema ILP de regressão TILDE-RT e no sistema proposicional de aprendizado em fluxos VFDT. O algoritmo proposto utiliza o limitante de Hoeffding para tornar o processo de aprendizado escalável. O HTILDE-RT foi comparado com o TILDE-RT em grandes massas de dados, com 2 milhões de exemplos cada, acelerando o aprendizado entre $2,4$ e 260 vezes, gerando predominantemente modelos menores e sem perdas estatisticamente significativas em relação ao coeficiente de correlação de Pearson, porém com discreta perda em relação ao RMSE, entre 0% a 2% maior. Esses resultados experimentais sugerem uma boa troca entre escalabilidade (velocidade e tamanho) e acurácia. Adicionalmente, apresentamos novos resultados obtidos com o algoritmo ILP de classificação HTILDE, um dos sistemas base do HTILDE-RT.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

HTILDE-RT: A FIRST ORDER LOGIC REGRESSION TREE LEARNER FOR RELATIONAL DATA STREAMS

Glauber Marcius Cardoso Menezes

December/2011

Advisor: Gerson Zaverucha

Department: Systems Engineering and Computer Science

Currently, modern organizations store their data under the form of relational databases which grow faster than hardware capacities. However, extracting information from such databases has become crucial for corporations survival. In this work we propose HTILDE-RT, a scalable incremental algorithm to learn first-order logical regression trees efficiently from relational data streams. HTILDE-RT is based on the regression ILP system TILDE-RT and the propositional data stream learner VDFT. The proposed algorithm uses the Hoeffding bound to scale up the learning process. HTILDE-RT was compared with the batch learner TILDE-RT over large regression datasets, with two million examples each, speeding up the learning time between 2.4 and 260 times, generating predominantly shorter models and without showing significant statistical differences with respect to Pearson coefficient, but with a discrete loss in RMSE, between 0% and 2%. These experimental results suggest a good trade-off between scalability (speed and size) and accuracy. Moreover, we show new results yielded by the classification ILP system HTILDE, a base system of HTILDE-RT.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Mineração em fluxos de dados tornou-se foco de pesquisa em variados campos da Ciência da Computação tais como Aprendizado de Máquina e Mineração de Dados [5]. Como exemplos de onde técnicas para fluxos são úteis podemos citar: aplicações de suporte a transações de cartões de crédito, tráfego de Internet, monitoramento em telecomunicações e pesquisa científica em dados biológicos.

Os bancos de dados das organizações modernas crescem numa velocidade superior à da evolução da capacidade de hardware. Entretanto, extrair informações ocultas em tais bancos tornou-se uma necessidade crítica para competitividade e sobrevivência das corporações, sendo portanto atividade estratégica indispensável.

Atualmente, as bases de dados modernas são estruturadas sob o paradigma relacional. Porém, algoritmos de aprendizagem tradicionais não são perfeitamente aplicáveis em tal tipo de dados, pois utilizam o paradigma proposicional para representar o conhecimento, cuja capacidade de representação não permite descrever relações entre objetos nos dados. Uma outra importante questão é o fato de que os algoritmos tradicionais não são capazes de manipular massas de dados tão grandes, pois necessitam que todos os exemplos estejam na memória principal ao mesmo tempo, o que é impraticável no contexto atual.

Em tal ambiente dados novos chegam em taxas elevadas e um algoritmo eficiente deve processá-los prontamente, porém a restrição de armazená-los como um todo na memória é irrealizável. Dado isto, as áreas de Aprendizado de Máquina e Mineração de Dados direcionaram esforços para sanar tal tipo de demanda de forma eficiente.

Domingos e Hulten [4] desenvolveram uma metodologia para tratar fluxos de dados, que culminou no algoritmo VFDT (Very Fast Decision Trees). O VFDT é um algoritmo proposicional, que usa uma técnica de amostragem baseada no limitante de Hoeffding [6] para escolher qual é o melhor teste a ser introduzido no nó de uma árvore de decisão, sem a necessidade de analisar todos os exemplos ao mesmo tempo. O VFDT é o estado da arte para este tipo de problema.

O TILDE (Top-down Induction of Logical Decision Trees) [3] é a versão lógica

de primeira ordem do algoritmo de árvores de decisão, o qual é capaz de modelar a grande maioria dos esquemas relacionais nos bancos de dados modernos, sendo, portanto, capaz de aprender a partir de dados descritos por mais de uma relação. Sua eficiência é garantida através do emprego das técnicas de pacotes de cláusulas (Query Packs) [7] e Aprendizado por Interpretações (Learning from Interpretations) [2]. Tal algoritmo é implementado como parte do sistema de Programação em Lógica Indutiva (ILP) ACE [8].

Para estender a metodologia utilizada pelo VFDT à lógica de primeira ordem, Lopes e Zaverucha [9] modificaram o TILDE, desenvolvendo o algoritmo HTILDE (Hoeffding TILDE), baseado no TILDE e no VFDT. Portanto, o HTILDE é a versão em lógica de primeira ordem do VFDT capaz de tratar eficientemente problemas de classificação a partir de fluxos de dados relacionais.

Neste trabalho apresentamos o HTILDE-RT, que é uma modificação do HTILDE para o tratamento de regressão a partir de fluxos de dados relacionais. Da mesma forma que o HTILDE, o HTILDE-RT combina a expressividade do paradigma da lógica de primeira ordem e a eficiência da técnica de amostragem guiada pelo limitante de Hoeffding. Um importante motivador está no fato de que as árvores de regressão são modelos interessantes, pois são capazes de descrever funções numéricas mantendo a interpretabilidade e eficiência das árvores de decisão.

Esta dissertação está organizada da seguinte maneira: no capítulo 2, são apresentados conceitos essenciais para este trabalho, como árvores de decisão e regressão, programação em lógica indutiva e avaliação de aprendizado; no capítulo 3, apresentaremos os seguintes algoritmos relacionados a este trabalho: TILDE na seção 3.1, VFDT na seção 3.2 e HTILDE na seção 3.3; no capítulo 4, discutiremos as propriedades assintóticas dos modelos aprendidos pelo sistema ILP de classificação HTILDE, base do HTILDE-RT, bem como apresentaremos novos resultados experimentais; no capítulo 5, detalharemos o algoritmo HTILDE-RT, bem como os resultados experimentais obtidos; e no capitulo 6, expomos as conclusões deste trabalho, bem como enumeramos desejáveis trabalhos futuros.

Capítulo 2

Conceitos fundamentais

Neste capítulo, conceitos fundamentais relacionados a este trabalho serão apresentados de acordo com a seguinte organização: na seção 2.1, explicaremos o aprendizado de árvores de decisão para problemas de classificação; na seção 2.2, o aprendizado de árvores de regressão será discutido; na seção 2.3, introduziremos o conceito de programação em lógica indutiva bem como sua aplicação em problemas de regressão; e na seção 2.4, trataremos conceitos relacionados a avaliação do aprendizado.

2.1 Árvores de Decisão

Uma árvore de decisão é um modelo capaz de descrever conceitos, de forma aproximada, através de regras que mapeiam um objeto em um valor específico. Tais regras são representadas através de um grafo do tipo árvore, onde a cada nó interno corresponde um teste. O mapeamento de um objeto é feito atravessando-o ao longo da árvore, partindo-se da raiz até chegar em uma folha. Ao alcançar um nó interno, o objeto é confrontado com o teste associado àquele e o resultado do teste determina o próximo nó da travessia. Finalmente, ao chegar em uma folha, o objeto recebe um valor de acordo com uma função associada à mesma. Os algoritmos clássicos para aprendizado de árvores de decisão são o ID3 [10] e seu sucessor o C4.5 [11].

2.1.1 Representação do conhecimento

Um ponto fundamental para o aprendizado e inferência nas árvores de decisão, bem como em qualquer algoritmo de aprendizado, é a maneira como a informação é estruturada e representada nos dados. A representação mais comum é conhecida como *atributo-valor*, ou ainda *proposicional*, onde todos os objetos nos dados são compostos por um mesmo conjunto fixo de atributos, e cada atributo possui um conjunto de possíveis valores que pode assumir (seu domínio). Caso um atributo

seja numérico, ou possivelmente contínuo, costuma-se adotar um passo prévio de discretização, porém existem outras formas para tratar atributos não categóricos conforme veremos na seção 2.2.2. Essencialmente, dados proposicionais podem ser representados através de uma tabela única.

Para auxiliar nas explicações, o exemplo do conceito *Jogar Tênis* será usado. Na tabela 2.1 existem informações meteorológicas sobre alguns dias previamente observados, juntamente com a informação da decisão que foi tomada, que pode ser jogar (Sim) ou não jogar tênis (Não).

Dia	**Tempo**	**Temperatura**	**Umidade**	**Vento**	**Jogar Tênis**
D1	Ensolarado	Quente	Alta	Fraco	Não
D2	Ensolarado	Quente	Alta	Forte	Não
D3	Nublado	Quente	Alta	Fraco	Sim
D4	Chuvoso	Amena	Alta	Fraco	Sim
D5	Chuvoso	Fria	Normal	Fraco	Sim
D6	Chuvoso	Fria	Normal	Forte	Não
D7	Nublado	Fria	Normal	Forte	Sim
D8	Ensolarado	Amena	Alta	Fraco	Não
D9	Ensolarado	Fria	Normal	Fraco	Sim
D10	Chuvoso	Amena	Normal	Fraco	Sim
D11	Ensolarado	Amena	Normal	Forte	Sim
D12	Nublado	Amena	Alta	Forte	Sim
D13	Nublado	Quente	Normal	Fraco	Sim
D14	Chuvoso	Amena	Alta	Forte	Não

Tabela 2.1: Objetos associados ao conceito *Jogar Tênis*.

Neste exemplo, os objetos são os dias, linhas da tabela, os atributos são as informações meteorológicas, colunas, e o conceito que se deseja representar é o de quando jogar, ou não jogar, tênis.

No grafo em forma de árvore, cada nó é associado a um atributo e cada possível valor deste fica associado a uma aresta. Na figura 2.1 há um exemplo de árvore de decisão para o conceito *Jogar Tênis*. Para saber qual decisão tomar em um dia específico, basta confrontar os atributos deste objeto com os testes ao longo da árvore, até que uma folha seja alcançada.

Suponha que fosse necessário saber qual decisão tomar num dia com as seguintes características (atributos): ensolarado, temperatura fria, umidade normal e vento forte. Suponha, também, que foi fornecida a árvore da figura 2.1. De acordo com esta, a decisão seria "Sim", isto é, 'jogar'. Note que, não há tal registro na tabela, logo não seria possível responder tal pergunta apenas consultando a mesma.

Conforme dito, tal representação codifica um conjunto de regras [3], portanto, como consequência da estrutura e do processo para a tomada de decisão, cada folha é na verdade associada a cabeça de uma regra, cujo corpo é formado pela conjunção

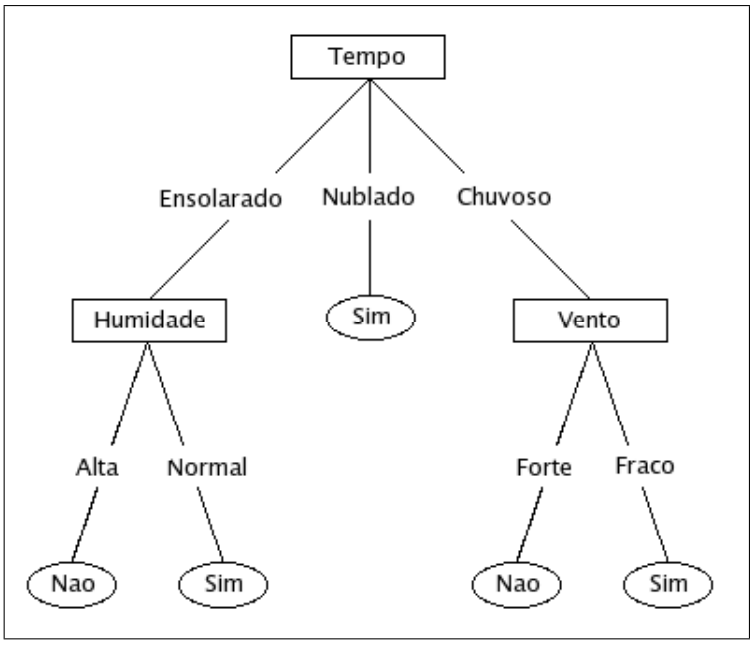


Figura 2.1: Exemplo de uma árvore de decisão para o conceito *Jogar Tênis*.

dos testes associados ao caminho da raiz até a folha em questão. Por exemplo, a árvore da figura 2.1 expressa o seguinte conjunto de regras na figura 2.2.

SE $Tempo = Ensolarado$ E $Umidade = Alta$ ENTÃO $Classe = Não$
SE $Tempo = Ensolarado$ E $Umidade = Normal$ ENTÃO $Classe = Sim$
SE $Tempo = Nublado$ ENTÃO $Classe = Sim$
SE $Tempo = Chuvoso$ E $Vento = Forte$ ENTÃO $Classe = Não$
SE $Tempo = Chuvoso$ E $Vento = Fraco$ ENTÃO $Classe = Sim$

Figura 2.2: Conjunto de regras correspondente à árvore da figura 2.1.

2.1.2 Ganho de informação

Antes de discutir o algoritmo de aprendizado de árvores de decisão, iremos detalhar nesta seção como o C4.5 avalia a qualidade dos atributos que são candidatos a teste durante a indução. Embora existam outras funções que possam ser usadas para avaliar a qualidade de um atributo, as mais comuns são o índice de Gini e o ganho de informação. O ganho de informação tem importância particular neste trabalho e, por isso, será detalhado.

Conforme dito, o C4.5 utiliza o ganho de informação para medir a qualidade dos atributos como candidatos a teste, quando trata problemas de classificação. O ganho de informação utiliza o conceito de *entropia*, sendo esta uma medida de quão organizado, ou puro, é um conjunto em relação ao conceito que se deseja capturar. Portanto, definiremos primeiro a entropia e na sequência o ganho de informação.

Para simplificar a explicação, suponha que tenhamos um problema de classificação binária, i.e., o atributo alvo pode assumir exatamente dois valores, por exemplo: positivo (pos) ou negativo (neg). Seja S um conjunto de exemplos de algum conceito a ser aprendido, a entropia de S em relação ao atributo alvo é dada pela equação 2.1, onde $p_{\oplus}$ é a fração de exemplos positivos em S e $p_{\ominus}$ é a fração de exemplos negativos em S. Adicionalmente, defini-se $0 \log 0 = 0$.

$$Entropia(S) = \sum_{i=1}^{|C|} -p_i \log_2 p_i \tag{2.1}$$

onde C é o conjunto das possíveis classes e p_i é a probabilidade da classe i ocorrer. No caso binário, por exemplo, poderíamos ter: $C = \{pos, neg\}$ e $|C| = 2$.

A figura 2.3 mostra o gráfico da entropia para o caso binário. Duas constatações notáveis são: 1) a entropia torna-se 0 caso todos os exemplos pertençam à mesma classe; e 2) a entropia é máxima, igual a 1 no caso binário, quando cada classe possuir número igual de exemplos.

Note que, pela equação 2.1 pode-se concluir que a entropia é limitada entre 0 e $\log_2 |C|$. Esta é um constatação importante para este trabalho conforme será visto na seção 3.2.6.

Um exemplo será construído baseado no problema *Jogar Tênis* para auxiliar o entendimento. Seja S os exemplos mostrados na tabela 2.1. É fácil notar que há 9 exemplos da classe *sim* e 5 exemplos da classe *não*. Portanto, o valor numérico da entropia deste conjunto é dado pela expressão 2.2:

$$Entropia(S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0,940 \tag{2.2}$$

Agora que o conceito de entropia foi definido e exemplificado, podemos definir o ganho de informação:

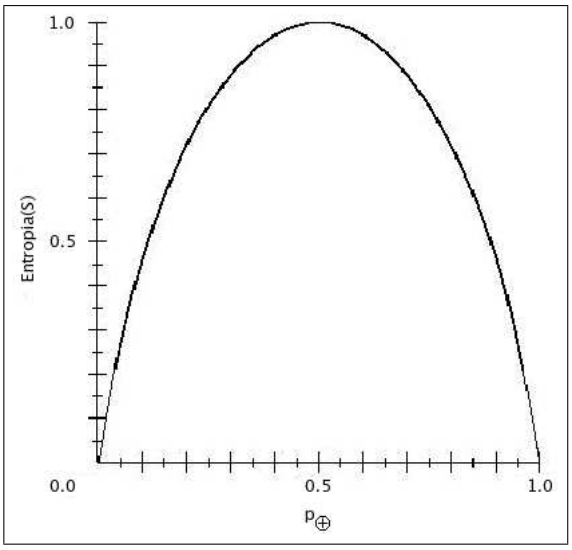


Figura 2.3: Gráfico da entropia relativa a uma classificação booleana. p_sim é a proporção de exemplos positivos e varia de 0 a 1.

O *Ganho de informação* quantifica a redução causada na entropia quando leva-se em conta a informação sobre um atributo [1]. Sejam: A um atributo presente nos exemplos do conjunto S; $dom(A)$ o conjunto dos valores que A pode assumir, i.e., seu domínio; e S_v o subconjunto dos elementos de S em que A possui valor v, com $v \in dom(A)$. Então, o ganho de informação do atributo A em relação ao conjunto S é dado pela equação 2.3.

$$Ganho(S, A) = Entropia(S) - \sum_{v \in dom(A)} \frac{|S_v|}{|S|} Entropia(S_v) \tag{2.3}$$

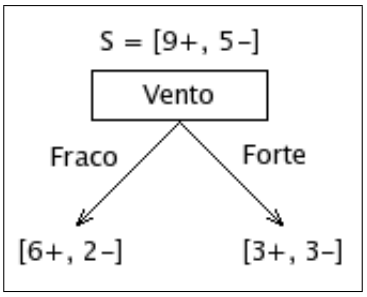
Continuando o exemplo do conceito *Jogar Tênis*, vamos calcular o ganho de informação para o atributo vento $Vento$, cujo domínio é $\{Fraco, Forte\}$.

Como visto anteriormente, S contém 9 exemplos positivos e 5 negativos (usaremos a notação $S = [9+, 5-]$). Destes 14 exemplos, 6 positivos e 2 negativos possuem $Vento = Fraco$; os demais têm $Vento = Forte$. A figura 2.4 mostra o cálculo do ganho de informação do atributo $Vento$ para este caso.

2.1.3 Aprendizado (Indução)

O C4.5 gera seus modelos a partir de um conjunto de dados, previamente observado, o qual traz informações empíricas sobre o conceito que o modelo deve captar. Tais dados são frequentemente chamados de *conjunto de treinamento* e seus elementos de *exemplos*. Utilizando-se de uma estratégia do tipo *dividir-para-conquistar*, o C4.5 particiona recursivamente os dados em subconjuntos progressivamente mais organizados (puros).

O aprendizado inicia com a pergunta: "Qual é o melhor atributo para ser usado



$Valores(Vento) = \{Fraco, Forte\}$
$S = [9+, 5-]$
$S_{Fraco} \leftarrow [6+, 2-]$
$S_{Forte} \leftarrow [3+, 3-]$

$Ganho(S, Vento) = Entropia(S) - \sum_{v \in \{Fraco, Forte\}} \frac{|S_v|}{|S|} Entropia(S_v)$
$= Entropia(S) - (8/14)Entropia(S_{Fraco}) - (6/14)Entropia(S_{Forte})$

$Entropia(S) = 0,940$ (calculada anteriormente)
$Entropia(S_{Fraco}) = -(6/8)\log_2(6/8) - (2/8)\log_2(2/8) = 0,811$
$Entropia(S_{Forte}) = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1,00$

Logo,
$Ganho(S, Vento) = 0,940 - (8/14)0,811 - (6/14)1,00 = 0,048$

Figura 2.4: Exemplo de cálculo de ganho de informação.

como teste na raiz?", tal pergunta é respondida avaliando-se cada atributo disponível através de alguma heurística, então o melhor atributo é selecionado como o teste para o nó raiz. Um descendente é criado para cada possível valor deste atributo e os exemplos são, então, repassados aos filhos de acordo com o valor que possuem em tal atributo. Este processo é repetido em cada descendente até que algum critério de parada seja satisfeito. Devido aos fatos de: o algoritmo nunca reconsiderar uma decisão prévia e sempre escolher o melhor atributo, pode-se dizer que adota uma estratégia gulosa [1].

Conforme dito na seção 2.1.2, existe mais de uma função que pode ser usada para avaliar a qualidade de um candidato a teste, porém estaremos usando o ganho de informação neste trabalho.

Devido a natureza recursiva do algoritmo, critérios de parada devem ser determinados. Estes podem ser estabelecidos de várias formas, como por exemplo: número mínimo de exemplos em uma folha, inexistência de mais atributos a serem usados como teste, todos exemplos da folha pertencem à mesma classe, acurácia mínima, dentre outros. É importante citar que o algoritmo pode verificar vários critérios de parada durante o aprendizado, não estando limitado a apenas um.

A função preditiva armazenada nas folhas, para problemas de classificação, é a

classe majoritária. Ou seja, quando uma folha classifica um exemplo, ela o atribui a classe que mais se repetiu nos exemplos que a alcançaram durante o processo de indução. Em caso de empate na raiz, pode-se predizer uma classe padrão, que deve ser previamente especificada, já nos nós internos, esta classe padrão será a mesma do nó ancestral imediato.

Uma técnica frequentemente usada para melhorar a qualidade dos modelos gerados é conhecida como *poda* e normalmente ocorre após a construção da árvore. A poda consiste num processo de remoção de nós, que parte das folhas em direção à raiz, ela tenta eliminar distinções potencialmente desnecessárias, isto é, que aumentam o modelo sem melhorar a qualidade preditiva.

É importante citar que modelos excessivamente grandes costumam gerar super-adaptação aos dados de treinamento (*overfitting*). Esta super-adaptação tem como efeito boas medidas de avaliação em relação ao conjunto de treino, mas costuma ter o efeito de reduzir a capacidade preditiva em novos objetos, i.e., casos não foram observados durante o aprendizado do modelo.

Para finalizar, mostraremos no algoritmo 2.1 o pseudo-código de uma versão simplificada do C4.5, com os procedimentos adotados durante o aprendizado de uma árvore de decisão.

Algoritmo 2.1 Versão do algoritmo C4.5 para aprender árvores de decisão para problemas de duas classes [1]

Entrada:

Exs é o conjunto de exemplos de treinamento,

AtribObjetivo é o atributo cujo valor a árvore deverá predizer.

Atribs é uma lista dos outros atributos, que poderão ser testes de nós internos da árvore.

Saída:

Uma árvore de decisão que classifica corretamente os exemplos de treinamento.

Procedimento C4.5 (*Exs*, *AtribObjetivo*, *Atribs*)

1: Crie *Raiz*, que será o nó raiz da árvore.
2: Se todos os exemplos de *Exs* forem positivos, então
3: Faça *Raiz.classe* = "+".
4: Retorne *Raiz*, que é o nó raiz da árvore de decisão (formada por apenas um nó).
5: Se todos os exemplos de *Exs* forem negativos, então
6: Faça *Raiz.classe* = "-".
7: Retorne *Raiz*.
8: Se *Atribs* for uma lista vazia, então
9: Faça *Raiz.classe* = valor de *AtribObjetivo* mais frequente entre os exemplos de *Exs* (ou seja, a classe majoritária).
10: Retorne *Raiz*.
11: Senão
12: Faça A ser o atributo de *Atribs* que melhor classifica os exemplos de *Exs* (ou seja, o atributo com o maior ganho de informação).
13: Faça *Raiz.atributo* = A.
14: Para cada possível valor v_i do atributo A
15: Adicione um novo ramo ao nó *Raiz*, correspondente ao teste $A = v_i$.
16: Faça Exs_{v_i} ser o subconjunto de *Exs* formado pelos exemplos que têm o atributo A com valor v_i.
17: Se o conjunto Exs_{v_i} for vazio, então
18: Crie uma nova folha F.
19: Faça *F.classe* = valor de *AtribObjetivo* mais frequente entre os exemplos de *Exs* (ou seja, a classe majoritária).
20: Adicione F ao ramo.
21: Senão
22: Crie uma subárvore *SubArv*.
23: Faça *SubArv* = C4.5(Exs_{v_i}, *AtribObjetivo*, *Atribs* - $\{A\}$).
24: Adicione *SubArv* ao ramo.
25: Retorne *Raiz*, que é o nó raiz da árvore de decisão.

2.2 Árvores de Regressão - O caso contínuo

O C4.5 também é capaz de resolver problemas de regressão. Tais problemas são caracterizados pelo fato de o conceito que se deseja aprender ser numérico em vez de categórico. Inicialmente, apresentaremos um exemplo para facilitar as explicações adiante.

Utilizaremos como exemplo o problema de predizer o consumo de combustível de um carro, a partir de outros atributos. Portanto, o atributo alvo será o atributo *MPG*, que representa o consumo de combustível de um carro em milhas por galão. O exemplo aqui mostrado é uma simplificação do problema *Miles-per-gallon*, um dataset de regressão disponível no repositório UCI [12]. Então, na tabela 2.2 são mostrados alguns elementos deste dataset. Conforme pode ser observado, a representação atributo-valor é usada da mesma forma que no dataset *Jogar Tênis*.

Carro	**Cilindros**	**Cilindradas** (in^3)	**Peso (lbs)**	**Acel.** (ft/s^2)	**MPG**
C1	8	429,0	4341,0	10,0	15,0
C2	8	454,0	4354,0	9,0	14,0
C3	8	455,0	4425,0	10,0	14,0
C4	8	390,0	3850,0	8,5	15,0
C5	8	383,0	3563,0	10,0	15,0
C6	4	140,0	2408,0	19,0	22,0
C7	6	250,0	3282,0	15,0	19,0
C8	6	250,0	3139,0	14,5	18,0
C9	3	70,0	2124,0	13,5	18,0
C10	3	80,0	2720,0	13,5	21,5
C11	4	90,0	1985,0	21,5	43,1
C12	4	98,0	1800,0	14,4	36,1
C13	5	131,0	2830,0	15,9	20,3
C14	8	400,0	5140,0	12,0	13,0

Tabela 2.2: Objetos associados ao conceito *Milhas por Galão*.

Duas metas desejáveis do processo de regressão são: 1) predizer o mais corretamente possível o atributo alvo (variável dependente) a partir dos demais atributos (variáveis independentes/preditivas) observados em um novo exemplo; e 2) explicar a relação existente entre as variáveis independentes e o atributo alvo [13].

Um fato importante a ser catitado é que as árvores de regressão preservam a interpretabilidade das árvores de decisão. Através da estrutura de árvore, que codifica regras, as relações entre as variáveis preditivas e a variável dependente podem ser humanamente interpretadas. Um exemplo de árvore de regressão para o problema *MPG* pode ser vista na figura 2.5:

Seja um carro com as seguintes características: $Cilindros = 4$, $Cilindradas = 121$, $Peso = 2933$ e $Aceleração = 14,5$. Então, de acordo com a árvore na figura

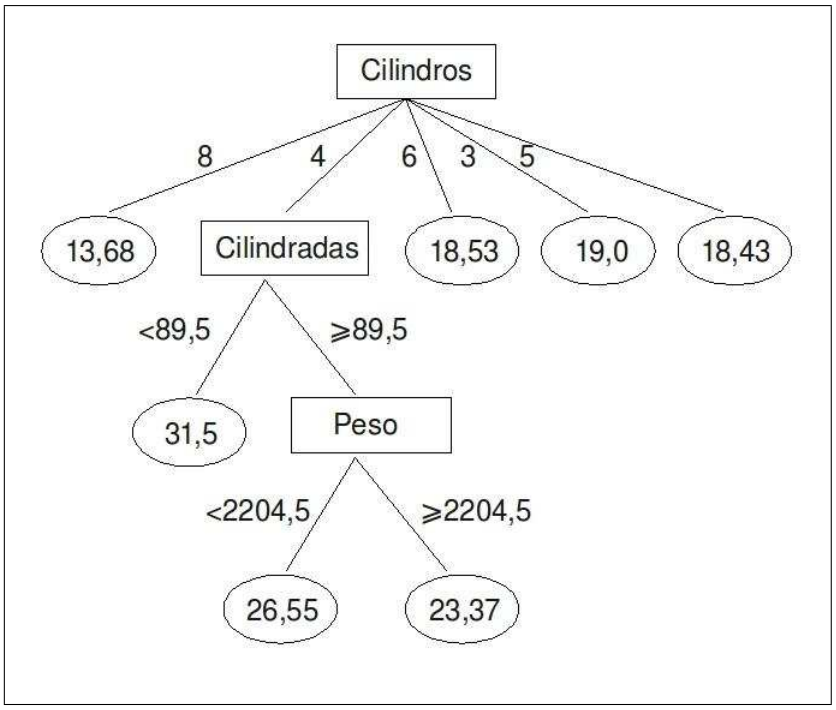


Figura 2.5: Exemplo de uma árvore de regressão para o conceito *MPG*.

2.5, a predição para o seu consumo seria de $23,37$ milhas por galão.

É importante notar que agora dois tipos de atributos estão envolvidos: cilindros, que é categórico; e cilindradas e peso, que são contínuos. O tratamento de atributos contínuos será melhor explicado na seção 2.2.2

Conforme veremos na seção 2.2.4, o processo de construção de árvores de regressão é bastante similar ao das árvores de classificação. Porém, alguns itens devem ser devidamente ajustados à natureza numérica do atributo alvo. Conforme veremos, são eles: 1) a heurística para avaliar a qualidade de um atributo como candidato a teste; 2) o modelo preditivo armazenado nas folhas; e 3) as métricas de qualidade do modelo gerado.

2.2.1 Redução da variância

Quando é usado para problemas de regressão, o C4.5 utiliza como heurística a redução da variância, embora possa usar outras heurísticas. Tais heurísticas visam quantificar o grau de organização dos dados, para que o processo de aprendizado, de forma análoga a classificação, seja capaz de progressivamente gerar partições mais puras/organizadas.

Da mesma forma que definimos entropia antes do ganho de informação, para definirmos a redução da variância primeiramente definiremos a variância, denotada por S^2 em [13], em um conjunto de exemplos qualquer S.

Então, seja S um conjunto de exemplos com um atributo alvo numérico Y, então

sua variância, $S^2(S)$, é definida por:

$$S^2(S) = \frac{1}{|S|}\sum_{i \in S}(y_i - \overline{y})^2 \tag{2.4}$$

onde y_i é o valor do atributo alvo no i-ésimo exemplo e $\overline{y}$ é o valor médio do atributo alvo observado nos exemplos de S, dado por:

$$\overline{y} = \frac{\sum_i^{|S|}(y_i)}{|S|} \tag{2.5}$$

É possível observar na equação 2.4 que a variância é uma medida de dispersão. Esta quantifica o quão organizado um conjunto numérico, composto pelos valores do atributo alvo neste caso, está em relação à sua própria média.

Para exemplificar, suponha que S seja o conjunto de exemplos exposto na tabela 2.2, então o valor numérico de $S^2(S)$ é dado por:

$$S^2(S) = \frac{(15,0 - 20,29)^2 + (14,0 - 20,29)^2 + ... + (13,0 - 20,29)^2}{14} = 71,64 \tag{2.6}$$

Seja A um atributo preditivo categórico, cujo domínio é $dom(A)$. Então, ao particionarmos os exemplos de um conjunto S com $dom(A)$, a redução da variância em S causada por S é dada por:

$$\phi(S, A) = S^2(S) - \sum_{v \in dom(S)} \frac{|S_v|}{|S|} S^2(S_v) \tag{2.7}$$

onde S_v é o subconjunto de exemplos S que tem valor v no atributo A. Note que, a variância de cada partição S_v é ponderada pela fração de exemplos em relação a S.

Continuando o exemplo, vamos computar a redução da variância de S causada pelo atributo $Cilindros$. Seja $dom(Cilindros) = \{3, 4, 5, 6, 8\}$, então:

$$\phi(S, Cilindros) = 71,64 - \frac{(2 * 3,06 + ... + 6 * 0,56)}{14} = 71,64 - 17,21 = 54,43 \tag{2.8}$$

2.2.2 Atributos numéricos

O C4.5 é capaz de tratar atributos numéricos e este processo é bastante direto. Dado um atributo contínuo A, o método consiste em realizar um split binário, baseado no teste $A < c$, onde c é um limiar constante, separando, portanto, os exemplos em dois subconjuntos: $\{x|x < c\}$ e $\{x|x \geq c\}$.

Para achar a constante c mais adequada para o split binário, o C4.5 primeiramente ordena os exemplos de acordo com os valores de A e usa os pontos médios como candidatos ao limiar de corte. Por exemplo, ao analisar o exemplo na tabela 2.2 os candidatos a limiar para o atributo cilindradas seriam: $\{75, 85, 94, 114.5, 135.5, 195, 250, 316.5, 386.5, 395, 414.5, 441.5, 454.5\}$. Em seguida, computa a heurística para cada limiar, levando em conta os subconjuntos gerados por cada corte, e escolhe-se o que tiver a melhor heurística. Este processo é feito para todo atributo numérico.

Para evitar o custo de repetidas ordenações, cada um com custo $O(nlog(n))$, em [14] é proposto o uso de listas de atributos. Ainda na raiz, antes do processo de indução começar, é construído uma lista ordenada para cada atributo numérico com os índices dos exemplos e uma lista adicional que relaciona cada exemplo a um nó da árvore. A medida que o particionamento dos dados se processa, as listas dos atributos de novos nós podem ser reconstruídas a partir da lista do nó ancestral, com complexidade $O(n)$, sem a necessidade de novas ordenações. Maiores informações sobre o uso das listas de atributos podem ser obtidas em [14].

É importante notar que este processo, que torna desnecessário outros pré-processamentos, só é possível devido ao fato de o C4.5 carregar todos os dados de uma só vez (aprendizado em batch).

Para exemplificar como o C4.5 efetivamente trata os atributos contínuos, a tabela 2.3 mostra o resultado do procedimento de escolha do ponto de corte aplicado ao atributo *Cilindradas*, usando os registros da tabela 2.2.

Carro	**Cilindradas**	**MPG**	**Cortes**	**Heurística**
C9	70	18	75	0,4
C10	80	21,5	85	0,05
C11	90	43,1	94	14,33
C12	98	36,1	114,5	35,26
C13	131	20,3	135,5	31,37
C6	140	22	195	32,15
C7	250	19	250	29,47
C8	250	18	316,5	26,57
C5	383	15	386,5	20,58
C4	390	15	395	15,8
C14	400	13	414,5	9,66
C1	429	15	441,5	6,59
C2	454	14	454,5	3,04
C3	455	14		

Tabela 2.3: Tratamento do atributo numérico Cilindradas.

Utilizando os valores da coluna *Cilindradas*, o algoritmo calcularia a redução da variância, coluna *Heurística*, causada pelo atributo cilindradas para cada possível

ponto médio, coluna *Cortes*. Este processo escolheria como constante $c = 114,5$. Caso o atributo cilindradas fosse o de melhor heurística dentre os demais, os dado seriam particionados em: $\{C9, C10, C11, C12\}$ e $\{C1, ..., C8, C13, C14\}$.

É importante observar que um atributo categórico abre uma aresta para cada valor, esgotando toda informação que pode fornecer, portanto somente é usado ao máximo uma vez. Já o numérico, por usar split binário, pode ser usado várias vezes, porém isso reduz a interpretabilidade da árvore. Uma alternativa seria tornar o uso do atributo contínuo exaustivo, i.e., semelhante aos dos categóricos.

Duas possibilidades seriam: usar splits multiponto (multi-way splits) ou pré-processamento de discretização. Por exemplo, o atributo cilindradas poderia ser discretizado conforme: $[70, 140) = baixa$, $[140, 250.0) = média$, $[250, 455) = alta$.

Em [15] é proposto o uso de árvores binárias de busca para cada atributo contínuo candidato a teste em uma folha. Cada vez que um exemplo chega a uma folha, a árvore de cada atributo contínuo neste folha é atualizada. Para escolha do melhor candidato, analogamente, ordenam-se os exemplos e as próprias chaves da árvore são usadas como pontos de corte. O ponto de corte com a melhor heurística é então escolhido como teste.

2.2.3 Modelo preditivo

O modelo usado pelo C4.5 em suas folhas é a média dos valores do atributo alvo dos exemplos. Logo, conforme o exemplo *MPG*, se a árvore sofresse um split em $Cilindradas < 114,5$ e parasse de crescer, então a folha da esquerda iria associar o valor $29,68$ a novos exemplos, e a folha da direita $16,53$.

Devido ao modelo preditivo e à natureza da própria heurística usada, o C4.5 precisa capturar informações que ajudem a obter a média e variância de forma eficiente, tanto para a folha quando para os atributos. Então, as seguintes estatísticas são computadas, para cada atributo e para os exemplos da folha como um todo: número de exemplos $|S|$, soma dos valores $\sum_{i \in S} y_i$ e soma dos valores quadrados $\sum_{i \in S} y_i^2$ para o atributo alvo. Assim, a equação da variância, que usa apenas uma passada nos exemplos, pode ser usada conforme:

$$S^2(S) = \frac{\sum_{i \in S} y_i^2}{|S|} - \big(\frac{\sum_{i \in S} y_i}{|S|}\big)^2 \tag{2.9}$$

Como as três estatísticas são coletadas para a folha e parada cada atributo candidato a teste, se o processo de indução parar, basta descartar as estatísticas dos atributos e usar as estatísticas da folha para fornecer a média, através de $|S|$ e $\sum_{i \in S} y_i$.

Outras estatísticas de resumo podem ser usadas na predição como, por exemplo, moda e mediana. É possível ainda que as folhas usem outros modelos como as

funções lineares das model trees [16] [17] ou perceptrons [18].

2.2.4 Indução de árvores de regressão

De forma análoga à classificação, os modelos de regressão são construídos através de um particionamento recursivo dos dados de treinamento, de modo a obter subconjuntos progressivamente mais organizados/puros em relação ao atributo alvo que se deseja aprender.

Conforme já mencionado, o algoritmo tem diferenciações em relação a heurística usada e os modelos armazenados nas folhas. Além disso, as métricas de avaliação para modelos de regressão devem ser adequadas a este contexto. As métricas usadas neste trabalho serão mostradas na seção 2.4.

Os critérios de parada podem ser variados, como número mínimo de exemplos numa folha, altura máxima da árvore, valor mínimo para alguma medida de qualidade de um nó. Em geral, os critérios de parada são passados ao algoritmo através de parâmetros e de forma adaptada à implementação sendo usada.

Uma importante melhoria do C4.5 em relação ao ID3 é a capacidade de tratar atributos numéricos usando o método explicado na seção 2.2.2, isto fica evidente no pseudo-código exibido no algoritmo 2.2 (linhas 12 e 23). Segue, então, o pseudo código que mostra os principais procedimentos adotados pelo C4.5 no processo de regressão no algoritmo 2.2.

Algoritmo 2.2 Versão do algoritmo C4.5 para aprender árvores de regressão

Entrada:

Exs é o conjunto de exemplos de treinamento,

AtribAlvo é o atributo cujo valor a árvore deverá predizer.

Atribs é uma lista dos outros atributos, que poderão ser testes de nós internos da árvore.

Saída:

Uma árvore de regressão que resume corretamente os exemplos de treinamento.

Procedimento C4.5 (*Exs*, *AtribAlvo*, *Atribs*)

1: Crie *Raiz*, que será o nó raiz da árvore.
2: Se algum critério de parada foi alcançado, então
3: Faça Raiz virar uma folha
4: seja $\overline{y}$ o valor médio de *AtribAlvo* em *Exs*
5: Raiz.predição = $\overline{y}$
6: retorne Raiz
7: Senão
8: Para cada atributo em *Atribs* computar a heurística usando *Exs*
9: Seja *A* o atributo de *Atribs* que causa a maior redução da variância
10: Faça Raiz.atributo = *A*
11:
12: Se *A* for contínuo, então
13: Seja *c* a constante obtida para o teste de *A*
14: Adicione dois novos ramos, *Esq* e *Dir*, à *Raiz*,
15: Seja Exs_{esq} os exemplos de *Exs* que passam no teste $A < c$
16: Faça $SubArvEsq$ = C4.5(Exs_{esq}, *AtribAlvo*, *Atribs*).
17: Adicione $SubArvEsq$ ao ramo *Esq*.
18:
19: Seja Exs_{dir} os exemplos de *Exs* que passam no teste $A \geq c$
20: Faça $SubArvDir$ = C4.5(Exs_{dir}, *AtribAlvo*, *Atribs*).
21: Adicione $SubArvDir$ ao ramo *Dir*.
22:
23: Senão (*A* é categórico)
24: Para cada possível valor v_i do atributo *A*
25: Adicione um novo ramo ao nó *Raiz*, correspondente ao teste $A = v_i$.
26: Faça Exs_{v_i} os exemplos de *Exs* com o atributo $A = v_i$.
27: Crie uma subárvore *SubArv*.
28: Faça $SubArv$ = C4.5(Exs_{v_i}, *AtribAlvo*, $(Atribs \setminus \{A\})$).
29: Adicione *SubArv* ao ramo.
30:
31: Retorne *Raiz*, que é o nó raiz da árvore de decisão.

2.3 Programação em Lógica Indutiva - ILP

Programação em Lógica Indutiva (ILP) é definida como a intersecção entre o aprendizado de máquina indutivo e a programação lógica [19] [20] [21] [22]. Do aprendizado de máquina indutivo, herda o objetivo: aprender hipóteses a partir de observações (exemplos) e sintetizar novo conhecimento a partir da experiência. Através do uso da lógica computacional como mecanismo de representação das hipóteses e observações, ILP pode superar duas grandes limitações das técnicas clássicas de aprendizado de máquina: 1) o uso de uma representação de conhecimento limitada (essencialmente lógica proposicional); e 2) a dificuldade de usar conhecimento preliminar substancial no processo de aprendizado.

A tarefa mais fundamental é o aprendizado de uma relação, chamada de *predicado alvo*, a partir das observações (exemplos) e de relações definidas no conhecimento preliminar e, possivelmente, através do próprio predicado alvo (relações recursivas).

Na figura 2.6 está exposto um esquema de aprendizado em ILP. As observações E são divididas em duas partições: E^+ e E^-, respectivamente aquelas que pertencem à relação alvo, exemplos positivos, e àquelas que não pertencem, exemplos negativos. O conhecimento preliminar B representando demais relações e H é a hipótese, em lógica de primeira ordem, que deseja-se aprender. Ao final do processo de aprendizado, deseja-se que $H \cup B$ explique todos os exemplos positivos e nenhum exemplo negativo.

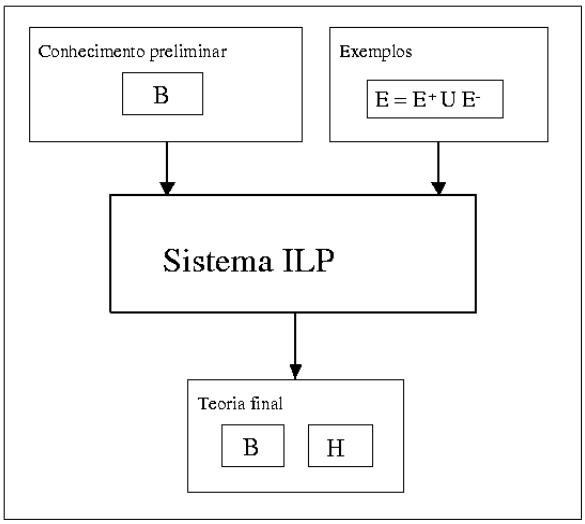


Figura 2.6: Esquema do aprendizado em programação em lógica indutiva (*ILP*), onde B é o conhecimento preliminar, E é o conjunto de exemplos, formado pelos exemplos positivos (E^+) e negativos (E^-), e H é a teoria aprendida pelo sistema.

O processo de aprendizado pode ser formalizado de mais de uma maneira [23], porém a definição mais tradicional usada em ILP é a chamada de *Aprendizado por Implicação Lógica* elaborada conforme a definição 2.1.

Definição 2.1 (Aprendizado a partir de implicação lógica)**.** *Sejam dados:*

- *Um conjunto E^+ de exemplos positivos e um conjunto E^- de exemplos negativos.*
- *Uma teoria preliminar B.*

Deseja-se achar:

- *Uma hipótese H, tal que*
  - $\forall e \in E^+ : H \wedge B \vDash e, \qquad e$
  - $\forall e \in E^- : H \wedge B \nvDash e$

Para exemplificar e consolidar a definição, o exemplo dos trens será descrito. Este é um problema clássico de ILP em que deseja-se separar um conjunto de trens em dois casos: os que vão para leste e os que vão para oeste. Cada trem é composto de um ou mais vagões, e cada vagão tem um conjunto de propriedades como: curto ou longo, aberto ou fechado, carrega objetos ou não, dentre outras. Na figura 2.7 há 10 exemplos de trens, dos quais 5 vão para leste e 5 vão para oeste.

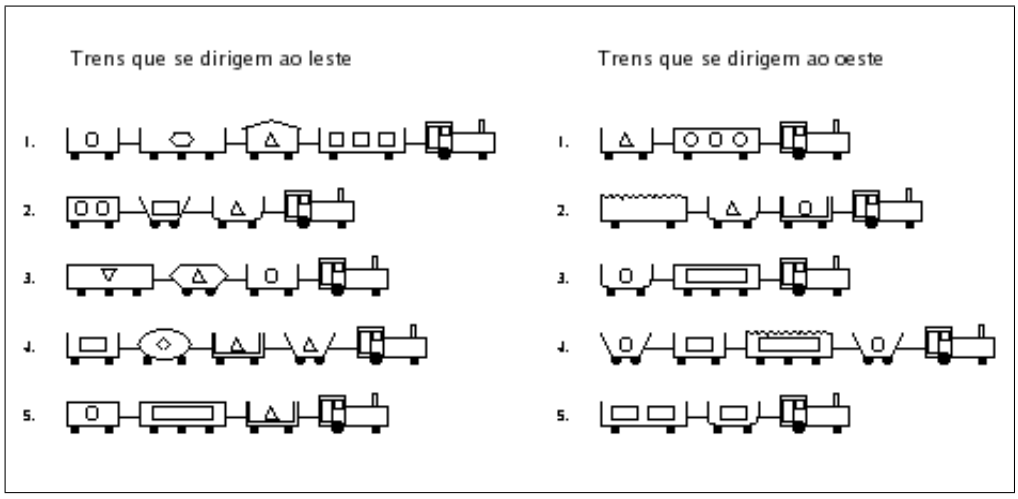


Figura 2.7: Exemplos considerados no problema dos trens.

O problema pode ser representado em ILP da seguinte maneira: o predicado alvo *direção_leste(T)* será *verdadeiro* se o trem T dirigir-se a *leste* e falso se o trem T dirigir-se a *oeste*, portanto os exemplos de 1 a 5 são os *positivos* e os demais *negativos*, conforme a figura 2.3.

Parte do conhecimento preliminar B, pode ser, então, representado conforme a figura 2.3.

Espera-se que um algoritmo ILP seja capaz de aprender uma regra como: *direção leste(A) :- tem vagão(A,B), curto(B), fechado(B).* que juntamente com o conhecimento preliminar é capaz de provar os exemplos de 1 a 5 sem provar os exemplos de 6 a 10.

Exemplos positivos:	**Exemplos negativos:**
direção_leste(leste1).	direção_leste(oeste1).
direção_leste(leste2).	direção_leste(oeste2).
direção_leste(leste3).	direção_leste(oeste3).
direção_leste(leste4).	direção_leste(oeste4).
direção_leste(leste5).	direção_leste(oeste5).
tem_vagão(leste1,vagão_11).	tem_vagao(leste1,vagão_12).
tem_vagão(leste1,vagão_13).	tem_vagão(leste1,vagão_14).
longo(vagão_11).	curto(vagão_12).
aberto(vagão_11).	fechado(vagão_12).
carrega(vagão_11,retângulo,3).	carrega(vagão_12,triângulo,1).
...	

Note que, por uma característica particular deste problema, podemos concluir que se um trem não for coberto pelo predicado *direção_leste(A)* então ele vai para oeste. Porém, caso tivéssemos outras possibilidades como: leste, oeste, norte e sul; bastaria aprender uma regra para cada conceito, de modo que um exemplo provado por uma teoria pertenceria ao conceito codificado por esta e os demais seriam considerados negativos.

É importante destacar que a álgebra relacional, empregada nos bancos dados modernos, pode ser entendida como um subconjunto da lógica de primeira ordem, portanto é possível modelar dados relacionais sem perdas através de LPO. Além disso, deve-se destacar que domínios relacionais são esparsos, ou seja, o número de instancias realmente observadas de cada relação é tipicamente muito menor que a quantidade de todas as possibilidades de instâncias. A ILP consegue explorar tal característica, codificando em poucas cláusulas, relações que no paradigma proposicional necessitariam de uma grande listagem de casos. Por exemplo:

ancestral(A,C) :- ancestral(A,B),ancestral(B,C).
ancestral(o1,o10).
ancestral(o1,o11).
...
ancestral(o10,o100).
...
ancestral(o10,o199).

No paradigma proposicional, para indicarmos os fatos de que o1 é ancestral dos objetos que vão de o100 a o199 precisaríamos de muito mais entradas relacionando o1 diretamente àqueles.

2.3.1 Regressão em ILP

Embora existam outras abordagens que buscam mesclar o tratamento de regressão com ILP, a abordagem usada neste trabalho é mais próxima a formalização do sistema *FORS*, elaborada em [24].

Ao formular um problema de regressão em ILP nesta abordagem, o predicado alvo deve receber argumentos numéricos, e tais argumentos são chamados de *atributos contínuos*, X_i. Adicionalmente, existe um argumento numérico especial, Y, que representa a variável dependente que será aprendida no processo de regressão, é comum que esta variável especial seja chamada de atributo alvo. Sendo assim, Y só pode ser usada no lado esquerdo de equações que apareçam nas cláusulas da teoria a ser aprendida, e os atributos contínuos no lado direito. Portanto, o conjunto de cláusulas aprendidas sobre o predicado alvo tem o formato conforme mostrado na figura 2.8.

```
f( Y, X1, X2, ...) :-
    Literal1, Literal2, ...,
    Y is a1 * Xi1 + a2 * Xi2 + ..., !.
```

Figura 2.8: Exemplo de cláusula aprendida pelo FORS

Embora o sistema FORS realize regressão linear usando os atributos contínuos do predicado alvo, esta abordagem não impõe limitações sobre os tipos de equações aprendidas para Y, além disso, o conhecimento preliminar pode introduzir novas variáveis numéricas que pode ser usadas nas equações aprendidas.

O FORS, da mesma forma que o FOIL, usa aprendizado por implicação lógica. Desta forma, os exemplos são fatos sobre o predicado alvo e o conhecimento preliminar é representado através de cláusulas Prolog. O resultado do aprendizado é uma teoria lógica composta por cláusulas. O sistema FORS é capaz de aprender predicados recursivos, controlando a profundidade da recursão através de uma profundidade máxima estabelecida como parâmetro.

Cada cláusula da teoria aprendida particiona o espaço de exemplos, através de suas regras lógicas, associando cada sub-espaço à equação de regressão da cláusula. Dois fatos importantes a serem notados são: 1) Não existe o conceito de exemplos negativos, pois os dados de treinamento são considerados como "medições"da variável dependente e, portanto, todos os exemplos são vistos como "positivos"; 2) A qualidade de uma regra aprendida está relacionada ao grau de similaridade existente a sua equação de regressão e aos exemplos cobertos pela cláusula.

Para finalizar vamos mostrar como exemplo o aprendizado da função fatorial na figura 2.9. Nela é mostrado a especificação do predicado alvo, um conjunto de exemplos, conhecimento preliminar e a saída obtida pelo sistema.

Entradas:

% Indicação do predicado alvo
$target(f(-integer,\ +integer)).$

% Exemplos
$f(1,0).\ f(1,1).\ f(2,2).\ f(6,3).\ f(24,4).$

% Conhecimento preliminar
$bkl(decr(-integer,\ +integer),\ total).$
$bkl(times(-integer,\ +integer,\ +integer),\ total).$
$decr(N1,\ N)$:- $N1\ is\ N-1.$
$times(R,\ A,\ B)$:- $R\ is\ A*B.$

Teoria aprendida:

$f(F,N)$:- $N \leq 1,\ F\ is\ 1, !.$
$f(F,N)$:- $decr(A,N),\ f(B,A),\ times(F,N,B), !.$

Figura 2.9: Exemplo do problema fatorial

O sistema FORS faz restrições sobre o espaço de busca através do uso de tipos e modos das variáveis, cujo uso pode ser observado na figura 2.9. Os tipos definem o domínio das variáveis restringindo a unificação. Os modos indicam se uma variável é de entrada (-), saída (+) ou os dois (+-).

2.4 Avaliação do Aprendizado

No cenário de regressão não há classes, mas um atributo alvo numérico, logo as medidas tradicionalmente usadas em classificação, como acurácia, medida f, precisão, dentre outras, não podem ser usadas. Portanto, métricas mais adequadas devem usadas. Nesta seção, estão definidas as medidas de avaliação de performance usadas neste trabalho.

Comumente em regressão, usam-se medidas capazes de indicar o quão próxima a predição de um modelo está em relação ao valor real de um exemplo. Adicionalmente, no contexto de árvores de regressão, é também importante medir o tamanho do modelo gerado, bem como o tempo do aprendizado.

Desta forma, as medidas de interesse usadas neste trabalho foram: Tempo de aprendizado (Tempo), tamanho das teorias ou número de nós na árvore (Nós), número total de literais (Literais), coeficiente de correlação de Pearson (r na equação 2.10) entre os valores reais e os preditos do predicado alvo e raiz quadrada do erro

quadrático médio (RMSE), equação 2.11, conforme [8].

$$r = \frac{\sum_{i=1}^{n}(p_i - \overline{p})(a_i - \overline{a})}{\sqrt{\sum_{i=1}^{n}(p_i - \overline{p})^2 \sum_{i=1}^{n}(a_i - \overline{a})^2}} \tag{2.10}$$

$$MSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}} \tag{2.11}$$

Onde, p_i é o valor predito para o exemplo i e a_i seu valor real. $\overline{p}$ e $\overline{a}$ são as médias e n o número de exemplos.

O coeficiente Pearson é delimitado entre em $[1, -1]$ e indica a existência de relação linear entre o modelo aprendido e o conceito que se deseja captar, resultados próximos de 1 indicam relação linear direta, próximos de -1 indicam relação inversa, e próximos de 0 indicam ausência de relação linear. Já o RMSE tem valor mínimo em 0 e máximo indefinido, ele tem seu melhor resultado em 0, que significa que o modelo conseguiu atribuir o valor exato a cada exemplo. Esta medida cresce conforme o modelo realiza predições que desviam do verdadeiro valor dos exemplos.

Validação cruzada é um método que ajuda a avaliar o quão bem um algoritmo consegue generalizar um conceito aprendido. Para tal, sorteia-se os conjuntos de treino e de teste de forma sistemática e repete-se o processo de aprendizado e medição de erro para cada modelo aprendido. A média de todos os resultados obtidos é usada como um estimador para medida específica, por exemplo o MSE. Espera-se que a estabilidade do resultado seja maior conforme o número de repetições aumenta.

Neste trabalho, usamos a validação cruzada do tipo 5x2 [25]. O conjunto de exemplos, relativos a um conceito que se deseja aprender, é divido em 5 subconjuntos, que chamaremos de folds. Cada fold, de f_1 a f_5, tem aproximadamente o mesmo número de exemplos e é subdividido em 2 subfolds, f_{ia} e f_{ib}, $i \in [1, 5]$. Em cada fold f_i, realiza-se o aprendizado usando f_{ia} como teste e f_{ib} como treino e depois o inverso. Ao final colhe-se os resultados de cada um dos 10 classificadores e obtém-se o valor médio da medida de interesse.

Como neste trabalho comparamos dois algoritmos, HTILDE-RT e TILDE-RT, sendo o primeiro uma modificação do segundo, executamos o teste t corrigido e pareado, [26] [27], para verificar a significância estatística da diferença entre os resultados obtidos por cada algoritmo. Tal teste é uma adaptação do teste t de Student que leva em consideração a razão entre número de exemplos usado para treino e teste $\frac{n_2}{n_1}$ e a variância da diferença média obtida nos folds. O valor crítico para t é então definido conforme a equação 2.12.

$$t = \frac{\overline{d}}{\sqrt{(\frac{1}{k} + \frac{n_2}{n_1})\sigma_d^2}} \tag{2.12}$$

onde: $\overline{d}$ é a média das diferenças pareadas entre os algoritmos para cada fold, k é o número total de folds/subfolds usado, n_1 e n_2 é o número de exemplos usados respectivamente no treino e teste, e σ_d^2 a variância das diferenças.

Capítulo 3

Sistemas Relacionados

Neste capítulo, serão apresentados sistemas que serviram de base para este trabalho conforme a seguinte organização: na seção 3.1, introduziremos o sistema de programação em lógica indutiva TILDE, bem como analisaremos elementos pertinentes a este trabalho; na seção 3.2, será apresentado o sistema proposicional de classificação VFDT, escalável para aprendizado em grandes massas de dados; e, na seção 3.3, apresentaremos o sistema HTILDE que combina a capacidade lógica do sistema TILDE com a eficiência do sistema VFDT.

3.1 TILDE - Top-down Induction of Logical Decision Trees

Nesta seção, será apresentado o sistema TILDE (Top-down Induction of Logical Decision Trees) [3] e [2], bem como sua aplicação em problemas de regressão. Na seção 3.1.1, será feita uma introdução ao método. Na seção 3.1.2, explicaremos como um problema e representado no sistema e apresentaremos a definição de aprendizado por interpretações, que é um conceito fundamental para o TILDE. Na seção 3.1.3, mostraremos como especificar um problema a ser resolvido pelo TILDE. Na seção 3.1.4, falaremos sobre o conceito de *refinamentos* da ILP, usado pelo TILDE. Na seção 3.1.5, falaremos sobre a técnica *pacote de cláusulas* (Query Packs), que melhora a eficiência do TILDE. Na seção 3.1.6, apresentaremos como TILDE trata problemas de regressão. Na seção 3.1.7, mostraremos a heurística usada pelo TILDE em modo de regressão. Finalizaremos este capítulo mostrando o algoritmo de aprendizado na seção 3.1.8.

3.1.1 Introdução

O algoritmo TILDE é uma extensão do C4.5 para o paradigma da lógica de primeira ordem. As principais diferenças a serem destacadas nesta abordagem são: a estrutura binária da árvore gerada, a escolha dos testes através do conceito de *refinamento*, a cobertura dos exemplos e o uso dos Pacotes de Cláusulas (Query Packs) para melhoria da eficiência. O TILDE também é capaz de resolver problemas de regressão e quando opera neste modo recebe o nome de TILDE-RT (TILDE - Regression Tree).

Nas árvores geradas pelo TILDE, cada nó interno é associado a uma conjunção de literais, de primeira ordem. Os testes nos nós internos não mais são realizados confrontando o valor de um atributo, mas através de testes lógicos representados por consultas Prolog. Os modelos gerados pelo TILDE são chamados de árvores de decisão de lógica de primeira ordem (first order logical decision trees - FOLDT) ou simplesmente árvores de decisão lógicas (Logical Decision Trees - LDT).

Seja N um nó interno qualquer, então o teste associado ao mesmo é representado por $\leftarrow Q \wedge conj$. A componente Q é chamada de *consulta associada* de N e a componente $conj$ é a conjunção de literais que fica armazenada em N. Uso do símbolo $\leftarrow$ é usado apenas para denotar que a conjunção será executada sob a forma de uma consulta Prolog.

A *consulta associada* é composta iterativamente da seguinte forma: a consulta associada do filho esquerdo de um nó qualquer é composta de sua consulta associada e sua conjunção: no caso de N, $\leftarrow Q \wedge conj$ seria a consulta associada de seu filho esquerdo; a consulta associada do filho direito não recebe a conjunção armazenada no nó, sendo portanto igual a consulta associada de seu pai: no caso de N a consulta associada de seu filho direito seria apenas $\leftarrow Q$, pois neste ramo a parte $conj$ falha. Este processo começa na raiz, cuja consulta associada é $Q = true$.

Em função de usar tais testes lógicos, as árvores geradas são binárias, de modo que: o ramo esquerdo de N é associado ao sucesso no teste $Q \wedge conj$ e o ramo direito associado à falha.

Antes de exemplificar uma árvore gerada pelo TILDE, introduziremos o problema BONGARD [28]. Neste problema deseja-se classificar figuras como *positivas* ($\oplus$) ou *negativas* ($\ominus$). Em cada figura, há uma quantidade variada de objetos relacionados entre si através de suas posições. A figura 3.1 ilustra 4 exemplos positivos e 4 exemplos negativos do problema BONGARD.

Na seção 3.1.2 discutiremos como representar este problema para que o TILDE possa tratá-lo. Na figura 3.2 há um exemplo de uma árvore gerada pelo TILDE para o BONGARD.

Segundo a árvore na figura 3.2, se uma figura não contiver um triângulo será

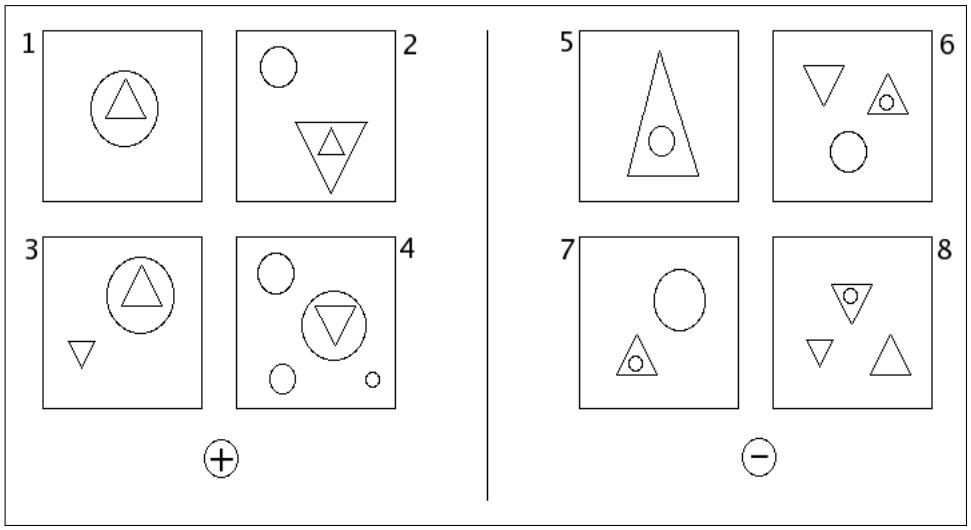


Figura 3.1: Exemplos do problema Bongard.

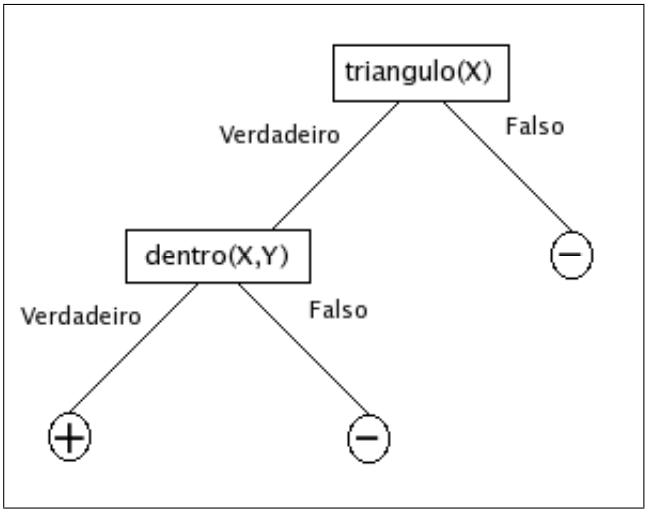


Figura 3.2: Exemplo de árvore gerada pelo TILDE.

classificada como (⊖), seguindo pelo ramo *Falso* da raiz; se houver um triângulo que esteja dentro de outro objeto, então será considerada (⊕); e se houver um triângulo que não esteja dentro de outro objeto, será considerada (⊖).

Devido ao paradigma da lógica de primeira ordem, as árvores geradas pelo TILDE se equivalem a cláusulas de Horn, cujos corpos são compostos pela conjunção dos literais que aparecem nos caminhos da raiz até suas folhas. O programa Prolog equivalente a árvore mostrada na figura 3.1.1 pode ser escrito como:

```
classe(pos) :- triangulo(X), dentro(X,Y), !.
classe(neg) :- triangulo(X), !.
classe(neg).
```

Em função dos testes lógicos, durante o aprendizado, variáveis introduzidas em um nó não são repassadas ao ramo direito deste nó, pois este ramo é associado aos exemplos que falham em seu teste. Isto se reflete no uso do *cut* (!), que mantém as

cláusulas do programa Prolog equivalente mutuamente exclusivas.

3.1.2 Aprendizado por Interpretação

No paradigma proposicional (sistemas atributo-valor) há apenas uma relação (tabela única), cujo conjunto de atributos (colunas) é fixo e cada linha equivale a um exemplo. Por exemplo, com 5 atributos poderíamos ter as seguintes informações para cada exemplo: {*forma-objeto1, aponta-objeto1, forma-objeto2, aponta-objeto2, classe*}, onde forma ∈ {*triângulo, círculo*} e aponta ∈ {*cima, baixo, esquerda, direita*}.

Entretanto, os exemplos na figura 3.1 podem ter quantidades arbitrárias de objetos, além disso, os objetos podem estar relacionados, por exemplo, um dentro do outro. Logo, não seria possível representar o problema BONGARD em apenas uma tabela única.

Algoritmos ILP superam esta limitação, entretanto a expressividade da ILP tem um custo associado, pois em função de usar o *Aprendizado por Implicação Lógica* a cobertura de cada exemplo é feita de forma global, o que significa que o teste lógico para verificar a cobertura de um único exemplo leva em conta todo conhecimento preliminar juntamente com todos os outros exemplos da base de dados. Na cobertura global existe a suposição de que os exemplos podem estar relacionados entre si. Isto não acontece no paradigma proposicional, onde cada exemplo pode ser testado individualmente, portanto de forma muito mais rápida.

Para melhorar a eficiência da cobertura dos exemplos [2], [23], [29], o TILDE usa o *Aprendizado por Interpretação*, onde cada exemplo é representado por um conjunto de fatos, i.e., uma *Interpretação*, conforme a definição 3.1.

Definição 3.1 (Aprendizado por Interpretação)**.** *Sejam dados:*

- *Um conjunto de classes C (cada classe c é um predicado sem termos).*
- *Um conjunto de exemplos E classificados (cada elemento de E é da forma (e,c), onde e é um conjunto de fatos e c é uma classe).*
- *Uma teoria preliminar B.*

Deseja-se achar:

- *Uma hipótese H (um programa Prolog), tal que para todo (e,c) ∈ E,*
  - $H \wedge e \wedge B \vDash c, \qquad e$
  - $\forall c' \in C$ - {c}: $H \wedge e \wedge B \nvDash c'$

O *Aprendizado por Interpretação* é um meio termo entre a representação atributo-valor [30] e a representação usada no *Aprendizado por Implicação Lógica*, pois possibilita cobertura local e maior capacidade de representação que o proposicional. Embora não seja capaz de tratar todos os problemas que o *Aprendizado por Implicação Lógica*, como, por exemplo, problemas recursivos, o *Aprendizado por Interpretação* é suficiente para a maioria das aplicações práticas [2].

Em [31], é discutida a diferença, sob o ponto de vista de bancos de dados relacionais, entre o aprendizado proposicional e o aprendizado por interpretações. Uma diferença fundamental apontada é que o aprendizado proposicional representa cada exemplo com uma única tupla de uma única tabela (relação), enquanto no outro cada exemplo pode conter múltiplas tuplas de múltiplas relações e permite o uso de conhecimento preliminar (background knowledge).

Contudo, na prática, dados são armazenados em bancos de dados relacionais, mas sistemas ILP usam dados representados através de lógica, então, em [31], é provido um algoritmo para converter um banco de dados relacional em um dataset usando o formato de interpretações. Uma noção fundamental usada neste algoritmo de conversão é a de que: toda informação pertinente a um certo exemplo está contida em uma pequena parte do banco e que esta informação pode ser localizada e extraída. Esta noção é chamada de *suposição de localidade* e quando não for satisfeita o processo de conversão pode gerar duplicação de informação.

Conforme discutido em [32], o uso de amostragem diretamente em dados relacionais pode gerar estimadores enviesados, porém, uma vez que o dataset esteja convertido (ou construído) no formato de interpretações, a suposição de localidade torna-se válida. Cada interpretação contém toda informação necessária sobre o exemplo, não dependendo dos demais. Logo, sortear indivíduos de forma aleatória e independente é suficiente para gerar boas amostras [2] [31]. Maiores informações sobre os diferentes tipos de aprendizado podem ser encontradas em [23].

3.1.3 Especificação de um problema

O TILDE armazena os exemplos num arquivo de extensão ".kb"(knowledge base), cujo nome é o mesmo do problema que se deseja aprender. Cada exemplo é considerado um modelo e tem um identificador (*id*) associado. Desta forma, os fatos sobre um exemplo são delimitados por um bloco iniciado com o fato "*begin(model(id)).*"e terminado pelo fato "*end(model(id)).*". Embora um exemplo possa ser descrito por um número arbitrário de fatos, cada exemplo fica associado a apenas uma classe, portanto apenas um fato, por exemplo, "*pos*"ou "*neg*", deve ser usado em cada exemplo. Para os exemplos da figura 3.1 o arquivo "bongard.kb", poderia ter o seguinte conteúdo:

```
begin(model(1)).          begin(model(6)).
pos.                      neg.
circulo(o1).              circulo(o16).
triangulo(o2).            circulo(o17).
aponta(o2,cima).          triangulo(o18).
dentro(o2,o1).            triangulo(o19).
end(model(1)).            aponta(o18,cima).
                          aponta(o19,baixo).
begin(model(2)).          dentro(o16,o18).
pos.                      end(model(6)).
...                       ...
```

Figura 3.3: Conteúdo do arquivo bongard.kb.

O conhecimento preliminar é fornecido em um arquivo opcional, de extensão ".bg"(background), cujo nome é o mesmo do problema em questão. Tal arquivo pode conter cláusulas auxiliares e fatos adicionais sobre o domínio do problema. Para o caso do BONGARD, o arquivo "bongard.bg"poderia conter a informação de que apenas triângulos têm a propriedade de apontar.

```
aponta(X) :- triangulo(X).
```

Figura 3.4: Conhecimento preliminar no arquivo bongard.bg.

Para testar a cobertura de um exemplo por uma fórmula, basta acrescentar os fatos do exemplo à base de conhecimento juntamente com o conhecimento preliminar, por exemplo, através do predicado "assert/1", e verificar se o conhecimento resultante é capaz de satisfazer a fórmula em questão. Em outras palavras, desejamos verificar se uma consulta Prolog $\leftarrow C$ é verdadeira em $e \wedge B$, sendo B o conhecimento preliminar e $\leftarrow$ apenas uma notação utilizada para explicitar que C é uma consulta.

Juntamente com o conhecimento preliminar e a base de dados, um arquivo de configurações, ".s"(settings), deve ser fornecido. Neste arquivo devem estar especificados diversos parâmetros do sistema, pertinentes ao problema que se deseja tratar, e o viés de linguagem, parte que instrui ao TILDE como os refinamentos podem ser determinados ao longo do processo de indução.

Ainda nas configurações devemos dizer qual o conceito que se deseja aprender. Isto pode ser feito com as seguintes construções: 1) Avisar o sistema qual o tipo de problema: *tilde_mode(tipo).*, onde "tipo"pode ser classify, regression ou cluster; 2) Fornecer o predicado alvo atributo e a variável dependente: *predict(predicadoAlvo(+X1,+X2,...,-Y)).*, onde Y, é uma variável de saída cujo domínio são as possíveis classes; 3) Fornecer a lista de possíveis classes: *classes([classe1,*

classe2, ...])..

Os corpos das cláusulas são construídos no processo de indução através dos refinamentos, que seguem um viés de linguagem determinado pelo usuário, basicamente através do conceito de "modes". Na seção 3.1.4 veremos mais detalhes sobre os refinamentos.

3.1.4 Refinamentos

Conforme dito, em vez de atributos, o TILDE usa testes lógicos (consultas Prolog) para particionar os dados. Ao decidir que uma folha deve ser particionada (split), o conceito de *refinamento* de uma cláusula é empregado.

Seja l uma folha a sofrer split, os candidatos a teste de l serão $\rho(\leftarrow Q)$. Onde ρ é o operador de refinamento, que, no caso do TILDE, gera novas conjunções adicionando literais a $\leftarrow Q$. Os literais que podem ser usados nos refinamentos são especificados pelo usuário, através do viés de linguagem do problema. Tal especificação deve ser listada em um arquivo de configurações, de extensão ".s", a ser provido ao sistema.

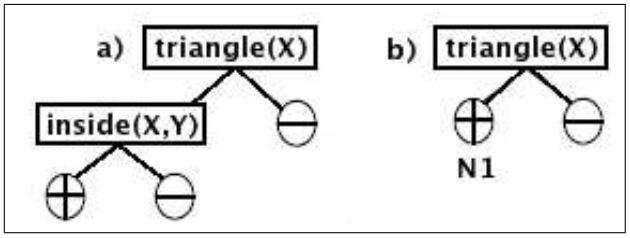


Figura 3.5: Exemplo de árvore gerada pelo TILDE.

Considere a árvore mostrada na figura 3.5.b. Suponha que desejamos saber quais testes podem ser colocados no nó $N1$. Uma vez que este nó é o filho esquerdo da raiz, sabemos que $triangulo(X)$ é verdade em $N1$ e, portanto, todos os refinamentos conterão este literal.

Digamos que o viés de linguagem determina que os literais que podem ser usados são: $triangulo(+\text{-}X)$, $dentro(+X,\text{-}Y)$, $dentro(\text{-}Y,+X)$, $quadrado(\text{-}X)$ e $circulo(\text{-}X)$, onde os sinais + e - indicam se as variáveis são de entrada ou saída. Considere ainda que cada literal pode ser usado apenas uma única vez no processo indução. Tal informação é descrita com as seguintes declarações:
$rmode(1 : triangulo(+\text{-}X))$.
$rmode(1 : dentro(+X,\text{-}Y))$.
$rmode(1 : dentro(\text{-}Y,+Y))$.
$rmode(1 : quadrado(\text{-}X))$.
$rmode(1 : circulo(\text{-}X))$.

Aplicando o operador de refinamento ρ em $\leftarrow triangulo(X)$, obtemos:

$$\rho(\leftarrow triangulo(X)) = \{\leftarrow triangulo(X), dentro(X,Y);$$
$$\leftarrow triangulo(X), dentro(Y,X);$$
$$\leftarrow triangulo(X), quadrado(Y);$$
$$\leftarrow triangulo(X), circulo(Y)\}$$

Figura 3.6: Exemplo de possíveis refinamentos para o filho esquerdo da raiz da árvore da figura 3.5.

Se o melhor refinamento for o primeiro, então o literal colocado no nó $N1$ será $dentro(X,Y)$, e seu teste será $\leftarrow triangulo(X), dentro(X,Y)$. Ao final, a estrtura da nova árvore será conforme a figura 3.5.a.

É importante destacar que tais refinamentos são processados sob a condição de θ-subsumption [2]. Também são consideradas restrições sobre o espaço de busca escolhidas pelo usuário, que podem ser estabelecidas, principalmente, pelo uso dos tipos e modos das variáveis. Uma referência mais completa sobre as configurações dos refinamentos e viés de linguagem do TILDE pode ser encontrada em [8].

Agregações

O TILDE/TILDE-RT é capaz de usar refinamentos baseados em funções de agregação. Estas função são úteis pois em ILP um exemplo pode ter uma quantidade arbitrária de de fatos sobre um mesmo predicado. Através das funções de agregação estes dados podem ser resumidos e a informação de agregação usada como refinamento.

O sistema ACE possibilita o uso de agregações como: min, máx, média, soma, moda, dentre outras. Para usar como refinamento um valor agregado, é necessário acrescentar a informação de tipo, bem como o padrão da agregação a ser usado. Por exemplo, considere um problema em que cada interpretação contém a informação sobre uma única pessoa e suas várias fontes de renda, sendo esta informação representada através da relação: $pagamento(Pessoa, Fonte, Valor)$. Caso desejássemos usar como refinamento a cláusula que separa pessoas com rendimento total menor que $R\$5000,00$, uma construção similar a seguinte deve ser usada:

1. $type(aggregate(funcDeAgregacao, padrao, variavelAgregada, resultado))$.
2. $rmode((aggregate(sum, pagamento(+Pessoa, \_Fonte, -Valor), Valor, Total), Total < 5000.00))$.

Com a declaração 1, indica-se o tipo do refinamento e o padrão de seus argumentos: o primeiro argumento é a função de agregação a ser usada; o segundo argumento

é o padrão a ser encontrado para agregação; o terceiro argumento é a variável que será agregada, e o quarto argumento é o resultado final da agregação.

Com a declaração 2, dizemos que a conjunção ($aggregate(sum$, $pagamento(+Pessoa$, $\_Fonte$, $-Valor)$, $Valor$, $Total)$, $Total < 5000.00$) pode ser acrescentada a uma cláusula que se deseja refinar. Esta conjunção soma o total recebido de todas as fontes de renda por uma pessoa e verifica se o total excede $R\$5000,00$.

É possível usar agregações muito mais complexas, maiores informações podem ser obtidas em [8].

3.1.5 Pacotes de cláusulas

Um pacote de cláusulas (Query Pack) [7] é um método de representar cláusulas sob a forma de disjunções a fim de evitarem-se provas idênticas de literais. Cláusulas de mesma cabeça têm suas caudas reunidas sob a forma de uma disjunção, de tal modo que literais comuns à cláusulas diferentes são devidamente fatorados e evidenciados.

Este conceito é melhor entendido através de um exemplo. Suponha que tenhamos um conjunto de cláusulas e desejamos saber quais são verdadeiras ou falsas para um exemplo e.

1: $p(X)$.
2: $p(X), q(X, a)$.
3: $p(X), q(X, b)$.
4: $p(X), q(X, c)$.
5: $p(X), q(X, Y), t(X)$.
6: $p(X), q(X, Y), t(X), r(Y, 1)$.
7: $p(X), q(X, Y), t(X), r(Y, 2)$.

Figura 3.7: Conjunto de cláusulas que se deseja saber se provam e

Tradicionalmente, armazenaríamos a asserção sobre e como um fato e cada cláusula seria testada individualmente. Note que, o literal $p(X)$ tem que ser provado sete vezes e a conjunção $p(X), q(X, Y), t(X)$ três vezes. Para evitar tal desperdício, o TILDE representa tais cláusulas como uma disjunção. Isto é importante no momento em que o TILDE computa a heurística para cada refinamento, onde é comum que dois ou mais refinamentos tenham literais em comum. A árvore na figura 3.8 representa o pacote de cláusulas de tal conjunto.

3.1.6 Tratamento de Regressão - TILDE-RT

O TILDE também é capaz de tratar problemas de regressão, sendo chamado TILDE-RT (TILDE Regression Trees) neste caso. O modelo de *Aprendizado por*

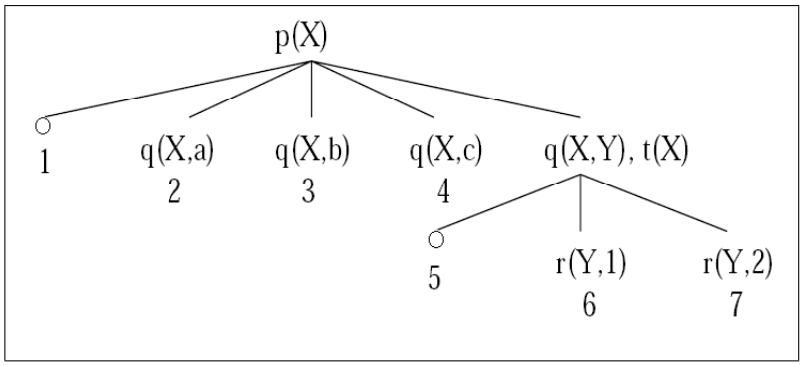


Figura 3.8: Pacote de cláusulas da figura 3.7.

```
begin(model(1)).          begin(model(10)).
valor(0.0).               valor(0.0).
ruim(gear).               ruim(uc).
ruim(engrenagem).         end(model(10)).
end(model(1)).            ...

begin(model(2)).
valor(1.0).
ruim(roda).
ruim(corrente).
end(model(2)).
...
```

Figura 3.9: Arquivo kb do problema Machines

Interpretações usado na classificação se mantém, porém, em vez de um fato sobre a classe do exemplo, deve haver um fato sobre o predicado alvo, e tal predicado deve ter uma variável numérica que será o alvo da regressão.

Para exemplificar a representação, vamos apresentar a versão do dataset *Machines* [8] para regressão. Neste problema, deseja-se saber o valor de uma máquina quando esta possui um componente desgastado/defeituoso (ruim/1). Cada interpretação possui a lista dos componentes defeituosos e o valor associado a máquina, conforme a figura 3.9.

Uma máquina pode ter: roda, motor, corrente, unidade de controle (uc) e engrenagem. Adicionalmente, existem componentes que podem ser substituídos (substituivel/1): roda, corrente e engrenagem; e componentes que não podem (insubstituivel/1): motor e unidade de controle. Esta informação é representada no conhecimento preliminar, figura 3.10.

No arquivo de configuração do problema devemos declarar que o TILDE deverá rodar em modo de regressão, através da declaração "$tilde\_mode(regression)$.". É necessário, também, especificar qual é o predicado alvo, bem como a variável dependente, através da declaração "$predict(valor(-))$.". A variável da regressão deve ser

```
substituivel(roda).
substituivel(corrente).
substituivel(engrenagem).

insubstituivel(motor).
insubstituivel(uc).
```

Figura 3.10: Conhecimento preliminar do problema Machines

uma variável de saída, identificada através do *modo* −, neste exemplo, o predicado tem apenas uma variável. Caso houvesse outras variáveis, deve-se especificar os modos de cada uma. Por consequência desta representação o TILDE-RT é capaz de tratar regressão em mais de uma variável, porém este assunto foge ao escopo deste trabalho.

As seguintes declarações podem ser usadas para especificar os literais da linguagem que podem ser usados nos refinamentos: "$rmode(substituivel(+X))$.", "$rmode(insubstituivel(+X))$." e "$rmode(ruim(+\text{-}X))$.".

Na figura 3.11 há um exemplo de árvore gerada pelo TILDE-RT para o problema Machines em modo de regressão.

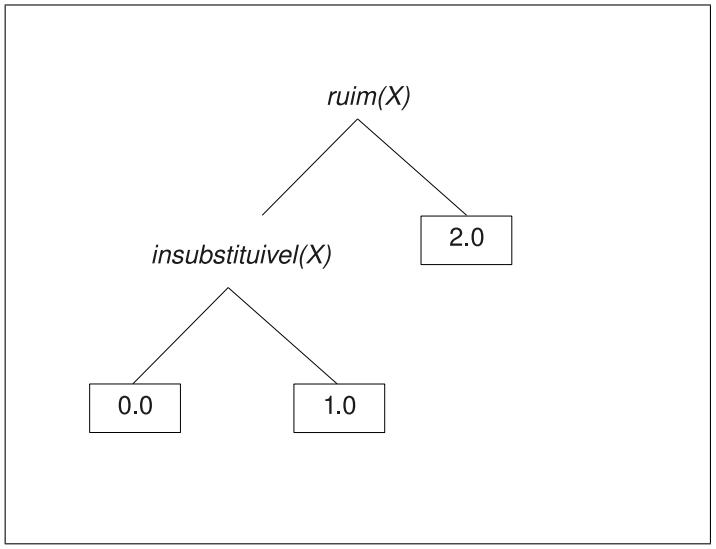


Figura 3.11: Arvore gerada para o problema Machines

3.1.7 Heurística

Quando em modo de classificação, o TILDE usa o *Ganho de Informação* de forma análoga ao C4.5, bem como outras heurísticas. Conforme dito, cada refinamento é um teste lógico que particiona um conjunto original em dois subconjuntos: dos exemplos que sucedem e dos exemplos que falham. Para cada refinamento a heurística é computada levando em conta os dois subconjuntos gerados, e o teste escolhido é o que obtiver melhor valor para heurística.

Em modo de regressão, a heurística usada é a *Redução na Soma dos Quadrados das Distancias*. Esta heurística também pode ser usada pelo C4.5 em vez da redução da variância (default no C4.5). Na sequência definiremos a soma dos quadrados das distâncias para um conjunto de números reais e, em seguida, a heurística em si.

Seja S um conjunto de exemplos, então a soma dos quadrados das distâncias do conjunto S, $SS(S)$, é dada por:

$$SS(S) = \sum_{i=1}^{|S|} (y_i - \mu(S))^2 \tag{3.1}$$

onde y_i é o valor da variável dependente no atributo alvo do i-ésimo exemplo de S e $\mu(S) = \frac{\sum_{i=1}^{|S|} y_i}{|S|}$.

Adicionalmente, sejam: l uma folha, L o conjunto de valores da variável regredida nos exemplos da folha l, ρ um candidato a refinamento em l, $P(\rho) \subset L$ os valores dos exemplos que passam no teste de ρ e $N(\rho) \subset L$ os valores dos exemplos que falham em ρ. Então, o valor da heurística do refinamento ρ na folha l, $H(\rho, l)$, é dado por:

$$H(\rho, l) = SS(L) - (SS(N(\rho)) + SS(P(\rho))) \tag{3.2}$$

Concluindo, desejamos o refinamento ρ que maximiza H na equação 3.2, i.e., que cause a maior redução na soma dos quadrados das distâncias quando comparamos os subconjuntos gerados ao conjunto original.

Note que, a única diferença entre a soma dos quadrados das distâncias é a ausência da divisão pelo número total de exemplos de S, conforme 3.3. Para diferenciação, sejam: $\Delta S^2(S)$ a redução da variância; e $\Delta SS(S)$ a redução na soma das distâncias quadráticas. Para simplificar, para um mesmo refinamento ρ faremos: $N = N(\rho)$ e $P = P(\rho)$.

$$
\begin{aligned}
\Delta S^2(S) &= S^2(S) - ((|N|/|S|) * S^2(N) + (|P|/|S|) * S^2(P)) \\
&= SS(S)/|S| - ((|N|/|S|) * SS(N)/|N| + (|P|/|S|) * SS(P)/|N|) \\
&= (SS(S)/|S| - ((1/|S|) * SS(N) + (1/|S|) * SS(P))) \\
&= (1/|S|) * (SS(S) - (SS(N) + SS(P))) \\
&= (1/|S|) * \Delta SS(S). \qquad (3.3)
\end{aligned}
$$

3.1.8 Indução de árvores de decisão de lógica de primeira ordem

Agora que os conceitos por trás do algoritmo foram expostos, finalizaremos este capítulo mostrando o pseudo-código do algoritmo TILDE que aprende árvores decisão de lógica de primeira ordem (First Order Logical Decision Trees - FOLDT).

Entretanto, antes de apresentar o algoritmo, é importante explicar a notação para os nós da árvore. Um nó da árvore pode ser: 1) uma folha, que possui um modelo preditivo associado ($N = folha(modelo\_preditivo(E))$), onde E é o conjunto de exemplos que passaram pela folha durante o treinamento; ou 2) um nó interno ($N = noInterno(conj, esq, dir)$), que possui filhos esq e dir, e uma consulta associada $conj$.

Novas variáveis introduzidas em um nó não são repassadas ao ramo direito, pois tais variáveis não têm sentido para os exemplos que falham no teste deste nó. Este fato é explicitado na chamada do método $ConstroiArvore$ para os ramos que ficam à direita, no algoritmo 3.1 mostrado a seguir.

O modelo preditivo depende do tipo de problema tratado: na classificação é a classe majoritária em E, na regressão é média da variável dependente que aparece no predicado alvo dos exemplos em E. Adicionalmente, os critérios de parada podem ser verificados na avaliação dos refinamentos, de modo que caso algum critério seja alcançado, nenhum refinamento será considerado.

Finalizaremos este capítulo mostrando os algoritmos de aprendizado e classificação do TILDE.

A chamada ao método recursivo ControiArvore(.), linha 1, dá início ao processo de indução. A indução começa na raiz, cuja consulta associada é definida como $Q = true$ e um conjunto de exemplos de treino, E, lhe é dado.

Em um nó N qualquer os possíveis refinamentos de $\leftarrow Q$ são determinados aplicando-se o operador de refinamentos, $\rho(\leftarrow Q)$. Desta forma, calcula-se a heurística para cada refinamento e o melhor refinamento, $\leftarrow Q_b$, é escolhido, linha 1.

Caso $\leftarrow Q_b$ não seja adequado, apesar de ser o melhor, então N torna-se uma

Algoritmo 3.1 Algoritmo do TILDE, para indução de árvores de decisão de lógica de primeira ordem, proposto em [2] [3]

Entrada:

E é um conjunto de exemplos,
Q é uma consulta Prolog.

Saída:

T é uma árvore de decisão de lógica de primeira ordem.

Procedimento ConstroiArvore (E, Q)

1: Seja $\leftarrow Q_b$ o elemento de $\rho(\leftarrow Q)$ com o melhor valor de heurística.
2: Se $\leftarrow Q_b$ não for bom (por exemplo, tiver ganho nulo), então
3: Faça T = folha(modelo_preditivo(E)).
4: Senão
5: Faça $conj = Q_b$ - Q.
6: Faça $E1 = \{e \in E | \leftarrow Q_b$ é verdadeira em $e \wedge B\}$.
7: Faça $E2 = \{e \in E | \leftarrow Q_b$ é falsa em $e \wedge B\}$.
8: Faça esq = ConstroiArvore($E1$, Q_b).
9: Faça dir = ConstroiArvore($E2$, Q).
10: Faça T = noInterno($conj$, esq, dir).
11: Retorne T.

Entrada:

E é um conjunto de exemplos.

Saída:

T é uma árvore.

Procedimento Tilde (E)

1: Seja T = ConstroiArvore(E, $true$).
2: Retorne T.

folha com um certo *modelo preditivo*, o qual leva em conta os exemplos em E, linhas 2 e 3. No caso das árvores do TILDE tal modelo prediz a classe mais frequente em E.

Caso contrário, isto é, $\leftarrow Q_b$ pode ser usado como teste, então a conjunção $conj$, a qual foi usada para refinar $\leftarrow Q$, é extraída de $\leftarrow Q_b$ e é armazenada no nó, linha 5. Portanto, um split acaba de ocorrer no nó sendo processado.

Os exemplos de E são particionados em $E1$ e $E2$ de acordo com o teste $\leftarrow Q_b$. Na sequência, criam-se dois novos ramos: esq (esquerdo) que ficará associado à subárvore que contempla os exemplos que sucedem no teste do nó; e dir (direito) que ficará associado à subárvore que contempla os exemplos que falham no teste, linhas 6 e 7.

O processo de indução é recursivamente executado, chamado-se o procedimento ConstroiArvore para cada novo ramo. Conforme dito, no ramo esquerdo a consulta associada é $\leftarrow Q_b$ e no ramo direito a consulta associada é $\leftarrow (Q_b - conj) = Q$. Portanto, cada ramo tornar-se-á uma subárvore, linhas 8 e 9.

No retorno das recursões, o nó N, torna-se um nó interno cujos filhos são esq e dir e que armazena a conjunção $conj$ em sua estrutura, de modo que seu teste é $\leftarrow (Q \wedge conj)$, linha 10.

Algoritmo 3.2 Algoritmo de classificação de um exemplo usando um $FOLDT$ (com um conhecimento preliminar B), proposto em [2] [3]

Entrada:

e é um exemplo.

Saída:

c é uma classe.

Procedimento Classifica (e)

1: Seja Q a consulta Prolog $\leftarrow true$.
2: Seja N o nó raiz da árvore.
3: Enquanto $N \neq$ **folha**(c)
4: Seja $N =$ **noInterno**$(conj, esq, dir)$.
5: Se $Q \wedge conj$ for verdadeira em $e \wedge B$, então
6: Faça $Q = Q \wedge conj$.
7: Faça $N = esq$.
8: senão
9: Faça $N = dir$.
10: Retorne c (a classe da folha).

O procedimento de classificação de um exemplo e inicia na raiz, cuja consulta associada Q é $\leftarrow true$, linhas 1 e 2.

Para o nó N qualquer, caso não seja uma folha, então componha seu o teste usando Q e $conj$, i.e., $\leftarrow Q \wedge conj$, caso e suceda no teste, faça $N = esq$ e $Q =\leftarrow$

$Q \wedge conj$, onde esq é seu filho esquerdo, caso contrário faça $N = esq$ e $Q = Q$, linhas 3 a 9.

Caso contrário, i.e., N é uma folha, então retorne a classe armazenada pela folha como predição para e, linha 10.

3.2 VFDT - Very Fast Decision Tree

Nesta seção, será apresentado o sistema VFDT (*Very Fast Decision Tree*) [4], que é um dos sistemas que serviram de base para o nosso trabalho. Na seção 3.2.1, faremos uma descrição geral do método; na seção 3.2.2, apresentaremos o resultado estatístico conhecido como *limitante de Hoeffding*; na seção 3.2.3, mostraremos como o limitante de Hoeffding foi aplicado no aprendizado de árvores de decisão; na seção 3.2.4, discutiremos o comportamento assintótico do algoritmo; na seção 3.2.5, discutiremos como o VFDT trata atributos numéricos; e na seção 3.2.6, apresentaremos o algoritmo do VFDT.

3.2.1 Introdução

O VFDT um é algoritmo escalável que aprende árvores proposicionais para problemas de classificação de forma incremental. Para tal, o VFDT inspeciona o conjunto de treino de forma sequencial refinando o modelo aprendido progressivamente, à medida que novos exemplos são inspecionados. Ao inspecionar uma quantidade suficiente de exemplos numa certa folha, determinada pelo resultado estatístico conhecido como *limitante de Hoeffding*, o algoritmo escolhe um novo atributo para teste, realizando o *split* nesta folha, entretanto, o VFDT descarta tais exemplos em vez de repassá-los aos novos ramos da árvore. Adicionalmente, por ser incremental, o VFDT pode realizar a tarefa de classificação antes do processo de aprendizado ter sido totalmente finalizado, i.e., que os exemplos de treinamento tenham se esgotado.

3.2.2 Limitante de Hoeffding

Seja r uma variável aleatória real cujo domínio tem tamanho R. Considere n observações independentes desta variável e seja $\overline{r}$ sua média observada. O limitante de Hoeffding [6] atesta que $P(\mu_r \geq \overline{r} - \epsilon) = 1 - \delta$, onde μ_r é a verdadeira média da variável r, ϵ é dado pela equação 3.4 e δ é um número muito pequeno.

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{3.4}$$

3.2.3 Árvores de Hoeffding

Proposta por Domingos e Hulten [4], a modelagem mostrada a seguir utiliza o limitante de Hoeffding na escolha dos testes. Em função da aplicação do limitante de Hoeffding parametrizado por δ, da-se o nome de *árvores de Hoeffding* para tais modelos.

Seja G a heurística a ser maximizada, sejam X_a e X_b os atributos com os maiores valores para G. Seja $\Delta G_r = G(X_a) - G(X_b)$ a diferença nos valores das heurísticas dos atributos quando o algoritmo utiliza todo o conjunto de dados, e seja $\Delta\overline{G} = \overline{G}(X_a) - \overline{G}(X_b)$ a diferença observada quando apenas n exemplos chegaram a um nó.

Então, dado um valor para δ, através do limitante de Hoeffding sabemos que com probabilidade $1-\delta$ que $\Delta G_r \geq \Delta\overline{G} - \epsilon$, onde ϵ é dado pela equação 3.4. Se $\Delta\overline{G} > \epsilon$, então podemos dizer que com mesma probabilidade $\Delta G_r > 0$, logo $G(X_a) > G(X_b)$, i.e., com probabilidade $1-\delta$, X_a é o melhor atributo.

Em termos práticos, para podermos afirmar se X_a é a melhor escolha, precisamos verificar se $\Delta\overline{G} \geq \epsilon$. Então, o que precisamos fazer é processar exemplos até acontecer que $\Delta\overline{G} \geq \epsilon$, para o valor escolhido de δ e o número corrente de exemplos numa folha.

Pode acontecer que o melhor atributo, X_a, seja muito melhor que X_b, porém realizar o split usando X_a pode não ser tão vantajoso. Para tratar tal situação, considera-se um atributo extra, $X_\emptyset$, que na verdade representa o valor da heurística considerando apenas os exemplos na folha. Desta forma pode-se tratar o caso de um split no melhor atributo não causar uma diferenciação efetivamente boa.

Destacamos que o limitante de Hoeffding independe da distribuição da variável considerada. Esta independência vem com o custo de ser necessário processar mais exemplos do que em métodos que usam limitantes específicos. Porém, dado o cenário das grandes massas de dados, isto não configura um problema. Além disso, para o ganho de informação temos $0 \leq G \leq log_2(|C|)$, onde C é o conjunto das possíveis classes, o que satisfaz a condição de a variável ter seu domínio delimitado por uma faixa de tamanho R. É importante destacar que a distribuição das observações deve ser a mesma, portanto é feita uma hipótese de *distribuição estacionária* no fluxo.

Segue, então, o pseudo-código do algoritmo de indução das árvores de Hoeffding.

O algoritmo de indução das árvores de Hoeffding começa na raiz l_1, único nó e folha da árvore sendo construída. Conforme dito, além dos possíveis atributos $\mathbf{X}$, considera-se um atributo extra, $X_\emptyset$. Ele, além de funcionar como um contador global na folha, representa o caso de não se fazer teste algum, linhas 1 e 2. Além disso, o valor da heurística de $X_\emptyset$ é justamente a entropia do conjunto.

Inicialmente, os contadores do nó, identificados pela trinca: $i = atributo$, $j =$

Algoritmo 3.3 Algoritmo de indução de uma árvore de Hoeffding, proposto em [4]

Entradas:

S é uma sequência de exemplos,

$\mathbf{X}$ é um conjunto de atributos discretos,

$G(.)$ é uma função de avaliação de *split*,

δ é um menos a probabilidade desejada de se escolher o atributo correto em um dado nó.

Saída:

HT é uma árvore de decisão.

Procedimento HoeffdingTree (S, $\mathbf{X}$, G, δ)

Seja HT uma árvore com uma única folha l_1 (a raiz).
Seja $\mathbf{X}_1 = \mathbf{X} \cup \{X_\emptyset\}$.
Seja $\overline{G_1}(X_\emptyset)$ o valor de $\overline{G}$ obtido ao se predizer a classe mais frequente em S.
Para cada classe y_k
    Para cada valor x_{ij} de cada atributo $X_i \in \mathbf{X}$
        Faça $n_{ijk}(l_1) = 0$.
Para cada exemplo ($\mathbf{x}$, y_k) em S
    Desça ($\mathbf{x}$, y) até uma folha l usando HT.
    Para cada x_{ij} em $\mathbf{x}$ tal que $X_i \in \mathbf{X}_l$
        Incremente $n_{ijk}(l)$.
    Rotule a folha l com a classe majoritária dos exemplos que chegaram nela até agora.
    Se os exemplos que chegaram até agora em l não forem todos da mesma classe, então
        Calcule $\overline{G_l}(X_i)$ para cada atributo $X_i \in \mathbf{X}_l$ - $\{X_\emptyset\}$ usando os contadores $n_{ijk}(l)$.
        Faça X_a ser o atributo com o maior valor de $\overline{G_l}$.
        Faça X_b ser o atributo com o segundo maior valor de $\overline{G_l}$.
        Calcule ϵ usando a equação 3.4.
        Se $\overline{G_l}(X_a)$ - $\overline{G_l}(X_b) > \epsilon$ e $X_a \neq X_\emptyset$, então
            Substitua l por um nó interno cujo teste é o atributo X_a (ou seja, faça o *split* com o atributo X_a).
            Para cada ramo do *split*
                Adicione uma nova folha l_m, e faça $\mathbf{X}_m = \mathbf{X}$ - $\{X_a\}$.
                Faça $\overline{G_m}(X_\emptyset)$ ser o valor de $\overline{G}$ obtido ao se predizer a classe mais frequente em l_m.
                Para cada classe y_k e cada valor x_{ij} de cada atributo $X_i \in \mathbf{X}_m$ - $\{X_\emptyset\}$
                    Faça $n_{ijk}(l_m) = 0$.
Retorne HT.

j-ésima possível valoração do atributo i e $k = classe$, são inicializados com valor 0, linhas 5 e 6.

Após tal inicialização, os exemplos de S começam a ser passados pela árvore em construção. Cada exemplo $(\mathbf{x}, y_k) \in S$ a atravessa até chegar em uma folha l adequada, linhas 7 e 8.

Uma vez que o exemplo $(\mathbf{x}, y_k)$ chegue na folha l, cada contador deve ser atualizado, de acordo com as valorações dos atributos de $\mathbf{x}$, linhas 9 e 10. l é, então, atualizada para predizer a classe que mais se repetiu no exemplos até então passados, linha 11.

Caso todos os exemplos pertençam à mesma classe, então l deve aguardar que mais exemplos cheguem, pois nenhum split é possível. Caso contrário, a heurística deve ser computada para cada atributo, a fim de determinar os dois melhores, linhas 12 a 15.

Determinados os dois melhores atributos, candidatos a teste no split, computa-se o limitante de Hoeffding. Verifica-se, então, se a diferença entre ambos já é suficiente, para que seja decidido se podemos realizar o split usando o melhor atributo, linhas 16 e 17.

Caso o limitante garanta que atributo X_a é estatisticamente melhor, então l, que era uma folha, torna-se um nó interno, cujo teste é o próprio X_a. Adicionalmente, um ramo descendente é criado para cada possível valor no domínio de X_a, linhas 18 a 20. Convém observar que os antigos contadores de l são descartados neste processo, portanto qualquer resquício de informação sobre os exemplos é perdida.

Para cada novo filho l_m, analogamente separa-se um atributo $X_\emptyset$. Os contadores dos filhos são analogamente inciados com valor 0, linhas 21 a 23.

Então, o processo retorna ao ciclo principal, descrito na linha 7. Desta forma, mais um exemplo será conduzido até uma folha, a qual avaliará a pertinência de um split. Ao esgotarem-se os exemplos o processo termina.

3.2.4 Propriedades das Árvores de Hoeffding

Em [4] foi demostrado que as árvores de Hoeffding são assintoticamente semelhantes àquelas produzidas por um algoritmo que escolhe os testes para seus nós inspecionando todo o conjunto de dados de treinamento, *batch*. Reproduziremos, portanto, tal demostração a seguir.

Propriedade 3.1. *O algoritmo de árvores de Hoeffding produz árvores assintoticamente próximas às produzidas por um algoritmo batch.*

A demostração da propriedade em questão depende de conceitos que caracterizem diferenças entre duas árvores. Para tal, os seguintes conceitos são definidos:

Definição 3.2 (Desigualdade entre nós)**.** *Dois nós de árvores de decisão são diferentes nos seguintes casos:*

- *Dois nós internos são diferentes quando os seus testes são diferentes.*
- *Duas folhas são diferentes quando predizem classes diferentes.*
- *Um nó interno é diferente de uma folha.*

Definição 3.3 (Desigualdade entre caminhos)**.** *Dois caminhos de árvores de decisão são diferentes se eles diferem em tamanho ou em pelo menos um nó.*

A partir desdes dois conceitos, derivam-se as definições de *discordância* entre árvores. Sejam:

- $P(\mathbf{x})$ a probabilidade do vetor de atributos $\mathbf{x}$ ser observado.
- $I(.)$ uma função que retorna 1 se o seu argumento for verdadeiro e 0 em caso contrário.

Definição 3.4 (Discordância Extensional)**.** *Seja $DT_i(\mathbf{x})$ a classificação dada pela árvore DT_i para o exemplo $\mathbf{x}$. A discordância extensional Δ_e entre duas árvores de decisão DT_1 e DT_2 é a probabilidade de elas predizerem classes diferentes para um mesmo exemplo:*

$$\Delta_e(DT_1, DT_2) = \sum_{\mathbf{x}} P(\mathbf{x}) I[DT_1(\mathbf{x}) \neq DT_2(\mathbf{x})] \tag{3.5}$$

Definição 3.5 (Discordância Intensional)**.** *Seja $Caminho_i(\mathbf{x})$ o caminho do exemplo $\mathbf{x}$ pela árvore DT_i. A discordância intensional Δ_i entre duas árvores de decisão DT_1 e DT_2 é a probabilidade do caminho de um exemplo pela árvore DT_1 ser diferente do caminho do mesmo exemplo pela árvore DT_2.*

$$\Delta_i(DT_1, DT_2) = \sum_{\mathbf{x}} P(\mathbf{x}) I[Caminho_1(\mathbf{x}) \neq Caminho_2(\mathbf{x})] \tag{3.6}$$

Duas árvores concordam de forma intensional em relação a um exemplo se este atravessa por cada uma das árvores usando a mesma sequência de nós e ambas predizem a mesma classe para ele. Vale notar que discordância intensional é um conceito mais forte do que discordância extensional.

Seja p_l a probabilidade de que um exemplo que desça até o nível l em uma árvore de decisão alcance uma folha neste nível. Para simplificar, assumiremos que esta probabilidade é constante para todas as folhas ($\forall l\ p_l = p$), o que é uma hipótese realista para árvores de decisão típicas. Chamaremos p de *probabilidade de folha*. Seja HT_δ a árvore de Hoeffding gerada com a probabilidade desejada δ, dada uma sequência infinita de exemplos S. Seja DT_* a árvore de decisão assintótica produzida por um algoritmo *batch*, em que se escolhe em cada nó o atributo com maior valor de G, usando-se para isso infinitos exemplos em cada nó. Seja $E[\Delta_i(HT_\delta,DT_*)]$ o valor esperado de $\Delta_i(HT_\delta,DT_*)$, calculado considerando-se todas as infinitas sequências de treinamento possíveis. Podemos afirmar o seguinte resultado:

Teorema 3.1. *Se HT_δ é a árvore de decisão de Hoeffding produzida com a probabilidade desejada δ dados infinitos exemplos, DT_* é a árvore de decisão assintótica produzida por um algoritmo batch e p é a probabilidade de folha, então $E[\Delta_i(HT_\delta, DT_*)] \leq \delta/p$.*

Prova: Considere um exemplo $\mathbf{x}$ que, na árvore HT_δ, cai em uma folha no nível l_h e, na árvore DT_*, cai em uma folha no nível l_d. Sejam:

- $l = \min\{l_h,l_d\}$.
- $Caminho_H(\mathbf{x}) = (N_1^H(\mathbf{x}), N_2^H(\mathbf{x}), ..., N_l^H(\mathbf{x}))$ o caminho de $\mathbf{x}$ pela árvore HT_δ até o nível l, onde $N_i^H(\mathbf{x})$ é o nó em que o exemplo $\mathbf{x}$ cai no nível i em HT_δ. Analogamente, seja $Caminho_D(\mathbf{x})$ o caminho de $\mathbf{x}$ pela árvore DT_*. Se $l = l_h$, então $N_l^H(\mathbf{x})$ é uma folha e, portanto, prediz uma classe; de forma semelhante, se $l = l_d$, então $N_l^D(\mathbf{x})$ é uma folha.
- I_i a proposição "$Caminho_H(\mathbf{x}) = Caminho_D(\mathbf{x})$ até o nível i, inclusive", e $I_0 = true$. Note que $P(N_l^H(\mathbf{x}) \neq N_l^D(\mathbf{x})|I_{l-1})$ inclui $P(l_h \neq l_d)$, porque se os dois caminhos têm tamanhos diferentes, então uma árvore deve ter uma folha onde a outra tem um nó interno.

Então,

$P(Caminho_H(\mathbf{x}) \neq Caminho_D(\mathbf{x}))$

$$
\begin{aligned}
&= P((N_1^H \neq N_1^D) \vee (N_2^H \neq N_2^D) \vee ...(N_l^H \neq N_l^D)) \\
&= P(N_1^H \neq N_1^D | I_0) + P(N_2^H \neq N_2^D | I_1) + ... + P(N_l^H \neq N_l^D | I_{l-1})
\end{aligned}
$$

$$
= \sum_{i=1}^{l} P(N_i^H \neq N_i^D | I_{i-1}) \leq \sum_{i=1}^{l} \delta = \delta l \tag{3.7}
$$

Seja $HT_\delta(S)$ a árvore de Hoeffding gerada pela sequência de exemplos de treinamento S. Então $E[\Delta_i(HT_\delta, DT_*)]$ é a média, considerando-se todas as infinitas sequências de treinamento S, da probabilidade do caminho de um exemplo na árvore $HT_\delta(S)$ ser diferente do caminho do mesmo exemplo na árvore DT_*:

$E[\Delta_i(HT_\delta, DT_*)]$

$$
\begin{aligned}
&= \sum_S P(S) \sum_{\mathbf{x}} P(\mathbf{x}) I[Caminho_H(\mathbf{x}) \neq Caminho_D(\mathbf{x})] \\
&= \sum_{\mathbf{x}} P(\mathbf{x}) P(Caminho_H(\mathbf{x}) \neq Caminho_D(\mathbf{x})) \\
&= \sum_{i=1}^{\infty} \sum_{\mathbf{x} \in L_i} P(\mathbf{x}) P(Caminho_H(\mathbf{x}) \neq Caminho_D(\mathbf{x}))
\end{aligned} \tag{3.8}
$$

onde L_i é o conjunto de exemplos que caem em uma folha de DT_* no nível i. De acordo com a equação 3.7, a probabilidade do caminho de um exemplo na árvore $HT_\delta(S)$ ser diferente do caminho do mesmo exemplo na árvore DT_*, dado que este último tem tamanho i, é no máximo δi (já que $i \geq l$). Então,

$$
E[\Delta_i(HT_\delta, DT_*)] \leq \sum_{i=1}^{\infty} \sum_{\mathbf{x} \in L_i} P(\mathbf{x})(\delta i) = \sum_{i=1}^{\infty} (\delta i) \sum_{\mathbf{x} \in L_i} P(\mathbf{x}) \tag{3.9}
$$

A soma $\sum_{\mathbf{x} \in L_i} P(\mathbf{x})$ é a probabilidade de um exemplo $\mathbf{x}$ cair em uma folha de DT_* no nível i e é igual a $(1-p)^{i-1}p$, onde p é a probabilidade de folha. Então,

$E[\Delta_i(HT_\delta, DT_*)]$

$$
\begin{aligned}
&\leq \sum_{i=1}^{\infty} (\delta i)(1-p)^{i-1} p = \delta p \sum_{i=1}^{\infty} i(1-p)^{i-1} \\
&= \delta p \left[\sum_{i=1}^{\infty} (1-p)^{i-1} + \sum_{i=2}^{\infty} (1-p)^{i-1} + ... + \sum_{i=k}^{\infty} (1-p)^{i-1} + ... \right] \\
&= \delta p \left[\frac{1}{p} + \frac{1-p}{p} + ... + \frac{(1-p)^{k-1}}{p} + ... \right] \\
&= \delta \big[1 + (1-p) + ... + (1-p)^{k-1} + ... \big]
\end{aligned}
$$

$$= \delta \sum_{i=0}^{\infty} (1-p)^i = \frac{\delta}{p} \tag{3.10}$$

Isto completa a demonstração do Teorema 3.1.

3.2.5 Tratamento de atributos numéricos

Conforme visto na seção 2.2.2, ao tratar um atributo numérico Y, o C4.5 analisa todos os exemplos e tenta determinar qual o valor constante c, que gera o split binário $Y < c$ e $Y \geq c$, com o melhor valor de heurística.

O VFTD usa um método parecido com o do C4.5, entretanto, como não inspeciona todo os exemplos, ele armazena até 1000 valores que tenham sido observados numa folha e usa diretamente estes valores como os candidatos à constante do split. Vale salientar que, ainda assim, é necessário fazer a ordenação dos exemplos no momento de avaliar os candidatos. Como o algoritmo opera em streams, as otimizações obtidas pelas listas de atributos [14], para evitar excessivas ordenações, não podem ser usadas.

Existem muitas propostas de melhoria para o VFDT. Destacamos a abordagem usada em [18], onde se faz uso das árvores binárias de busca estendidas (Extended Binary Search Trees - E-BST). Cada atributo numérico em uma folha contém uma E-BST e cada nó de uma E-BST contém um valor chave que guia os exemplos ao sub ramos esquerdo ou direito através do teste "$y < Chave$". Adicionalmente cada nó possui dois vetores, um para cada filho, com as estatísticas: Número de exemplos, soma dos valores da atributo-alvo e soma dos quadrados dos valores. Cada vez que um exemplo novo chega: a chave é confrontada, o contador adequado é atualizado e o exemplo é repassado ao ramo adequado. Na hora do split, para achar a melhor constante, o algoritmo realiza uma busca em profundidade visitando cada nó, portanto no ordem correta, e vai acumulando os resultados da heurística para cada possível valor, para que no final seja escolhido o de melhor heurística. Uma descrição mais completa o método pode ser encontrada em [18].

3.2.6 VFDT

Utilizando modelagem das árvores de Hoeffding, Domingos e Hulten desenvolveram o algoritmo VFDT (Very Fast Decision Tree) [4], um algoritmo de aprendizado proposicional e escalável em grandes bases de dados.

O VFDT tem alguns elementos para melhorar ainda mais sua performance. Iremos mencionar dois deles: um parâmetro para acúmulo de exemplos nas folhas e_{min}; e um parâmetro para desempate entre atributos τ. Detalhes extras sobre o VFDT podem ser encontrados em [4].

O cômputo da heurística é um processo custoso. O parâmetro e_{min} reduz o tempo de indução, permitindo que o usuário determine que cada novo cômputo da heurística seja feito somente quando e_{min} novos exemplos chegaram desde a folha vazia ou o último cômputo. Isto melhora a performance pois o algoritmo não precisa computar a heurística para todos os possíveis refinamentos todas as vezes que um único exemplo chega à folha. Portanto, no contexto de conjuntos de dados gigantes, isto tem um grande impacto.

Quando dois ou mais atributos possuem valores de heurísticas muito próximos, o algoritmo pode necessitar de um número grande de exemplos até escolher o melhor atributo, o que é um desperdício, uma vez que não fará grande diferença qual atributo seja escolhido, já que estão muito próximos. Esta situação é chamada de *empate* e para resolvê-la o VFDT usa um parâmetro fornecido pelo usuário que impõe um limiar de desempate. Caso $\Delta\overline{G} < \epsilon < \tau$, o melhor atributo é escolhido como o teste para o nó.

Terminaremos esta seção mostrando as diferenças entre o algoritmo tradicional de árvores de decisão [1] e o VFDT para comparação.

Algoritmo 3.4 Algoritmo de Árvore de Decisão [1]

1: Raiz: Exs = todos os exemplos de treino.
2: Baseado no conjunto de exemplos Exs, selecione o melhor atributo A como teste do nó.
3: Crie um ramo para cada valor v_i do atributo discreto A, correspondendo ao teste $A = v_i$.
4: Separe os exemplos entre os nós filhos do nó corrente de acordo com o valor de A, criando um novo conjunto de exemplos Exs para cada filho.
5: Repita o passo 2 para cada filho.

Algoritmo 3.5 VFDT [4]

1: *Raiz: Exs = N primeiros exemplos de treino, onde N é obtido pelo limitante de Hoeffding ($\Delta\overline{G} > \epsilon$ ou $\Delta\overline{G} < \epsilon < \tau$).*
2: Baseado no conjunto de exemplos Exs, selecione o melhor atributo A como teste do nó.
3: Crie um ramo para cada valor v_i do atributo discreto A, correspondendo ao teste $A = v_i$.
4: *Descarte os exemplos que estavam antes no nó. Separe os próximos exemplos entre os filhos do no corrente, de acordo com o valor de A, criando um conjunto de exemplos Exs para cada filho. O número de exemplos de cada filho será também determinado pelo limitante de Hoeffding.*
5: Repita o passo 2 para cada filho.

3.3 HTILDE - Hoeffding TILDE

Nesta seção, será apresentado o HTILDE. Este sistema é fundamental para este trabalho, pois o algoritmo HTILDE-RT é uma extensão do HTILDE para resolver problemas de regressão. Na seção 3.3.1, faremos uma introdução ao sistema e na seção 3.3.2, o algoritmo do HTILDE será exposto, na seção 3.3.3, detalhes relacionados a especificação de um problema serão apresentados.

3.3.1 Introdução

O HTILDE, proposto em [9], é um algoritmo baseado no TILDE e no VFDT, combinando a expressividade da ILP à eficiência da amostragem através do limitante de Hoeffding. Sendo, portanto, um sistema de programação em lógica indutiva escalável para aprendizado em fluxos de dados relacionais.

Por ser baseado no TILDE, o HTILDE também utiliza *aprendizado por interpretações*, o que permite a cobertura local dos exemplos, bem como se beneficia dos

pacotes de cláusulas. Porém, o processo de escolha dos refinamentos para teste se dá de forma análoga à que o VFDT usa para escolher atributos para teste, herdando, portanto, os três parâmetros básicos do VFDT: δ, e_{min} e τ

Vários sistemas proposicionais usaram a metodologia proposta por Domingos e Hulten [33] para tornar algoritmos de aprendizado de máquinas capazes de tratar grandes fluxos de dados. Dentre eles, o VFDT [4] é o mais próximo do TILDE, por ser uma árvore de decisão.

Entretanto, conforme constatado por Srinivasan [34], sampling em ILP não tem recebido muita atenção. Eu seu trabalho pioneiro existe uma metodologia de sub-sampling aplicada no sistema ILP *PROGOL*, que usa aprendizado por implicação lógica. Nesta metodologia um tamanho fixo para todos os samples é determinado por um limitante [35] dependente da suposição de normalidade. Além disso, o tamanho da população (número total de exemplos) deve ser conhecido para cômputo dos estimadores envolvidos, porém no contexto de streams tal parâmetro é potencialmente desconhecido.

O HTILDE é, portanto, um sistema que induz incrementalmente árvores de decisão de lógica de primeira ordem e utiliza o limitante de Hoeffding para tratar grandes fluxos de dados. Na seção 3.3.2, entraremos em mais detalhes a respeito do seu algoritmo.

3.3.2 Algoritmo

O pseudo-código do HTILDE é muito parecido com o do VFDT, mostrado no algoritmo 3.3. A diferença é que, em cada nó, em vez de se avaliar qual é o melhor atributo que pode ser colocado como teste do mesmo, são considerados os possíveis refinamentos.

Na tabela 3.1, ressaltamos as diferenças entre os algoritmos do VFDT e do HTILDE. Novamente, assim como nos capítulos 3.1 e 3.2, exibimos os pseudo-códigos em alto nível, demonstrando como ocorre a escolha de um atributo para ser colocado como teste de um nó interno nos dois algoritmos. Os passos que não estão exibidos do HTILDE (1 e 5) são iguais aos do VFDT. Já no algoritmo 3.6, detalhamos o pseudo-código do HTILDE, que utiliza de forma auxiliar a função *FazSplit*, mostrada no algoritmo 3.7.

O processo de indução do HTILDE é bastante parecido tanto com o VFDT quanto com o TILDE, pois herda suas características de ambos. Começando por uma única folha inicial, a raiz l_1, cuja consulta associada é $\leftarrow true$, os possíveis refinamentos são determinados pelo operador $\rho(.)$, linhas 1 e 2.

Analogamente ao VFDT, cada folha l possui um conjunto de contadores, n, que são identificados pelas seguintes informações: 1) o índice da própria folha, l (l_1 no

Algoritmo 3.6 Algoritmo do HTILDE, para indução de uma árvore de decisão de lógica de primeira ordem escalável para grandes fluxos dados.

Entradas:

S é uma sequência de exemplos,

$\rho(.)$ é o operador de refinamento,

$\mathbf{Q}$ é um conjunto de refinamentos,

$G(.)$ é uma função de avaliação de *split*,

δ é um menos a probabilidade desejada de se escolher o refinamento correto em um dado nó.

τ é um valor definido pelo usuário para detectar empates de refinamentos.

e_{min} é um número que indica de quantos em quantos exemplos será verificado se a folha deve sofrer um *split*.

Saída:

HT é uma árvore de decisão de lógica de primeira ordem.

Procedimento HTILDE (S, G, δ, τ, e_{min})

Seja HT uma árvore com uma única folha l_1 (a raiz).
Seja $\mathbf{Q}_1 = \rho(\leftarrow true)$.
Para cada classe c
    Para cada refinamento $Q_i \in \mathbf{Q}_1$
        Faça $n[l_1][i][Verdadeiro][c] = 0$.
        Faça $n[l_1][i][Falso][c] = 0$.
Para cada exemplo e em S
    Desça e até uma folha l usando HT.
    Faça Q_l ser a consulta associada de l.
    Seja $\mathbf{Q}_l = \rho(Q_l)$.
    Para cada refinamento $Q_i \in \mathbf{Q}_l$
        Se $\leftarrow Q_i$ for verdadeira em $e \wedge B$, então
            Incremente $n[l][i][Verdadeiro][classe(e)]$ em 1.
        Senão
            Incremente $n[l][i][Falso][classe(e)]$ em 1.
    Faça E_l ser o número de exemplos que chegaram até agora na folha l.
    Se os exemplos que chegaram até agora em l não forem todos da mesma classe e E_l módulo $e_{min} = 0$, então
        Calcule $\overline{G_l}(Q_i)$ para cada refinamento $Q_i \in \mathbf{Q}_l$ usando os contadores n.
        Faça Q_a ser o refinamento com o maior valor de $\overline{G_l}$.
        Faça Q_b ser o refinamento com o segundo maior valor de $\overline{G_l}$.
        Calcule ϵ usando a equação 3.4.
        Faça $\Delta\overline{G_l} = \overline{G_l}(Q_a)$ - $\overline{G_l}(Q_b)$.
        Se $\Delta\overline{G_l} > \epsilon$ ou $\Delta\overline{G_l} \leq \epsilon < \tau$, então
            FazSplit(HT, l, Q_a, Q_l).
Retorne HT.

VFDT	**HTILDE**
1. Raiz: *Exs* = os N primeiros exemplos de treinamento, N dado pelo limite de Hoeffding ($\Delta\overline{G} > \epsilon$ ou $\Delta\overline{G} < \epsilon < \tau$).	
2. Baseado no conjunto de exemplos *Exs*, escolha o melhor atributo A para ser colocado como teste do nó.	$\Rightarrow$ 2. Baseado no conjunto de exemplos *Exs*, escolha o melhor refinamento R para ser colocado como teste do nó.
3. Crie um ramo para cada valor vi do atributo A, correspondente ao teste $A = vi$.	$\Rightarrow$ Crie dois ramos: um correspondente ao teste $R = Verdadeiro$ e o outro a $R = Falso$.
4. Descarte os exemplos já considerados para o *split*. Divida os próximos exemplos entre os nós filhos, de acordo com o valor do atributo A, gerando os conjuntos de exemplos *Exs* de cada filho. O número de exemplos de cada filho também será determinado pelo limite de Hoeffding.	$\Rightarrow$ Descarte os exemplos já considerados para o *split*. Divida os próximos exemplos entre os nós filhos, de acordo com o valor do refinamento R, gerando os conjuntos de exemplos *Exs* de cada filho. O número de exemplos de cada filho também será determinado pelo limite de Hoeffding.
5. Repita 2 para cada filho.	

Tabela 3.1: Diferenças entre os algoritmos em alto nível do VFDT e do HTILDE.

caso da raiz); 2) pelo índice do refinamento, i; 3) um campo que pode ser $Verdadeiro$ ou $Falso$ se o refinamento provar o exemplo ou não; e 4) o índice c, que denota a classe do exemplo. Em todas folhas, estes contadores começam com valores 0, linhas 3 a 6.

Inicializada a raiz, os exemplos em S começam a ser entregues ao modelo em evolução. Ao percorrer a árvore em construção, cada exemplo $e \in S$ é então encaminhado a uma folha l adequada.

Para cada possível refinamento existente em l, atualizam-se seus contadores de acordo com o resultado do confronto entre o exemplo e e seu teste, linhas 9 a 15. Vale lembrar que os refinamentos são determinados aplicando-se $\rho(.)$ à consulta associada Q de l, quando de sua criação.

Atualizados os contadores, deve-se verificar se um split já é possível. Primeiramente, testa-se se um certo número de exemplos, e_{min} já chegou em l, linha 17. Em caso negativo, espera-se por mais exemplos; em caso positivo procede-se o cômputo da heurística para os refinamentos, linha 18.

Feito isso, determinam-se os dois melhores candidatos a teste para o split e então o limitante de Hoeffding é computado, linhas 19 a 21. Caso a diferença tenha ultrapassado o limitante, ou seja suficientemente pequena, então l sofrerá um

split (procedimento FazSplit(.)), linhas 22 e 23. Caso contrário, espera-se por mais exemplos.

Ao término do split, l deixa de ser uma folha e passa a ser um nó interno com dois filhos. Conforme dito, o procedimento de split está detalhado no algoritmo 3.7, a ser explicado em seguida.

Algoritmo 3.7 Função *FazSplit*, chamada pelo HTILDE, cujo pseudo-código é exibido no algoritmo 3.6.

Entradas:

HT é a árvore lógica de primeira ordem que vai ser modificada,
l é a folha de HT que vai sofrer o *split*,
Q_a é o melhor refinamento para o nó l,
Q_l é a consulta associada do nó l.

Saída:

HT, que é a árvore de decisão de lógica de primeira ordem, tendo sofrido o *split* no nó l.

Procedimento FazSplit (HT, l, Q_a, Q_l)

1: Faça $conj = Q_a - Q_l$.
2: Substitua l por um nó interno cujo teste é a conjunção de literais $conj$.
3: Para cada ramo do *split*
4: Adicione uma nova folha l_m.
5: Se for o ramo esquerdo, então
6: Faça $\mathbf{Q}_m = \rho(Q_a)$.
7: Senão (ou seja, se for o ramo direito)
8: Faça $\mathbf{Q}_m = \rho(Q_l)$.
9: Para cada classe c
10: Para cada refinamento $Q_i \in \mathbf{Q}_m$
11: Faça $n[l_m][i][Verdadeiro][c] = 0$.
12: Faça $n[l_m][i][Falso][c] = 0$.
13: Retorne HT.

Dados: o nó l, que é uma folha; seu melhor refinamento, Q_a; e sua consulta associada Q_l; a conjunção $conj$ é obtida e armazenada em l, que se torna um nó interno com dois filhos, linhas 1 e 2.

A consulta associada do filho esquerdo é a própria consulta do melhor refinamento, Q_a, enquanto que a consulta associada do filho direito é Q_l, i.e., a consulta associada de l. Dado isto, os candidatos a refinamentos de cada filho são determinados por $\rho(.)$, linhas 3 a 8.

Em cada filho, os contadores de cada candidato é inicializado com valor 0, linhas linhas 9 e 12. Finalmente, a árvore modificada, HT, é retornada ao processo que chamou este procedimento.

Finalizamos a seção exibindo o algoritmo 3.8, em que mostramos o pseudo-código utilizado para classificar um exemplo usando o HTILDE. Ele é muito parecido com

o algoritmo 3.2, que classifica um exemplo com o TILDE. A diferença agora é que pode ocorrer o caso em que uma folha não tenha exemplos, ou seja, ela foi criada por um *split* do nó pai, mas depois nenhum outro exemplo de treinamento chegou nela. Para contornar estes casos e podermos classificar um exemplo que venha a cair neste tipo de folha, estabelece-se como classe padrão de uma folha a classe que o seu nó pai tinha antes de sofrer o *split*. Esta estratégia é semelhante à utilizada pelo VFDT nesses casos.

Algoritmo 3.8 Algoritmo de classificação de um exemplo usando o HTILDE (com um conhecimento preliminar B)

Entrada:
e é um exemplo.

Saída:
c é uma classe.

Procedimento Classifica (e)

1: Seja Q a consulta Prolog $Verdadeiro$.
2: Seja N o nó raiz da árvore.
3: Enquanto N não for uma folha
4: Seja $N =$ **noInterno**($conj$, esq, dir).
5: Se $Q \wedge conj$ for verdadeira em $e \wedge B$, então
6: Faça $Q = Q \wedge conj$.
7: Faça $N = esq$.
8: Senão
9: Faça $N = dir$.
10: Faça $n =$ número de exemplos de treinamento que caíram na folha N.
11: Se $n = 0$, então
12: Faça $c =$ a classe do nó pai de N, antes de sofrer o *split*.
13: Senão
14: Faça $c =$ a classe mais frequente entre os exemplos de treinamento que caíram nó N.
15: Retorne c.

O processo de predição do HTILDE é bastante semelhante ao do TILDE. Porém, como existe a possibilidade de uma folha não receber exemplo algum após sua criação, deve-se armazenar numa folha recém-criada a predição de seu pai, até que algum exemplo chegue. Esta situação diferente é contemplada nas linhas 11 e 12.

3.3.3 Especificação de um problema

Antes de concluirmos o capítulo, é interessante fazer um comentário sobre como um problema é representado no HTILDE. Como ele foi desenvolvido no mesmo ambiente do TILDE, um problema é especificado de maneira muito semelhante nos

dois sistemas: através dos arquivos *.kb*, *.bg* e *.s*, que foram explicados no capítulo 3.1.

Para utilizar o HTILDE, algumas pequenas mudanças devem ser feita no arquivo *.s*. O usuário deve incluir três linhas específicas para definir os parâmetros utilizados pelo algoritmo: *htilde_delta*, *htilde_emin* e *htilde_tau*. Estes comandos definem, respectivamente, os parâmetros δ, e_{min} e τ do algoritmo. Para começar a executá-lo, deve-se utilizar o comando *execute(induce(htilde))*. Finalmente, vale ressaltar que a heurística é a mesma usada como padrão no VFDT, que é o ganho de informação. Como esta não é a medida padrão do TILDE, que é a taxa de ganho de informação, deve-se colocar mais um comando, *heuristic(gain)*, para definir que deseja-se usar o ganho de informação como heurística. Um exemplo destes comandos podem ser vistos a seguir:

```
htilde_delta(0.000001).
htilde_emin(300).
htilde_tau(0.05).
heuristic(gain).
execute(induce(htilde)).
```

Além das alterações no arquivo *.s*, pode ser necessário incluir novas linhas no arquivo *.bg* para informar quais predicados são simétricos. O tratamento de predicados simétricos é importante pois pode gerar empates inquebráveis, impedindo o processo de split. O tratamento deste problema é detalhado em [9].

Capítulo 4

HTILDE: Novos resultados

Neste capítulo, apresentaremos contribuições complementares ao trabalho de Lopes e Zaverucha em [9], que são relacionadas à classificação. Na seção 4.1, discutiremos o comportamento assintótico dos modelos aprendidos pelo HTILDE e, na seção 4.2, apresentaremos resultados complementares com a base de dados real CORA [36].

4.1 Propriedades das HTILDE-Trees

Enunciaremos nesta seção que as propriedades assintóticas das árvores de Hoeffding proposicionais se preservam nas árvores geradas pelo HTILDE (HTILDE-Trees) quando o aprendizado por interpretações é usado. Portanto, o teorema 4.1, a ser enunciado, estende a validade do teorema 3.1 para o aprendizado por interpretações.

Teorema 4.1. *Seja S uma sequência de infinitos exemplos, onde cada exemplo* $\mathbf{x}$ *é uma interpretação I com um número arbitrário de átomos. Seja HTT_δ uma árvore de Hoeffding de lógica de primeira ordem do HTILDE, com parâmetro δ, treinada usando S. Seja LDT_* a árvore de decisão de lógica de primeira ordem treinada com os infinitos exemplos de S. Então, analogamente ao caso proposicional, é válido que $E[\Delta_i(HTT_\delta, DT_*)] \leq \delta/p$ quando o aprendizado por interpretação é usado.*

Prova: A demonstração é bastante direta e consiste em mostrar que $P(N_i^{HTT_\delta} \neq N_i^{LDT_*}|I_{i-1}) \leq \delta$, onde δ é determinado pela equação 3.4, permanece válida. Para tal, vamos analisar a inequação $P(N_i^{HTT_\delta} \neq N_i^{LDT_*}|I_{i-1}) \leq \delta'$ no contexto do aprendizado lógico por interpretações.

Durante o treinamento, esta inequação do limitante de Hoeffding limita a probabilidade de os modelos terem escolhas diferentes no nível i, com ambos possuindo caminhos/testes iguais até o nível $i-1$. Queremos, então, argumentar que $\delta' = \delta$. Além disso, duas fórmulas f_1 e f_2 são equivalentes, denotado por $f_1 \equiv f_2$, se diferem apenas pela nomeação de suas variáveis. Vem que:

1. Como o viés de linguagem é o mesmo para ambos modelos, todo candidato a refinamento de HTT_δ terá um, e somente um, correspondente equivalente em LDT_* no nível i, e vice e versa (duas fórmulas são equivalentes se diferem apenas na nomeação de suas variáveis). Ou seja, existe uma função bijetora que relaciona os candidatos a refinamentos dos modelos num mesmo nível. Portanto, da mesma forma que no caso proposicional os algoritmos escolhiam seus testes baseados em conjuntos de atributos iguais, na representação pode interpretações, os algoritmos lógicos escolhem sobre conjuntos de refinamentos equivalentes.

2. Como o background knowledge é o mesmo para ambos algoritmos (pois é específico do problema), um refinamento de HTT_δ cobre o mesmo espaço de exemplos o seu equivalente em LDT_*, e vice e versa. Portanto, ambos particionam o espaço de exemplos da mesma forma.

3. No aprendizado por interpretações, a classe de um exemplo não é influenciada pela presença ou ausência de quaisquer outros exemplos, além disso, sua cobertura por uma cláusula depende apenas: da cláusula, dos fatos em sua interpretação e do background knowledge; que são os mesmos para ambos modelos. Logo, um exemplo, além de independente dos demais, é coberto da mesma forma por ambos modelos, quando compara-se os pares de cláusulas relacionadas pela bijeção.

4. Permanece a hipótese de distribuição estacionária do fluxo, i.e., o fluxo de dados têm distribuição igual (muito próxima) a da população original (ausência de concept drifts). Logo, em qualquer segmento do fluxo, com tamanho suficiente, a distribuição das classes terá proporção muito próxima a do dataset inteiro.

Dado o exposto, concluímos que a probabilidade de haver escolhas diferentes no nível i não sofre a influência de elementos adicionais gerados pelo aprendizado por interpretação, i.e., elementos que não estivessem presentes no caso proposicional, e, portanto, podemos computar δ' usando diretamente a expressão do limitante de Hoeffding sem alterações. Portanto, $\delta' = \delta$, onde δ é dado pela equação 3.4. Podemos afirmar, então, que a inequação $P(N_i^{HTT_\delta} \neq N_i^{LDT_*} | I_{i-1}) \leq \delta$ se preserva, da mesma forma que no paradigma proposicional.

O restante da demonstração segue da mesma forma que no caso proposicional, ou seja, conforme o restante do teorema 3.1.

4.2 Resultados com o dataset CORA

O CORA é um dataset relacional real introduzido por McCallum em [37] e está disponível em [36]. Ele contém informações sobre publicações científicas, como título, nome do autor, editora, ano de publicação. O problema proposto consiste em identificar quando duas referências estão relacionadas a um mesmo artigo.

Em relação a sua estrutura, o CORA é uma base desbalanceada cujos exemplos positivos compõem apenas 4% do total, sendo negativos os 96% restantes. Desta forma, a medida-f (f-measure) foi levada em consideração na avaliação da performance. Além disso, o dataset é composto de 5 partições (folds), de modo que cada uma possui aproximadamente 840 mil exemplos e conhecimento preliminar (background knowledge).

Em [9] executou-se variados experimentos com o CORA visando uma melhor medida-f, porém os resultados obtidos com o HTILDE não ultrapassaram 41%. Dado o exposto, estudamos a estrutura do dataset buscando um entendimento do que estava acontecendo. Concluímos que o dataset distribuído em [36] apresenta, em cada um de seus folds, a seguinte organização: uma sequência de treino com, aproximadamente, 400000 exemplos positivos seguida de 20000 negativos e uma sequência de teste com, aproximadamente, 400000 exemplos positivos seguidos de 20000 negativos. Uma vez que o HTILDE lê sequencialmente os dados, tal organização estava causando uma mudança de distribuição (concept drift) drástica nos dados, além disso este tipo de sequência dificilmente aconteceria em um fluxo real.

A partir desta descoberta, antes de rodar quaisquer experimentos, passamos a embaralhar cuidadosamente os exemplos de cada fold, o que nos levou a resultados expressivamente melhores. Antes de expormos os resultados discutiremos alguns detalhes sobre como realizamos os novos experimentos com o CORA.

Aproveitando o particionamento original, realizamos validação cruzada 5x2 [25], desta forma cada fold fornecia duas rodadas de treino/teste: na primeira rodada a primeira metade serve de treino (e restante de teste) e na segunda rodada a primeira serve de teste (e o restante de treino). As medidas colhidas foram: tempo de indução (em segundos), número de nós (regras), número de literais, acurácia, medida-f, AUC-ROC e AUC-PR. Cada bateria de 5x2 experimentos foi realizada outras 10 vezes, sempre com datasets reembaralhados, e usamos o teste t corrido [26] [27] para verificar se as diferenças entre os algoritmos eram significativas. Os resultados numéricos seguem na tabela 4.1.

O HTILDE apresentou um tempo de aprendizado 60% mais rápido, numero de nós (e portanto de regras) 3,8 vezes menor e literais 3,1 vezes menos, todos com

	TILDE	**HTILDE**
Tempos	35,3	21,1
Nós	23,9	6,2
Literais	57,7	18,4
Acurácia	0,99	0,99
Medida-F	0,94	0,90
AUC-PR	0,95	0,93
AUC-ROC	0,99	0,99

Tabela 4.1: Resultados para o dataset *CORA*

diferença estatisticamente significativa para $p < 0,001$. Adicionalmente, não houve diferenças em relação a acurácia e AUC-ROC. Porém, o TILDE conseguiu uma medida-f 0,03 melhor e AUC-PR 0,02 melhor, também para $p < 0,001$.

Dados o tamanho do dataset *CORA* e o comportamento assintótico esperado do HTILDE, tais resultados mostraram-se coerentes com os obtidos em [9] com a base BONGARD. A medida que o número de exemplos cresce as medidas de performance melhoraram. As curvas de aprendizado obtidas com o CORA estão mostradas na tabela 4.2. Para confeccionar tais curvas, construímos sub-datasets com percentuais dos exemplos originais e progressivamente aumentamos a porcentagem de exemplos para treino. Além disso, preservamos a proporção de 96% e 4% entre as classes.

%exemplos	Tempo	Nós	Lits.	Acur.	MedidaF	AUC-PR	AUC-ROC
HTILDE							
0,004	1,6	0,0	0,0	0,96	0,00	0,03	0,48
0,008	1,7	1,0	3,0	0,99	0,91	0,83	0,99
0,016	1,9	1,5	4,5	0,99	0,91	0,80	0,99
0,031	2,2	1,7	5,1	0,99	0,91	0,80	0,99
0,063	2,9	2,0	6,0	0,99	0,91	0,80	0,99
0,125	4,1	2,4	7,2	0,99	0,91	0,80	0,99
0,250	7,2	3,7	11,1	0,99	0,89	0,96	0,99
0,500	11,8	4,8	14,2	0,99	0,90	0,97	0,99
1,000	21,1	6,2	18,4	0,99	0,90	0,93	0,99
TILDE							
0,004	1,6	3,2	9,6	0,99	0,92	0,93	0,98
0,008	1,8	4,7	13,7	0,99	0,94	0,94	0,99
0,016	2,0	5,7	16,9	0,99	0,94	0,97	0,99
0,031	2,6	7,9	21,7	0,99	0,93	0,94	0,99
0,063	3,9	10,0	25,6	0,99	0,93	0,93	0,99
0,125	5,5	11,6	30,6	0,99	0,93	0,95	0,99
0,250	10,2	15,5	39,3	0,99	0,93	0,95	0,99
0,500	17,4	20,2	50,0	0,99	0,93	0,95	0,99
1,000	35,3	23,9	57,7	0,99	0,94	0,95	0,99

Tabela 4.2: CORA - Curvas de aprendizado

É fácil notar que a acurácia pouco varia, em função de o dataset ser desbalanceado, a AUC-ROC também mostrou pouca sensibilidade devido a este desbalanço, bastando-se predizer a classe mais frequente para um bom resultado. A AUC-PR, assim como a medida-f, leva em conta a precisão e a revocação, caso um classificador deixe de aprender uma das classes, influenciado pelo mal-balanceamento, estas medidas tornam-se baixas. Logo, ambas medidas mostraram-se mais confiáveis e sensíveis. Os gráficos relacionados às curvas de aprendizado para a medida-f e AUC-PR estão mostrados nas figuras 4.1 e 4.2, respectivamente.

Investigamos também como os parâmetros e_{min}, τ e δ poderiam melhorar os modelos aprendidos. Para tal, executamos validação interna 5x2 da seguinte forma: cada parâmetro foi variado de um valor mínimo a um máximo, seguindo uma certa forma de atualização, enquanto os demais foram mantidos nos valores default.

O parâmetro e_{min} foi testado com 1% e depois variado entre 5% e 100% dos exemplos, com incrementos de 5%. O parâmetro τ foi variado entre 0% e 20% com incrementos de 1% e de 20% a 100% com passos de 5%, esta última granularidade foi usada pois o parâmetro mostrou-se mais sensível na faixa de 0% a 20%. E, por fim, o parâmetro δ foi testado com os seguintes valores: $\{10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Os valores de cada parâmetro que

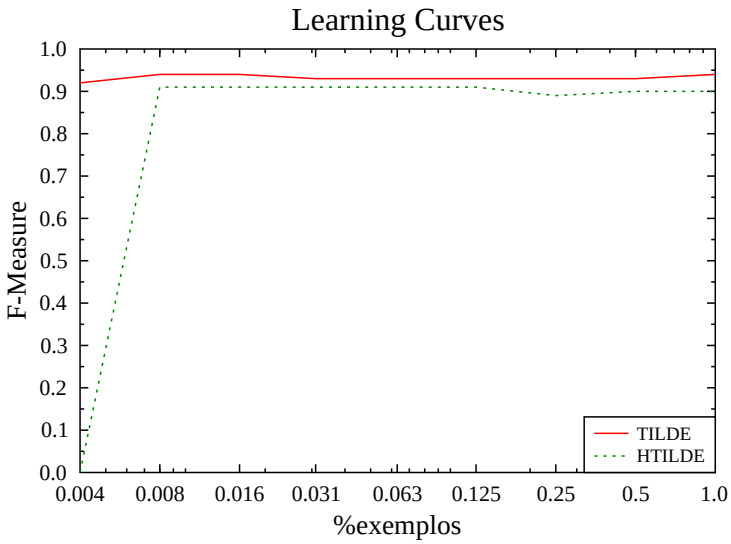


Figura 4.1: CORA: Curvas de aprendizado da medida-f

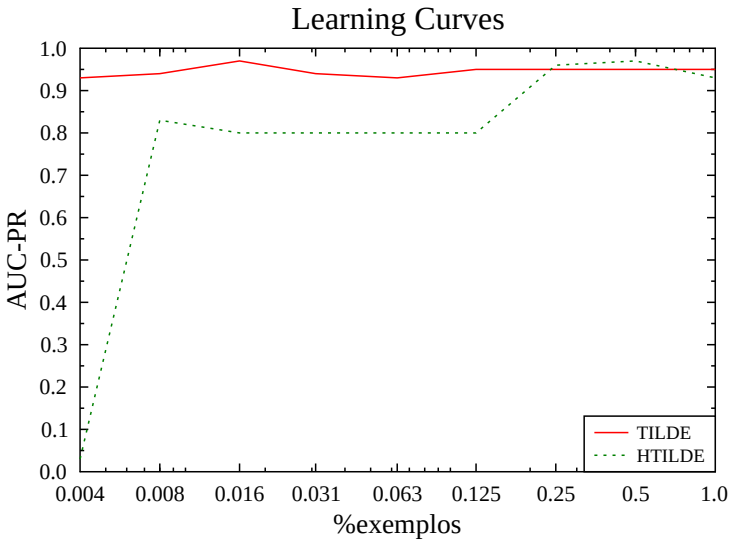


Figura 4.2: CORA: Curvas de aprendizado da AUC-PR

obtiveram os melhores resultados foram combinados com os melhores valores dos demais e novos experimentos foram executados.

Ao término dessa bateria de experimentos, verificamos que a melhor parametrização foi justamente aquela composta pelos valores default do VFDT, i.e.: $e_{min} = 300$, $\tau = 0,05$ e $\delta = 10^{-6}$, o mesmos usados em nossos experimentos. A melhor parametrização foi considerada aquela que obteve menor tempo de indução, menor tamanho dos modelos e melhores medidas de qualidade.

Capítulo 5

HTILDE-RT

Neste capítulo, apresentaremos as principais contribuições deste trabalho: o algoritmo HTILDE-RT, que será explicado na seção 5.1 e os resultados experimentais, que serão discutidos na seção 5.2.

5.1 Hoeffding TILDE Regression Trees

Nesta seção, apresentaremos o algoritmo HTILDE-RT: um algoritmo escalável para o aprendizado de árvores de regressão de lógica de primeira ordem em fluxos de dados. Este capítulo está organizado da seguinte maneira: na seção 5.1.1, apresentaremos a motivação e faremos uma introdução ao método; na seção 5.1.2, discutiremos a adaptação da heurística às condições do limitante de Hoeffding; na seção 5.1.3, discutiremos como tratamos os atributos numéricos neste trabalho; na seção 5.1.4, apresentaremos o pseudo-código do algoritmo HTILDE-RT; na seção 5.1.5, discutiremos detalhes sobre a especificação de um problema; e na seção 5.1.6, argumentaremos sobre alguns detalhes da atual implementação.

5.1.1 Introdução

O HTILDE-RT é uma extensão do HTILDE para os problemas de regressão, que une o poder do representação do TILDE-RT à eficiência do VFTD. O HTILDE-RT, herda os três parâmetros e_{min}, τ e δ do VFDT e realiza o aprendizado de seus modelos de forma bastante parecida com a do TILDE-RT, conforme veremos a diante.

As árvores de regressão em lógica de primeira ordem permitem o tratamento de problemas de regressão em ILP de forma bastante direta, além de ter a atraente propriedade de manter interpretabilidade dos modelos, como nas árvores de decisão. Além disso, tais modelos podem ter aplicações em diversas outras áreas como, por exemplo: na representação das tabelas de probabilidade condicionais das redes de

dependências relacionais (Relational Dependency Networks) [38]; e na representação de funções potenciais em campos condicionais aleatórios aplicados a problemas relacionais [39].

Uma vez que os sistemas VFDT e TILDE/TILDE-RT já foram explicados nos capítulos anteriores, o entendimento do HTILDE-RT fica bastante facilitado. Porém, neste capítulo, explicaremos de forma mais detalhada as principais diferenças, bem como argumentos os teóricos por trás do HTILDE-RT.

5.1.2 Alteração da heurística

Vimos que o TILDE-RT usa como heurística a redução na soma dos quadrados das distâncias, porém o limitante de Hoeffding precisa que a variável observada esteja delimitada em um intervalo. Para tal, aplicamos uma normalização à heurística, dando origem à seguinte heurística modificada:

$$\mathcal{H}(\rho, l) = \frac{SS(L) - (SS(N(\rho)) + SS(P(\rho))}{SS(L)} \tag{5.1}$$

O teorema 5.1 garante que $SS(L) \geq SS(N(\rho)) + SS(P(\rho))$, então sabemos que $0 \leq \mathcal{H} \leq 1$. Logo, podemos usar $R = 1$ na equação do limitante de Hoeffding, resultando em:

$$\epsilon = \sqrt{\frac{ln(1/\delta)}{2n}} \tag{5.2}$$

Vale reforçar que o *Aprendizado por Interpretações* garante a cobertura local dos exemplos. Esta condição permite o uso de amostragem uma vez que a cobertura de um exemplo não depende de outro.

Transcreveremos uma pequena demonstração de que a normalização delimita a heurística usada, portanto satisfazendo as condições necessárias do limitante de Hoeffding. Seja $S \subset \mathbb{R}$ e $y \in \mathbb{R}$, então a *soma dos quadrados das distâncias* entre o valor y e os elementos de S é dado por:

$$SS(y, S) = \sum_{i}^{|S|} (y - y_i)^2 \tag{5.3}$$

Observação: *Para simplificar, cometemos uma sobrecarga do functor $SS/2$. Porém, somente no contexto desta demonstração, o primeiro argumento refere-se a um número real, em vez de um refinamento.*

Lema 5.1. *$SS(y, S)$ tem valor mínimo quando $y = \mu(S)$.*

Prova: Tomando-se a primeira derivada temos que:

$$\frac{dSS(y,S)}{dy} = \sum_{i}^{|S|} 2(y - y_i) = 2(|S|y - \sum_{i}^{|S|} y_i) \tag{5.4}$$

Igualando a zero, vem:

$$2(|S|y - \sum_{i}^{|S|} y_i) = 0 \rightarrow y = \frac{\sum_{i}^{|S|} y_i}{|S|} = \mu(S) \tag{5.5}$$

Teorema 5.1. *Sejam S_1 e S_2 subconjuntos de S, tais que: $S_1 \cup S_2 = S$ e $S_1 \cap S_2 = \emptyset$, então: $SS(S) \geq SS(S_1) + SS(S_2)$*

Prova: Sejam: $SS(S) = SS(\mu(S), S)$; $SS(S_1) = SS(\mu(S_1), S_1)$; e $SS(S_2) = SS(\mu(S_2), S_2)$.
Temos que: $SS(\mu(S), S) = SS(\mu(S), S_1) + SS(\mu(S), S_2)$.
Mas: $SS(\mu(S), S_1) \geq SS(\mu(S_1), S_1)$, pois a média $\mu(S_1)$ é o ponto que minimiza $SS(S_1)$, e o análogo pode ser dito para $SS(S_2)$.
Então: $SS(S) = SS(\mu(S), S_1) + SS(\mu(S), S_2) \geq SS(\mu(S_1), S_1) + SS(\mu(S_2), S_2)$.
Logo: $SS(S) \geq SS(S_1) + SS(S_2)$.

5.1.3 Tratamento de atributos numéricos

O TILDE/TILDE-RT pode tratar os atributos numéricos basicamente de duas formas: 1) O usuário usa seu conhecimento sobre o domínio e fornece as constantes candidatas a split; ou 2) Aplica um pré-processamento para determinar tais constantes, que pode ser feito através de um processo de discretização (clustering) ou de simples coleta de todos os valores que ocorrem nos dados (semelhante ao C4.5). Neste trabalho, fornecemos ao HTILDE-RT as constantes candidatas a split, visto que pré-processamento não é adequado ao ambiente de fluxos, pois apenas uma passada dos exemplos é possível, além disso como apenas datasets sintéticos foram usadas, tais constantes era conhecidas.

Suponha que o problema *Milhas por Galão* fosse tratado no HTILDE-RT. Então, um predicado $acelecaracao(Ex, A)$ poderia ter as contantes candidatas a split com as seguintes declarações: $rmode((acelecaracao(+Ex, A), A < \#[10, 15, 20]))$. e $rmode((acelecaracao(+Ex, A), A > \#[20]))$..

A variável $+Ex$ indica que o identificador do exemplo é uma variável de entrada, a variável A será unificada com a aceleração do exemplo analisado, sendo comparada através do operador $<$ com as constantes 10 e 15 e através do operador $>$ com a constante 20. Gerando os seguintes literais, que poderão ser acrescentados

aos refinamentos: $acelecaracao(+Ex, A), A < 10$, $acelecaracao(+Ex, A), A < 15$, $acelecaracao(+Ex, A), A < 20$ e $acelecaracao(+Ex, A), A > 20$.

É possível ainda usar uma forma mais avançada de declaração dos rmodes. Tal forma assemelha-se a estratégia do VFDT, em que uma certa quantidade de valores que ocorrem nos exemplos de uma folha são usados como thresholds. A construção possui o seguinte padrão: "rmode(N : #(n*m*V : conj1, conj2))."; onde:
N - é o número de vezes que a conjunção $conj2$ pode ser usada em um caminho entre a raiz e uma folha;
n - é o número máximo de exemplos a ser considerado dentre os exemplos da folha;
m - é o número máximo de ocorrências do padrão $conj1$ por exemplos;
V - é uma lista de variáveis que ocorre em $conj1$, cujas unificações com os exemplos retornarão as possíveis constantes;
conj1 - é o padrão a ser procurado nos exemplos;
conj2 - é a conjunção final, que poderá ser acrescentadas aos refinamentos.

Por exemplo, se o problema MPG fosse adaptado ao formato das interpretações, cada exemplo possuiria um fato sobre o atributo cilindradas: "$cilindradas(X)$.". Então, ao tentar refinar a raiz, a declaração:
"$rmode(1 : \#(3 * 1 * C : cilindradas(C), (cilindradas(+X), X < C)))$.";
implicaria que as conjunções: (cilindradas(+X), $X < 429$)), (cilindradas(+X), $X < 454$)), (cilindradas(+X), $X < 455$)); poderiam ser usadas nos refinamentos, cada uma no máximo $N = 1$ vezes.

Ainda neste exemplo: o padrão $conj1 = cilindradas(C)$ deve ser procurado nos exemplos; os $n = 3$ primeiros exemplos são usados para retornar possíveis valores de C; apenas $m = 1$ ocorrências por exemplo devem ser consideradas, maiores detalhes sobre a geração de constantes podem ser obtidos em [8].

5.1.4 Algoritmo

Antes de apresentarmos os algoritmos, é importante explicar a seguinte notação: cada folha l da árvore tem um contador $contador[l]$, que armazena estatísticas globais dos exemplos observados no vetor $contador[l].estatisticas$. Este vetor guarda as seguintes informações: a componente $contador[l].estatisticas[0]$ conta o número de exemplos que já passaram por l; a componente $contador[l].estatisticas[1]$ armazena a soma dos valores do atributo alvo y nos exemplos de l; e a componente $contador[l].estatisticas[2]$ armazena a soma dos quadrados dos valores do atributo alvo, i.e. y^2. Portanto, uma vez que o algoritmo preditivo 5.3 usa o valor médio de y como saída para uma nova observação que chegue em l, para computar a média em l basta usar: $contador[l].estatisticas[1]/contador[l].estatisticas[0]$.

Além disso, cada folha tem contadores adicionais associados a cada um de seus possíveis refinamentos, denotados conforme a seguinte notação: cada refinamento i de uma folha l tem um contador para os exemplos que sucedem no teste de i: $contador[l][i][true].estatisticas$; e, adicionalmente, um contador para os exemplos que falham no teste de i: $contador[l][i][false].estatisticas$. Quando l é criada, todos os seus contadores são inicializados com a tripla $\langle 0, 0, 0\rangle$, que significa: zero exemplos, soma dos valores de y igual a zero e soma dos quadrados dos valores de y igual a zero.

O pseudo-código do procedimento de indução é mostrado no algoritmo 5.1. Começa-se com uma folha única l_1, a raiz, e sua consulta associada é a consulta vazia, $Q_1 = true$. Então, através dos literais (ou conjunções) providas no viés de linguagem do problema, todos os possíveis refinamentos são gerados pelo operador de refinamento ρ, o qual cria uma nova consulta para cada possibilidade de literal a ser adicionado, linha 2. Uma vez que nenhum exemplos passou por l_1, as estatísticas globais, e de todos os refinamentos, são inicializadas com $\langle 0, 0, 0\rangle$, linhas 3 a 6.

Em seguida, os exemplos no fluxo S começam a ser entregues, um a um, para o modelo em aprendizado, linha 7. O exemplo e é levado até uma folha da árvore, o que é feito através da cobertura dos fatos que compõem sua interpretação com as consultas associadas de cada nó presente no caminho entre a raiz e a folha adequada, linha 8. No caso da raiz vazia, qualquer exemplo, inclusive e, sucede na consulta associada de l_1, $true$.

Então, as estatísticas globais de l_1, contador[l_1].estatisticas, são atualizadas. Se y é o atributo alvo de e, então:
contador[l_1].estatisticas[0] := contador[l_1].estatisticas[0]+1;
contador[l_1].estatisticas[1] := contador[l_1].estatisticas[1]+y;
contador[l_1].estatisticas[2] := contador[l_1].estatisticas[2]+y^2;
conforme a linha 12. A mesma atualização é feita nos contadores de cada refinamento de acordo com a cobertura de e: Se Q_i cobrir e, então $contador[l][i][true].estatisticas$ é atualizado de forma semelhante, caso contrário, $contador[l][i][false].estatisticas$ é atualizado, linhas 13 a 17.

Este processo de cobertura é altamente otimizado pelo uso dos pacotes de cláusulas, pois nume folha todos os refinamentos possuem o mesmo prefixo (sua consulta associada). Conforme a árvore cresce, sua profundidade de suas folhas também aumenta e o número de literais nas respectivas consultas associadas, portanto, a medida que o processo de aprendizado se desenvolve os pacotes proporcionam progressivamente mais economia.

Em cada momento que um novo exemplo chega em uma folha l, ela precisa verificar se e_{min} novos exemplos já foram observados, em outras palavras, ela verifica se já pode recomputar as heurísticas para cada refinamento e verificar se pode haver

split, linha 19. Caso não tenham chegado exemplos suficientes, então simplesmente espera-se por mais, caso contrário, as heurísticas dos refinamentos são computadas usando a equação 5.1. Em função do uso dos contadores, toda informação necessária para tal tarefa está prontamente disponível, linha 21.

Finalmente, deve-se verificar se a diferença entre os dois melhores refinamentos é maior que o limitante de Hoeffding, ou se é menor que o limiar de desempate τ. O limitante é computado usando a equação 5.2, que utiliza δ e o número de exemplos $|E_l|$, linhas 21 até 26. Se uma das duas condições estiver satisfeita, então a folha sofrerá split e tornar-se-á um nó interno cujo teste é Q_a, senão a folha deverá esperar por mais e_{min} novos exemplos até que o processo de verificação se repita. A notação $|E_l|$ é usada para tornar a leitura mais clara, pois, na verdade, basta consultar $contador[l].estatisticas[0]$ para saber o número de exemplos que passaram por l.

O algoritmo do HTILDE-RT guarda semelhanças do HTILDE. De fato, as principais mudanças no aprendizado estão: na heurística usada, que deve ser adequada ao caso contínuo, e ao passo extra de normalização, que acarreta apenas o custo de uma operação extra de divisão; e no acúmulo das estatísticas em vez de contagem das classes.

A função FazSplit está detalhada no algoritmo 5.2. Provida do melhor refinamento e a consulta associada de l, a função primeiramente identifica a conjunção de literais que havia sido adicionada em Q_l para gerar Q_a. Esta é uma tarefa fácil, uma vez que Q_l é prefixo de Q_a, linha 1.

Em seguida, l é convertida em um nó interno, cujo teste é $conj$, e seus contadores são descartados, linha 2.

Duas novas folhas são criadas para cada ramo: a folha esquerda ficará associada aos exemplos que sucederem no teste de seu pai, desta forma sua consulta associada será o melhor refinamento de seu pai (agora o teste do pai) e seus candidatos a refinamentos serão obtidos através de $\rho(Q_a)$, linha 6; a folha da direita ficará associada aos exemplos que falham no teste Q_a, porém a antiga consulta associada de l é válida nestes exemplos (a antiga consulta associada de l, quando era uma folha, era o teste de seu pai, que agora é avô da nova folha direita de l), portanto a consulta associada da folha direita será a mesma do avô Q_l e seus refinamentos serão obtidos por $\rho(Q_l)$, linha 8.

Note que este passo garante que novas variáveis que apareçam em Q_a não serão propagadas ao ramo direito, isto é importante pois elas não teriam significado para os exemplos que caíssem nesta folha.

Além disso, cada nova consulta tem seus contadores inicializados com a tupla $\langle 0, 0, 0 \rangle$, linhas 11 e 12. Finalmente, o novo modelo é retornado ao processo incremental de indução.

Algoritmo 5.1 Algoritmo HTILDE-RT, para indução de uma árvores de regressão de LPO em streams.

Entradas:

S é um fluxo de exemplos (interpretações);
$\rho(.)$ é um operador de refinamento;
$\mathbf{Q}$ um conjunto de possíveis refinamentos;
$\mathcal{H}(.)$ uma função de heurística para avaliar os refinamentos;
δ é um menos a probabilidade desejada de escolher
o teste correto em qualquer nó;
τ é o limiar de desempate definido pelo usuário;
e_{min} é o número mínimo de exemplos entre cada cômputo das heurísticas.

Saída: $HTRT$, uma árvore de regressão de lógica de primeira ordem.

Procedimento HTILDERT (S, $\mathcal{H}$, δ, τ, e_{min})

1: Seja $HTRT$ uma árvore com uma folha única l_1 (a raiz).
2: Seja $\mathbf{Q}_1 = \rho(\leftarrow true)$.
3: Faça $contador[l_1].estatisticas = \langle 0, 0, 0 \rangle$.
4: Para cada refinamento $\leftarrow Q_i \in \mathbf{Q}_1$
5: Faça $contador[l_1][i][true].estatisticas = \langle 0, 0, 0 \rangle$.
6: Faça $contador[l_1][i][false].estatisticas = \langle 0, 0, 0 \rangle$.
7: Para cada exemplo $e \in S$
8: Desça e até a folha l usando $HTRT$.
9: Seja $\leftarrow Q_l$ a consulta associada de l.
10: Seja $\mathbf{Q}_l = \rho(\leftarrow Q_l)$.
11: Seja y o atributo alvo de e
12: Atualize $contador[l].estatisticas$ com y.
13: Para cada refinamento $\leftarrow Q_i \in \mathbf{Q}_l$
14: Se $\leftarrow Q_i$ sucede a partir de $e \wedge B$, então
15: Atualize $contador[l][i][true].estatisticas$ com y.
16: Senão
17: Atualize $contador[l][i][false].estatisticas$ com y.
18: Seja E_l o conjunto atual dos exemplos de l.
19: Se $|E_l| \ mod \ e_{min} = 0$, então
20: Para cada refinamento $\leftarrow Q_i \in Q_l$
21: Compute $\overline{\mathcal{H}}(Q_i, l)$ usando a equação 5.1 e $|E_l|$
22: Seja $\leftarrow Q_a$ o refinamento de maior $\overline{\mathcal{H}}$.
23: Seja $\leftarrow Q_b$ o refinamento com o segundo maior $\overline{\mathcal{H}}$.
24: Compute ϵ usando a equação 5.2.
25: Seja $\Delta\overline{\mathcal{H}} = \overline{\mathcal{H}}(Q_a, l) - \overline{\mathcal{H}}(Q_b, l)$.
26: Se $\Delta\overline{\mathcal{H}} > \epsilon$ ou $\Delta\overline{\mathcal{H}} \leq \epsilon < \tau$, então
27: FazSplit($HTRT$, l, Q_a, Q_l).
28: Retorne $HTRT$.

Algoritmo 5.2 Função *FazSplit*, chamada pelo HTILDE-RT, cujo pseudo-código é exibido no algoritmo 5.1.

Entradas:

$HTRT$ é a árvore de regressão de LPO a ser modificada;
l é uma folha de $HTRT$ que sofrerá split;
Q_a é o melhor refinamento para o nó l;
Q_l é a consulta associada do nó l.

Saída: $HTRT$, que é a árvore de regressão de LPO que sofreu split no nó l.

Função FazSplit ($HTRT$, l, Q_a, Q_l)

1: Seja $conj = Q_a - Q_l$.
2: Substitua l por um nó interno cujo teste é $conj$.
3: Para cada ramo do split
4: Adicione uma nova folha l_m.
5: Se l_m é do ramo esquerdo, então
6: Faça $\mathbf{Q}_m = \rho(\leftarrow Q_a)$.
7: Senão (l_m é do ramo direito)
8: Faça $\mathbf{Q}_m = \rho(\leftarrow Q_l)$.
9: Seja $contador[l_m].estatisticas = \langle 0, 0, 0 \rangle$.
10: Para cada refinamento $\leftarrow Q_i \in Q_m$
11: Faça $contador[l_m][i][true].estatisticas = \langle 0, 0, 0 \rangle$
12: Faça $contador[l_m][i][false].estatisticas = \langle 0, 0, 0 \rangle$
13: Retorne $HTRT$.

O processo de predição de exemplo e, cujo atributo alvo é não observado, está mostrado no algoritmo 5.3. Ele consiste em fazer e percorrer a árvore até uma folha adequada l, linha 3, iterativamente testando o exemplo contra a consulta associada de cada nó no caminho da raiz a folha, linhas 4 a 9. As estatísticas armazenadas no contador global da folha são então usadas para obter o valor de saída. O valo predito é o valor médio do atributo alvo observado nos exemplos de treino que alcançaram l. Desta forma, a saída é obtida por: $contador[l].estatisticas[1]/contador[l].estatisticas[0]$, linha 10.

Algoritmo 5.3 Algoritmo de predição de um exemplo usando o HTILDE-RT

Entrada:

e é um exemplo.

Saída:

$\overline{y}$ é a média que ficou guardada na folha alcançada por e.

Procedimento Prediga (e)

1: Seja Q a consulta Prolog $true$.
2: Seja N o nó raiz da árvore.
3: Enquanto N não for uma folha
4: Seja $N =$ **noInterno**($conj$, esq, dir).
5: Se $Q \wedge conj$ for verdadeira em $e \wedge B$, então
6: Faça $Q = Q \wedge conj$.
7: Faça $N = esq$.
8: Senão
9: Faça $N = dir$.
10: Faça $\overline{y} = contador[N].estatisticas[1]/contador[N].estatisticas[0]$
11: Retorne $\overline{y}$.

5.1.5 Especificação de um problema

Para especificar-se um problema a ser resolvido pelo HTILDE-RT, basta usar os recursos já explicados nas seções anteriores. Para tal, deve-se acrescentar as seguintes declarações no arquivo de configurações ".s"do problema:

1. Pedir a carga do módulo do HTILDE: *load_package(htilde).*;
2. Determinar o predicado alvo e sua variável de saída: *predict(predicadoAlvo(+,...,-)).*;
3. Determinar que deve ocorrer uma regressão: *tilde_mode(regression).*;
4. Especificar o viés de linguagem através das declarações dos "rmodes"e dos tipos;
5. Especificar os parâmetros δ, e_{min} e τ, aqui exemplificados com seus valores default:

(a) *htilde_delta(0.000001).*
(b) *htilde_emin(300).*
(c) *htilde_tau(0.05).*

6. Determinar que a indução deve ser feita com o HTILDE: *execute(induce(htilde)).*

Além disso, deve-se prover o arquivos ".kb"que contém os exemplos, o arquivo ".bg"com o background knowledge (opcional).

É possível ainda fazer uso de outras funcionalidades do sistema como padrões de saída para os modelos, c45 ou Prolog, uso de validação cruzada, dentre outros. Estas e mais funcionalidades estão detalhadas em [8].

5.1.6 Aspectos de implementação

O HTILDE-RT foi implementado como parte do sistema ILP ACE. Embora o HTILDE-RT tenha a facilidade de usar funcionalidades providas pelo sistema ACE, tal sistema é um ambiente ILP genérico, que suporta outros algoritmos de aprendizado, não sendo verdadeiramente otimizado para tratar dados infinitos. De fato, o sistema armazena listas de inteiros que indexam os exemplos muitas outras informações que crescem na memória conforme o número de exemplos.

Portanto, dependendo do número de exemplos, o sistema por simplesmente abortar por falta de memória. Então, sob um ponto de vista teórico, o HTILDE-RT tem a capacidade de aprender de uma massa de dados potencialmente infinita, porém, em termos práticos, o ambiente em que foi desenvolvido não suporta tal tipo de processamento.

Durante as tentativas de contornar tais limitações, interagirmos diversas vezes com os desenvolvedores do sistema ACE, que, apesar de nos proporcionarem enorme ajuda com sistema, concluíram conosco que tais limitações somente podem ser rompidas numa implementação dos algoritmos HTILDE/HTILDE-RT que seja independe do sistema ACE.

5.2 Resultados Experimentais

Nesta seção, apresentaremos os resultados experimentais obtidos com o HTILDE-RT. Na seção 5.2.1, explicaremos a metodologia aplicada nos experimentos com todos os datasets. Na seção 5.2.2, cada dataset será introduzido e os respectivos resultados serão expostos e analisados. Nas seções 5.2.2 e 5.2.2, serão apresentados dois datasets relacionais, e nas seções 5.2.2, 5.2.2 e 5.2.2, serão expostos resultados em datasets proposicionais complementares.

5.2.1 Metodologia

Conforme constatado em [40], existe uma escassez de datasets de regressão relacionais publicamente disponíveis. Constatamos, também, que os datasets proposicionais disponíveis não possuem uma quantidade de exemplos suficientemente grande que justifique o uso do HTILDE-RT e tampouco um gerador de exemplos pode ser construído para torná-los maiores. Em função do exposto, utilizamos cinco datasets artificiais. Construímos programas geradores de exemplos para os dois datasets sintéticos relacionais introduzidos em [40] e para três datasets proposicionais adicionais: FRIEDMAN, MV e 2D-Planes; expostos em [41].

Todos os datasets foram gerados com 2 milhões de exemplos. Cada gerador foi implementado de acordo com as leis de formação específicas de cada dataset. Adicionalmente, curvas de aprendizado foram traçadas, monitorando como as medidas de interesse se comportaram conforme a ordem de grandeza do número de exemplos variou. Como os datasets são sintéticos os thresholds para tratamento dos atributos contínuos eram conhecidos e, portanto, foram fornecidos no viés de linguagem de cada dataset.

Para cada dataset, foi executada validação cruzada 5x2 [25] externa. As medidas de interesse coletadas foram: Tempo de aprendizado em segundos, *Tempo*; tamanho das árvores em número de nós, *Nós*; número de literais na teoria, *Literais*; coeficiente de correlação de Pearson, *Pearson*; e a raiz quadrada do erro quadrático médio, *RMSE*.

O algoritmo TILDE-RT foi usado para comparação, com todos os seus parâmetros deixados com o seus valores padrão. Convém citar que o HTILDE-RT herda os parâmetros do TILDE-RT, excetuando critérios de parada e poda. Os valores dos parâmetros do HTILDE-RT foram mantidos em configuração padrão igual a do *VFDT* [4]: $e_{min} = 300$, $\delta = 10^{-6}$ e $\tau = 0,05$. É importante salientar que ambos algoritmos funcionam dentro do mesmo sistema ILP, ACE, o que torna a comparação das medidas mais imunes a elementos externos.

Para mostrar que resultados melhores poderiam ser obtidos, executamos validação interna 5x2, nos datasets relacionais, *Artificial1* e *Artificial2*. O procedimento adotado para validação cruzada interna variou cada parâmetro de acordo com a seguinte estratégia:

- e_{min} foi variado no intervalo $[150, NbEx]$, dobrando seu valor a cada passo;
- δ foi variado no intervalo $[10^{-8}, 10^{-1}]$, sendo multiplicado por 10 a cada passo;
- τ foi variado no intervalo $[0.1, 0.9]$, com passos de $0,05$, adicionalmente foram usados os valores $\{0.01, 0.05, 0.95, 0.99\}$. [1]

[1] Excepcionalmente neste parágrafo, usamos o ponto como separador decimal para não haver

Cada parâmetro foi variado mantendo os dois restantes com seus valores padrão. Os valores que obtiveram: baixo tempo de aprendizado e/ou baixo tamanho do modelo e/ou baixo erro, foram selecionados para mais rodadas de validação interna. Nestas últimas rodadas, todos os parâmetros foram variados de acordo com as possíveis combinações geradas ao cruzar-se os valores promissores de cada parâmetro.

É importante deixar claro que o objetivo da validação interna foi apenas demonstrar que melhores resultados podem ser obtidos com um ajuste dos parâmetros do HTILDE-RT. Entretanto, como não fizemos validação interna para aprender os parâmetros do TILDE-RT, não seria justo usar os resultados obtidos após a validação interna do HTILDE-RT com os resultados obtidos com os valores de default do TILDE-RT, embora muitos deles sejam herdados pelo HTILDE-RT.

Após a validação externa foi executado o teste t corrigido [26] [27], usado para verificar se a diferença entre as medidas obtidas pelos algoritmos eram estatisticamente significativas. O valor de confiança do teste foi $p = 0,001$, e as hipóteses alternativas foram as seguintes:

1) O HTILDE-RT aprende os modelos mais rápido que o TILDE-RT;
2) O HTILDE-RT tem medidas de erro (Pearson/RMSE) melhores que o TILDE-RT;
3) O HTILDE-RT produz modelos menores que o TILDE-RT;

enquanto que a hipótese nula do teste foi: Ambos algoritmos são iguais.

5.2.2 Datasets

Artificial1

Este dataset possui dois atributos independentes: $x(X)$ e $y(Y)$, cujo argumento é uma variável real. Cada exemplo contém um fato sobre o predicado $x/1$ oito fatos sobre o predicado $y/1$, que podem ser resumidos através de agregações. O valores numéricos foram obtidos de forma aleatória, de uma distribuição uniforme entre 0 e 10. O predicado alvo, $t(T)$, tem o valor de seu argumento determinado de acordo com a função descrita na figura 5.1.

$x(X), X < 5?$
$+sim : maxY|y(Y) < 9?$
$| + sim : 4x(X) + 6avgY|y(Y)$
$| + nao : 3maxY|y(Y) + 1$
$+nao : avgY|y(Y) < 4?$
$| + sim : 3x(X) + 4$
$| + nao : x(X) - 2maxY|y(Y) + 3avgY|y(Y)$

Figura 5.1: Predicado alvo para *Artificial1*

confusão com a vírgula, que é usada nas representações de conjunto e intervalo

Adicionalmente, geramos uma segunda versão deste dataset acrescentando ruído gaussiano (*Artificial1**), que consistiu na adição de um valor ao predicado alvo, sorteado aleatoriamente de uma distribuição normal: $N \sim \sigma(0, 1)$. Na tabela 5.1 estão expostos os resultados obtidos.

	Tempos (s)	**Nós**	**Literais**	**Pearson**	**RMSE**
Artificial1					
TILDE-RT	2779,2	626,5	1253,0	1,00	1,49
HTILDE-RT	533,9	208,5	417,0	1,00	1,53
HTILDE-RT†	474,6	144,5	289,0	1,00	1,58
Artificial1*					
TILDE-RT	2756,6	560,5	1121,0	0,99	2,49
HTILDE-RT	534,3	205,0	410,0	0,99	2,53
HTILDE-RT†	456,8	143,5	287,0	0,99	2,58

Tabela 5.1: Resultados para *Artificial1*

O HTILDE-RT foi cinco vezes mais rápido que o TILDE-RT e gerou um modelo três vezes menor, além de não demonstrar perda em relação ao coeficiente Pearson. De acordo com o teste t, estas diferenças foram estatisticamente significativas para $p < 0,001$. Entretanto, o HTILDE-RT obteve um discreto aumento no RMSE de 2%.

O ruído afetou discretamente ambos algoritmos, porém de forma semelhante. Uma discreta redução no tamanho dos modelos, ou mesmo dos tempos, era esperada. Este fenômeno se deve ao fato de ambos algoritmos terem que adiar suas decisões para melhor ajudarem-se aos dados, consumindo mais exemplos por split.

Conforme dito, para mostrarmos que era possível melhorar os resultados, performamos uma validação interna, obtendo os parâmetros: $e_{min} = 2400$, $\delta = 10^{-8}$ e $\tau = 0,05$. Os resultados obtidos com tal configuração estão mostrados nas entradas marcada com *HTILDE-RT*†. A nova configuração causou um aumento de 2% no RMSE, entretanto uma redução de ao menos 11% no tempo de indução e 30% no tamanho dos modelos sem afetar o coeficiente Pearson.

A curva de aprendizado manteve o padrão apresentado em [9], mostrando que o HTILDE-RT tem uma curva de aprendizado mais lenta, porém os resultados de menor acurácia estão presentes nos datasets cuja quantidade de exemplos não justifica seu uso. Esta situação pode ser contornada usando a técnica de bootstrap mostrada [4]: treina-se um modelo batch com uma quantidade de pequena de exemplos, após certo ponto o modelo passa a evoluir como stream. Tal padrão de aprendizado foi verificado tanto no dataset original quanto no ruidoso. Convém notar que com 1000 exemplos, o HTILDE-RT nem mesmo chegou a realizar split (Note que o

parâmetro e_{min} impõe que somente a cada 300 exemplos observados as heurísticas para os refinamentos sejam recomputadas).

Nas tabelas 5.2 e 5.3 estão as curvas de aprendizado para ambos algoritmos, a na Figura 5.2 há um gráfico para a evolução do RMSE.

Tabela 5.2: Curva de Aprendizado para Artificial1

#Exemplos	**Tempo (s)**	**Nós**	**Literais**	**Pearson**	**RMSE**
HTILDE					
10^3	0,0	1,0	2,0	0,85	57,06
10^4	0,7	3,0	6,0	0,95	19,32
10^5	13,7	17,5	35,0	0,98	6,35
10^6	255,4	152,5	305,0	1,00	1,55
$2 * 10^6$	533,9	208,5	417,0	1,00	1,53
TILDE					
10^3	0,7	43,5	87,0	0,99	2,43
10^4	6,2	120,5	241,0	1,00	1,70
10^5	78,8	267,5	535,0	1,00	1,52
10^6	1271,2	537,5	1075,0	1,00	1,49
$2 * 10^6$	2779,2	626,5	1253,0	1,00	1,49

Tabela 5.3: Curva de Aprendizado para Artificial1*

#Exemplos	**Tempo (s)**	**Nós**	**Literais**	**Pearson**	**RMSE**
HTILDE					
10^3	0,0	1,0	2,0	0,85	58,60
10^4	0,7	3,0	6,0	0,95	20,27
10^5	13,8	19,0	38,0	0,98	7,80
10^6	188,0	103,0	206,0	0,99	2,74
$2 * 10^6$	534,3	205,0	410,0	0,99	2,53
TILDE					
10^3	0,7	40,5	81,0	0,99	4,08
10^4	5,6	99,0	198,0	0,99	2,74
10^5	71,8	230,0	460,0	0,99	2,53
10^6	736,6	425,5	851,0	0,99	2,49
$2 * 10^6$	2756,6	560,5	1121,0	0,99	2,49

Artificial2

Este segundo dataset possui três atributos independentes: $x(X)$, $y(C, Y)$ e $z(Z)$. Cada exemplo contém: um fato sobre os predicados $x/1$ e $z/1$, cujos argumentos são reais e identicamente distribuídos entre 0 e 10; e quinze fatos sobre o predicado $y/2$,

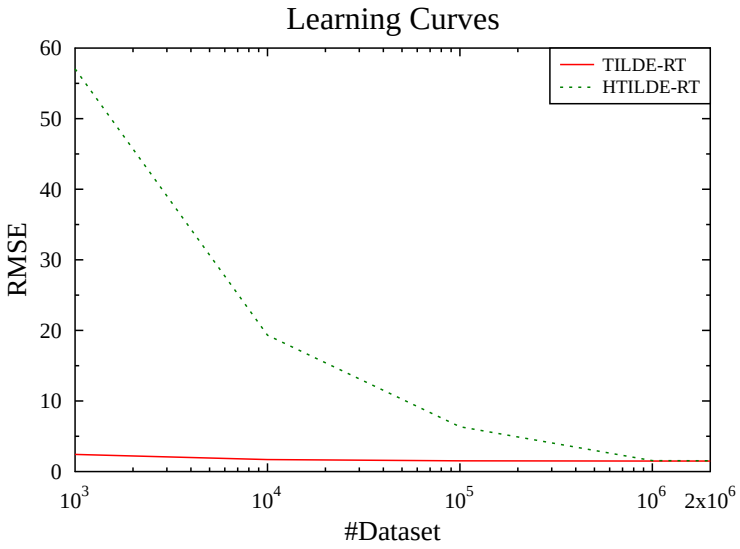


Figura 5.2: Artificial1 - Curvas de aprendizado

cujo primeiro argumento é booleano, gerado com probabilidades $P(C = true) = P(C = false) = 0,5$, e o segundo argumento é real e uniformemente distribuído entre 0 e 10. Novamente, os atributos com fatos múltiplos foram tratados com agregações. O predicado alvo, $t(T)$, é gerado de acordo com a função mostrada na figura 5.3.

$$
\begin{array}{l}
x(X), X < 6? \\
+sim : minY|y(\_, Y) < 1? \\
\ \ |+sim : 2x(X) + 3maxY|y(true, Y) + 3 \\
\ \ |+nao : 2x(X) + 5minY|y(\_, Y) \\
+nao : -2x(X) + z(Z)?
\end{array}
$$

Figura 5.3: Predicado alvo para *Artificial2*

De forma semelhante ao *Artificial1*, geramos uma segunda versão do dataset com ruído gaussiano, com distribuição $N \sim \sigma(0, 1)$, que chamamos de *Artificial2**. Na tabela 5.4 estão expostos os resultados obtidos.

Neste dataset, o HTILDE-RT conseguiu um aprendizado até $5,6$ vezes mais rápido e gerou um modelo 33% menor, sem afetar o coeficiente Pearson, significância $p < 0,001$. Novamente o ruído afetou igualmente ambos algoritmos, e de forma análoga ao exposto no Artificial1. O TILDE-RT foi melhor que o HTILDE-RT em relação ao RMSE, com $p < 0,001$, porém com diferença percentual de 1%. Os valores para o coeficiente Pearson tiveram mesmo comportamento que no *Artificial1*.

	Tempos (s)	Nós	Literais	Pearson	RMSE
Artificial2					
TILDE-RT	5453,6	302,0	604,0	0,99	1,44
HTILDE-RT	974,5	204,5	409,0	0,99	1,46
HTILDE-RT†	914,9	137,5	275,0	0,99	1,46
Artificial2*					
TILDE-RT	5437,3	272,5	544,0	0,99	2,44
HTILDE-RT	979,6	203,5	407,0	0,99	2,46
HTILDE-RT†	897,9	138,5	277,0	0,99	2,46

Tabela 5.4: Resultado para Artificial2

O processo de validação interna obteve os valores: $e_{min} = 2400$, $\tau = 0.05$ e $\delta = 10^{-8}$; cujo resultado está indicado por *HTILDE-RT*†. Desta vez, a nova configuração não causou impactos ao RMSE nem ao Pearson, porém reduziu em até 8% o tempo de indução e em 32% o tamanho das árvores.

As curvas de aprendizado mantiveram padrão semelhante ao do *Artificial1*. Nas tabelas 5.5 e 5.6 seguem as curvas das medidas colhidas para ambos datasets e na Figura 5.4 segue a curva de aprendizado do RMSE.

Tabela 5.5: Curva de Aprendizado para Artificial2

#Exemplos	Tempo (s)	Nós	Literais	Pearson	RMSE
HTILDE					
10^3	0,1	1,0	2,0	0,92	18,36
10^4	1,0	1,5	3,0	0,93	15,21
10^5	46,4	24,5	49,0	0,99	2,10
10^6	477,8	140,5	281,0	0,99	1,47
$2 * 10^6$	974,5	204,5	409,0	0,99	1,46
TILDE					
10^3	1,0	44,5	89,0	0,99	2,11
10^4	8,5	93,0	186,0	0,99	1,52
10^5	134,7	154,5	309,0	0,99	1,45
10^6	2377,3	259,5	519,0	0,99	1,44
$2 * 10^6$	5453,6	302,0	604,0	0,99	1,44

FRIED Artificial Domain

Este dataset proposicional foi introduzido por Friedman em [42]. Cada exemplo possui dez atributos (predicados) reais independentes, $x_i \mid i \in \{1, ..., 10\}$, todos obtidos com distribuição uniforme no intervalo $[0, 1]$. O atributo (predicado) alvo,

Tabela 5.6: Curva de Aprendizado para Artificial2*

#Exemplos	Tempo (s)	Nós	Literais	Pearson	RMSE
HTILDE					
10^3	0,1	1,0	2,0	0,91	19,35
10^4	1,0	1,0	2,0	0,91	19,01
10^5	39,7	23,0	46,0	0,99	3,17
10^6	479,4	131,5	263,0	0,99	2,47
$2 * 10^6$	979,6	203,5	407,0	0,99	2,46
TILDE					
10^3	1,0	42,0	84,0	0,99	3,47
10^4	8,3	90,0	180,0	0,99	2,54
10^5	130,9	140,5	281,0	0,99	2,45
10^6	2337,3	234,5	469,0	0,99	2,45
$2 * 10^6$	5437,3	272,5	544,0	0,99	2,44

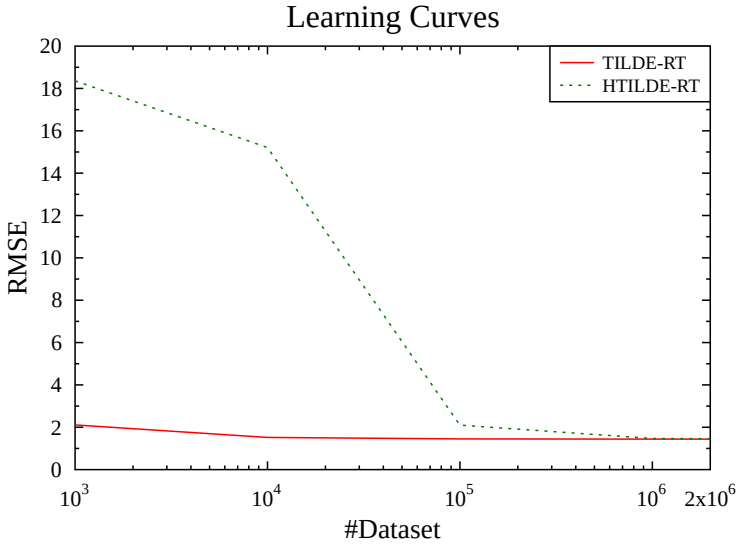


Figura 5.4: Artificial2 - Curvas de aprendizado

y, é obtido através da equação 5.6. Mais informações sobre este dataset podem ser obtidas em [41]

$$y = 10 * sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \sigma(0,1) \tag{5.6}$$

A adaptação deste datasets para interpretações de lógica de primeira ordem é bastante direta. Cada atributo x_i torna-se um predicado unário $x_i(X_i)$, cujo argumento é o valor numérico do atributo x_i, e a mesma conversão é feita para gerar o atributo alvo $y(Y)$. Além disso, não há background knowledge.

Este dataset, embora proposicional, tem a propriedade atrativa de o atributo alvo ser gerado através de uma função não trivial, além de ser um benchmark. Os resultados para o dataset *FRIED* são expostos na tabela 5.7

	Tempo (s)	Nós	Literais	Pearson	RMSE
TILDE-RT	118850,8	40376,0	80752,0	0,94	1,48
HTILDE-RT	457,5	260,0	520,0	0,94	1,51

Tabela 5.7: *FRIED*: FRIEDMAN

Neste dataset o HTILDE-RT obteve resultados expressivamente melhores, $p < 0,001$, chegando a aprender 260 vezes mais rápido com modelos 155 vezes menores, mantendo o mesmo coeficiente Pearson e RMSE discretamente aumentado em 2%.

O tamanho excessivo dos modelos do TILDE-RT está provavelmente relacionado à super-adaptação (overfitting), pois os modelos com grandes números de literais/nós sugerem uma tentativa de "memorização"dos exemplos de treino. Já o HTILDE-RT, por realizar o aprendizado através da inspeção de pequenas amostras, torna-se menos sensível a este efeito.

O grande número de nós/literais também manifestou-se nas curvas de aprendizado. Note que a ordem de grandeza do tamanho dos modelos do HTILDE-RT variou em mesma escala que a variação do número de exemplos, enquanto o TILDE-RT teve um crescimento mais elevado no tamanho de seus modelos. Seguem as curvas na tabela 5.8 e o gráfico do RMSE na Figura 5.5.

Tabela 5.8: Curva de Aprendizado para Fried

#Exemplos	Tempo (s)	Nós	Literais	Pearson	RMSE
HTILDE					
10^3	0,0	0,0	0,0	0,00	13,43
10^4	1,4	1,0	2,0	0,68	7,63
10^5	17,8	11,0	22,0	0,84	4,07
10^6	205,1	118,5	237,0	0,94	1,75
$2 * 10^6$	457,5	260,0	520,0	0,94	1,51
TILDE					
10^3	3,4	115,5	231,0	0,86	3,62
10^4	60,2	864,0	1728,0	0,91	2,41
10^5	2263,2	5276,0	10552,0	0,93	1,81
10^6	64446,5	26363,5	52727,0	0,94	1,52
$2 * 10^6$	118850,8	40376,0	80752,0	0,94	1,48

MV Artificial Domain

Este é outro dataset artificial proposicional. Os exemplos contém dez atributos, $x_i \mid i \in \{1, ..., 10\}$, interdependentes. A maneira como os valores do atributos são

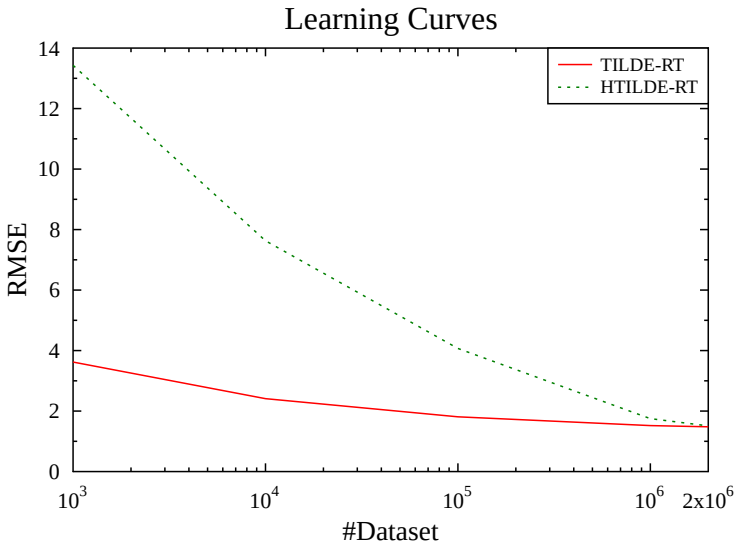


Figura 5.5: FRIED - Curvas de aprendizado

gerados, bem como a atributo alvo, y, está exposta na figura 5.6. O processo para conversão deste dataset para interpretações é o mesmo usado no dataset *FRIED*. Mais informações sobre este dataset podem ser obtidas em [41].

Embora proposicional, este dataset possui duas propriedades atrativas: a primeira é o fato de a função que gera o atributo alvo ser um conjunto de regras com formato próximo ao de uma árvore; e a segunda é o fato de os atributos preditivos estarem relacionados entre si. Os resultados obtidos para o *MV* estão expostos na tabela 5.9.

	Tempo (s)	**Nós**	**Literais**	**Pearson**	**RMSE**
TILDE-RT	1706,1	1230,5	2425,5	0,99	24,76
HTILDE-RT	354,2	230,0	453,0	0,99	24,95

Tabela 5.9: Resultados para *MV*

O HTILDE-RT conseguiu aprender $4,8$ vezes mais rápido com modelos $5,3$ vezes menores, $p < 0,001$, porém com um discreto aumento de $0,7\%$ no RMSE.

Em relação às curvas de aprendizado, o HTILDE-RT obteve uma curva mais lenta nos datasets menores, fato evidenciado principalmente pelo maior valor no RMSE. Porém, após 100000 exemplos o padrão de aprendizado seguiu o mesmo dos demais datasets. Adicionalmente, o coeficiente Pearson mostrou evolução parecida com a obtida no dataset *FRIED*. A tabela 5.10 mostra, então, as curvas de aprendizado para o conceito MV, além disso o gráfico do RMSE está exposto na figura 5.7.

Geração dos atributos:
x_1 : uniformemente distribuído em $[-5, 5]$
x_2 : uniformemente distribuído em $[-15, -10]$
x_3 : se $(x_1 > 0)$ então $x_3 = verde$
senão $x3 = vermelho$ com prob. $0,4$ e $x4 = marrom$ com prob. $0,6$
x_4 : se $(x_3 = verde)$ então $x_4 = x_1 + 2x_2$
senão $x_4 = x_1/2$ com prob. $0,3$, e $x_4 = x_2/2$ com prob. $0,7$
x_5 : uniformemente distribuído em $[-1, 1]$
x_6 : $x_6 = x_4 * \epsilon$, onde ϵ é uniformemente distribuído em $[0, 5]$
x_7 : $x_7 = sim$ com prob. $0,3$ e $x_7 = nao$ com prob. $0,7$
x_8 : se $(x_5 < 0.5)$ então $x_8 = normal$ senão $x_8 = grande$
x_9 : uniformemente distribuído em $[100, 500]$
x_{10} : inteiro uniformemente distribuído em $[1000, 1200]$

Para obter y, usar as seguintes regras:
se $(x_2 > 2)$ então $y = 35 - 0.5x_4$
senão se $(-2 <= x_4 <= 2)$ então $y = 10 - 2x_1$
senão se $(x_7 = sim)$ então $y = 3 - x_1/x_4$
senão se $(x_8 = normal)$ então $y = x_6 + x_1$
senão $y = x_1/2$

Figura 5.6: Gerador do dataset MV

2D-Planes

Este dataset é similar, porém não igual, ao descrito em [13]. Cada exemplo possui 10 variáveis independentes, $X_i \mid i \in \{1, ..., 10\}$, e uma variável dependente Y. O valor das variáveis preditivas são gerados de forma independente de acordo com as seguintes probabilidades:

Este dataset é mais simples em relação aos demais usados. Esta característica é interessante, pois permite verificar se o HTILDE-RT consegue adaptar-se aos dados tão rapidamente quanto o TILDE-RT. Devido à menor complexidade do conceito, era esperado que o TILDE-RT adaptar-se-ia rapidamente e que o HTILDE-RT conseguisse RMSE bem mais próximo ao do TILDE-RT.

O HTILDE-RT conseguiu um aprendizado $2,5$ vezes mais rápido, sem perdas em relação ao coeficiente Pearson e no RMSE, $p < 0,001$, conseguindo convergência nas duas medidas. Porém, o modelo gerado ficou 40% maior que o TILDE-RT. A ausência de perdas no RMSE evidenciou o fato esperado de que o HTILDE-RT conseguiria maior proximidade com os resultados do TILDE-RT, o que foi alcançado nos experimentos, com uma convergência rápida.

Para mostrarmos que é possível melhorar os modelos aprendidos, aplicamos diretamente a configuração dos parâmetros obtida na validação interna do *Artificial2*

Tabela 5.10: Curva de Aprendizado para MV

#Exemplos	Tempo (s)	Nós	Literais	Pearson	RMSE
HTILDE					
10^3	0,0	0,0	0,0	0,00	890,41
10^4	0,3	1,0	2,0	0,50	705,61
10^5	4,9	11,0	18,0	0,98	41,67
10^6	167,1	119,5	231,5	0,99	25,25
$2 * 10^6$	354,2	230,0	453,0	0,99	24,95
TILDE					
10^3	1,5	75,0	142,5	0,98	37,65
10^4	10,1	276,0	536,5	0,98	27,88
10^5	74,7	742,0	1452,5	0,99	25,91
10^6	738,5	1045,5	2059,5	0,99	24,80
$2 * 10^6$	1706,1	1230,5	2425,5	0,99	24,76

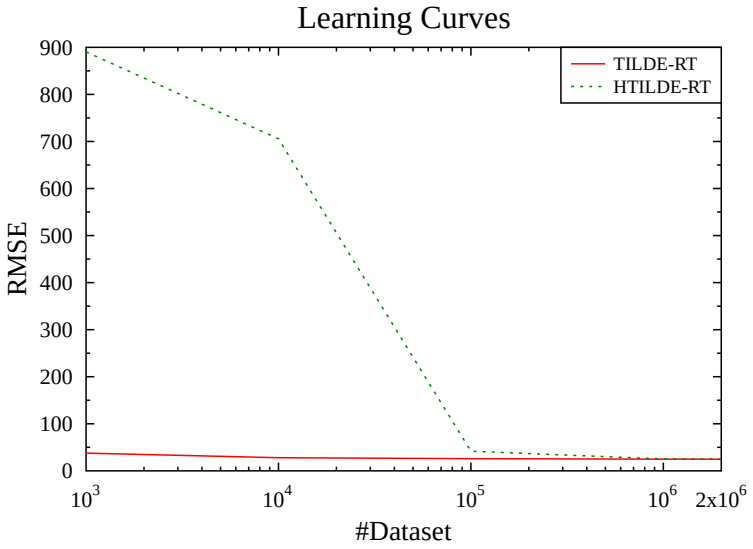


Figura 5.7: MV - Curvas de aprendizado

(resultados indicados por HTILDE-RT†). A nova parametrização conseguiu 23% de redução no tempo de aprendizado e 43% no tamanho das árvores, sem afetar as medidas de acurácia (Pearson e RMSE).

A intuição para o uso de tal configuração foi a mesma no *Artificial2*: com e_{min} maior o algoritmo faria menos cômputos da heurística, logo espera-se alguma redução no tempo de indução; e com δ menor, o algoritmo tornar-se-ia mais rigoroso, causando uma redução na quantidade de splits, levando a modelos menores. Entretanto, não foi gerada por validação interna deste dataset, e isto abre a possibilidade de que melhores resultados ainda podem ser obtidos.

A tabela 5.12 mostra o aprendizado neste dataset e a Figura 5.9 expõe a curva de aprendizado relativa ao RMSE.

Formação dos atributos:

$P(x_1 = 1) = P(x_1 = -1) = \frac{1}{2}$;

$P(x_m = 1) = P(x_m = 0) = P(x_m = -1) = \frac{1}{3}, m \in \{2, ..., 10\}$;

A variável dependente, y, é então formada através da equação:

$$y = \begin{cases} 3 + 3x_2 + 2x_3 + x_4 + \sigma(0,1), & \text{se } x_1 = 1 \\ -3 + 3x_5 + 2x_6 + x_7 + \sigma(0,1), & \text{se } x_1 = -1 \end{cases}$$

Figura 5.8: Gerador do dataset *2D-Planes*

	Tempo (s)	**Nós**	**Literais**	**Pearson**	**RMSE**
TILDE-RT	609,0	144,5	289,0	0,97	1,00
HTILDE-RT	259,1	201,5	402,0	0,97	1,00
HTILDE-RT†	199,6	113,0	226,0	0,97	1,00

Tabela 5.11: Resultados para *2D-Planes*

Tabela 5.12: Curva de Aprendizado para 2D-Planes

#Exemplos	**Tempo (s)**	**Nós**	**Literais**	**Pearson**	**RMSE**
HTILDE					
10^3	0,0	0,0	0,0	0,00	16,89
10^4	0,2	1,0	2,0	0,69	10,42
10^5	2,9	11,0	22,0	0,94	2,32
10^6	125,2	104,5	209,0	0,97	1,00
$2 * 10^6$	259,1	201,5	402,0	0,97	1,00
TILDE					
10^3	1,0	67,5	135,0	0,95	1,59
10^4	3,1	134,5	269,0	0,97	1,14
10^5	13,5	143,5	287,0	0,97	1,01
10^6	248,6	138,0	276,0	0,97	1,00
$2 * 10^6$	609,0	144,5	289,0	0,97	1,00

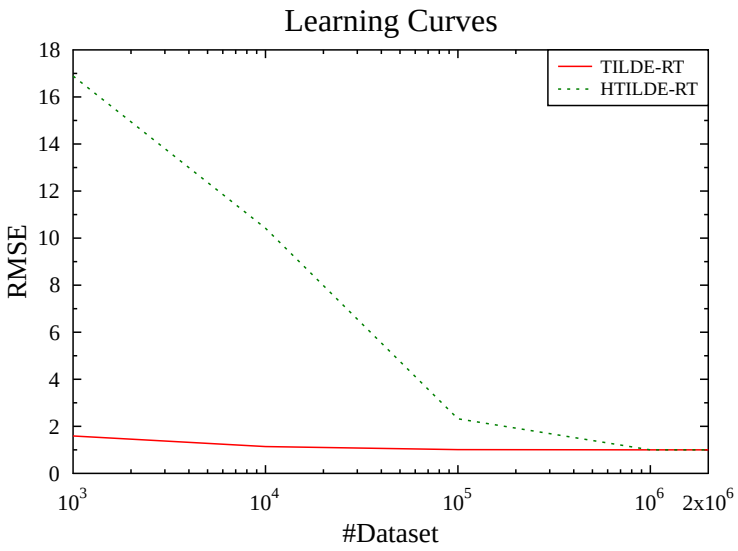


Figura 5.9: 2D-Planes - Curvas de aprendizado

Capítulo 6

Conclusão

Neste trabalho, foi desenvolvido e implementado o HTILDE-RT que é um algoritmo de programação em lógica indutiva escalável para grandes bases de dados relacionais e tratamento de fluxos de dados (streams). Este é uma extensão do HTILDE para tratar problemas de regressão que utiliza arcabouço da ILP aliado a eficiência da amostragem empregando o limitante de Hoeffding em problemas relacionais de regressão representados através do formato de interpretações.

Adicionalmente, uma alteração da heurística foi utilizada para contemplar as condições necessárias para a validade do limitante de Hoeffding, bem como enunciamos um corolário que estende a validade do comportamento assintótico das árvores de Hoeffding para o aprendizado por interpretações.

Experimentos em cinco grandes bases de dados artificiais foram realizados, onde duas bases são relacionais e as demais proposicionais, porém com propriedades interessantes para análise do comportamento do algoritmo. Os resultados sugerem uma boa troca entre eficiência e acurácia, pois o HTILDE-RT conseguiu aprendizado de 2.4 até 260 vezes mais rápido, com uma perda máxima de 2% na medida RMSE e sem perdas no coeficiente Pearson. Além disso, na grande maioria dos datasets o HTILDE-RT foi capaz de aprender modelos menores e, mesmo quando modelos maiores foram gerados, uma alteração intuitiva dos parâmetros conseguiu reduzir os tamanhos dos modelos sem causar grande perda nas demais medidas.

Com os resultados dos datasets sintéticos, *Artificial1* e *Artificial2*, este trabalho deu origem a um artigo publicado na XXXI Conferência da Sociedade Brasileira de Computação [43]. Em sequência, dispondo dos todos os resultados com os demais datasets, confeccionamos um novo artigo que aguarda resultado de submissão a uma conferência internacional [44].

6.1 Trabalhos Futuros

Nesta seção enumeramos uma lista de trabalhos futuros que consideramos desejáveis. Segue:

1. Aprimorar o tratamento de atributos contínuos para streams, através da aplicação das árvores binárias de busca estendidas conforme [18];
2. Embora aproveite muitas das funcionalidades do sistema ACE, tal sistema é genérico e comporta vários algoritmos de aprendizado ILP, não sendo adequadamente preparado para o tratamento de streams. Deste modo, o HTILDE-RT sofre limitações em função de certos gargalos que o sistema apresenta. Pretendemos, então, elaborar uma implementação do HTILDE-RT de forma independente do sistema ACE;
3. Obter bases de dados relacionais reais, com potenciais problemas de regressão, a fim de realizar experimentos com o HTILDE-RT em tais bases;
4. O HTILDE-RT possui aplicação potencial para a área de aprendizado relacional estatístico (SRL - Statistical Relational Learning), como por exemplo no aprendizado das funções potenciais quando campos aleatórios condicionais são usados em problemas relacionais (TILDE-CRF [39]), ou mesmo no aprendizado das distribuições condicionais nas redes de dependência relacionais (RDN - Relational Dependency Networks [38]);
5. Comparar diretamente outros sistemas propostos para streams com o HTILDE-RT, como por exemplo o VFDT e o FIMT, em bases de dados relacionais;
6. Adaptar o HTILDE-RT para o aprendizado de model trees [18] [17] [40];
7. Prover suporte a fluxos com mudança da distribuição (concept drifts) [45].

Referências Bibliográficas

[1] MITCHELL, T. *Machine Learning*. New York, McGraw-Hill, 1997.

[2] BLOCKEEL, H., RAEDT, L. D., JACOBS, N., et al. "Scaling Up Inductive Logic Programming by Learning from Interpretations", *Data Mining and Knowledge Discovery*, v. 3, n. 1, pp. 59–93, 1999.

[3] BLOCKEEL, H., RAEDT, L. D. "Top-Down Induction of First-Order Logical Decision Trees", *Artificial Intelligence*, v. 101, n. 1-2, pp. 285–297, 1998.

[4] DOMINGOS, P., HULTEN, G. "Mining high-speed data streams". In: *Knowledge Discovery and Data Mining*, pp. 71–80, 2000.

[5] AGGARWAL, C. "An Introduction to Data Streams". In: Aggarwal, C. C., Elmagarmid, A. K. (Eds.), *Data Streams: Models and Algorithms*, v. 31, *The Kluwer International Series on Advances in Database Systems*, Springer US, pp. 1–8, 2007.

[6] HOEFFDING, W. "Probability inequalities for sums of bounded random variables", v. 58, pp. 13–30, 1963.

[7] BLOCKEEL, H., DEHASPE, L., DEMOEN, B., et al. "Improving the efficiency of inductive logic programming through the use of query packs", *Journal of Artificial Intelligence Research*, v. 16, pp. 135–166, 2002.

[8] DEHASPE, L., RAMON, J., STRUYF, J., et al. *The ACE Data Mining System - User's Manual*, December 2005.

[9] LOPES, C., ZAVERUCHA, G. "HTILDE: scaling up relational decision trees for very large databases". In: *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA*, pp. 1475–1479, 2009.

[10] QUINLAN, J. R. "Induction of Decision Trees", *Machine Learning*, v. 1, n. 1, pp. 81–106, 1986.

[11] QUINLAN, J. R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[12] FRANK, A., ASUNCION, A. "UCI Machine Learning Repository". 2010. Disponível em: <http://archive.ics.uci.edu/ml>.

[13] BREIMAN, L., FRIEDMAN, J. H., OLSHEN, R. A., et al. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.

[14] MEHTA, M., AGRAWAL, R., RISSANEN, J. "SLIQ: A Fast Scalable Classifier for Data Mining". In: Apers, P. M. G., Bouzeghoub, M., Gardarin, G. (Eds.), *Advances in Database Technology - EDBT 96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings*, v. 1057, *Lecture Notes in Computer Science*, pp. 18–32. Springer, 1996. ISBN: 3-540-61057-X.

[15] GAMA, J. "Accurate decision trees for mining high-speed data streams". In: *In Proc. SIGKDD*, pp. 523–528. ACM Press, 2003.

[16] QUINLAN, J. R. "Learning With Continuous Classes". In: *5th Australian Joint Conference on Artificial Intelligence*, pp. 343–348, Singapore, 1992. World Scientific.

[17] POTTS, D., SAMMUT, C. "Incremental Learning of Linear Model Trees." *Machine Learning*, pp. 5–48, 2005.

[18] IKONOMOVSKA, E., , GAMA, J., et al. "Learning model trees from evolving data streams", *Data Mining and Knowledge Discovery*, v. 23, n. 1, pp. 128–168, 2011. ISSN: 1384-5810.

[19] Dzeroski, Lavrac (Eds.). *Relational Data Mining*. Berlin, SV, September 2001. ISBN: 3-540-42289-7.

[20] MUGGLETON, S., RAEDT, L. D. "Inductive Logic Programming: Theory and Methods", *Journal of Logic Programming*, v. 19/20, pp. 629–679, 1994.

[21] NIENHUYS-CHENG, S.-H., DE WOLF, R. *Foundations of Inductive Logic Programming*, v. 1228, *LNAI*. SV, February 1997. ISBN: 3-540-62927-0. Disponível em: <http://www.springer.de/cgi-bin/search_book.pl?isbn=3-540-62927-0>.

[22] RAEDT, L. *Logical and relational learning*. Cognitive Technologies. Springer, 2008. ISBN: 978-3-540-20040-6.

[23] RAEDT, L. D. "Logical settings for concept-learning", *Artificial Intelligence*, v. 95, n. 1, pp. 187–201, 1997. ISSN: 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(97)00041-6.

[24] KARALIC, A., BRATKO, I. "First Order Regression", *Machine Learning*, v. 26, n. 2-3, pp. 147–176, 1997.

[25] DIETTERICH, T. G. "Approximate Statistical Test For Comparing Supervised Classification Learning Algorithms", *Neural Computation*, v. 10, n. 7, pp. 1895–1923, 1998.

[26] NADEAU, C., BENGIO, Y. "Inference for the Generalization Error", *Machine Learning*, v. 52, n. 3, pp. 239–281, 2003.

[27] WITTEN, I. H., FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Sys. Morgan Kaufmann, June 2005. ISBN: 0120884070.

[28] BONGARD, M. M. *Pattern Recognition*. Spartan Books, 1970.

[29] DE RAEDT, L., DZEROSKI, S. "First order jk-clausal theories are PAC-learnable", *Artificial Intelligence*, v. 70, pp. 375–392, 1994.

[30] DE RAEDT, L. "Attribute-value learning versus inductive logic programming: The missing links". In: Page, D. (Ed.), *Inductive Logic Programming*, v. 1446, *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 1–8, 1998. ISBN: 978-3-540-64738-6. Disponível em: <`http://dx.doi.org/10.1007/BFb0027304`>. 10.1007/BFb0027304.

[31] BLOCKEEL, H. *Top-down Induction of First Order Logical Decision Trees*. Tese de Doutorado, Katholieke Universiteit Leuven, December 1998.

[32] JENSEN, D., NEVILLE, J. "Autocorrelation and linkage cause bias in evaluation of relational learners". In: *In Proceedings of the Twelfth International Conference on Inductive Logic Programming*, pp. 101–116. Springer, 2002.

[33] DOMINGOS, P., HULTEN, G. "A General Framework for Mining Massive Data Stream", *Journal of Computational and Graphical Statistics*, v. 12, pp. 945–949, 2003. Disponível em: <`citeseer.ist.psu.edu/hulten02mining.html`>.

[34] SRINIVASAN, A. "A study of two sampling methods for analysing large datasets with ILP", *Data Mining and Knowledge Discovery*, v. 3, n. 1, pp. 95–123, 1999.

[35] WIRTH, J., CATLETT, J. "Experiments on the Costs and Benefits of Windowing in ID3". In: *ML*, pp. 87–99, 1988.

[36] GROUP, K. U. L. M. L. "Cora data set". http://dtai.cs.kuleuven.be/ACE/data/cora.zip.

[37] MCCALLUM, A., NIGAM, K., RENNIE, J., et al. "Building domain-specific search engines with machine learning techniques". In: *Proc. AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace, 1999*, 1999.

[38] NEVILLE, J., JENSEN, D., CHICKERING, M. "Relational dependency networks", *Journal of Machine Learning Research*, v. 8, pp. 2007, 2007.

[39] GUTMANN, B., KERSTING, K. "TildeCRF: Conditional random fields for logical sequences". In: *In Proceedings of the 15th European Conference on Machine Learning (ECML-06*, pp. 174–185. Springer, 2006.

[40] VENS, C., RAMON, J., BLOCKEEL, H. "ReMauve: A Relational Model Tree Learner." In: Muggleton, S., Otero, R. P., Tamaddoni-Nezhad, A. (Eds.), *ILP*, v. 4455, *Lecture Notes in Computer Science*, pp. 424–438. Springer, 2006. ISBN: 978-3-540-73846-6.

[41] TORGO, L. "Regression Datasets". `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html`.

[42] FRIEDMAN, J. H. "Multivariate Adaptive Regression Splines", *Annals of Statistics*, v. 19, pp. 1–67, 1991. doi: 10.1214/aos/1176347963.

[43] MENEZES, G., ZAVERUCHA, G. "HTILDE-RT: Um algoritmo para aprender Árvores de Regressão Relacionais em grandes conjuntos de dados", *XXXI Congresso da Sociedade Brasileira de Computação - ENIA*, pp. 430–442, 2011.

[44] MENEZES, G., ZAVERUCHA, G. "HTILDE-RT: A first order logic regression tree learner for relational data streams", 2011. submitted.

[45] HULTEN, G., SPENCER, L., DOMINGOS, P. "Mining Time-Changing Data Streams". In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 97–106, San Francisco, CA, 2001. ACM Press.