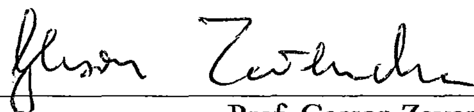
TORNANDO A PROGRAMAÇÃO EM LÓGICA INDUTIVA (ILP) ESCALÁVEL A BASES DE DADOS ARBITRARIAMENTE GRANDES

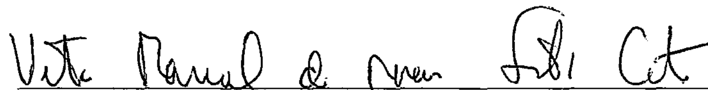
Pedro Motta Cardoso

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA 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.

Aprovada por:



Prof. Gerson Zaverucha, Ph.D.



Prof. Vítor Manuel de Morais Santos Costa, Ph.D.

Prof. Luís da Cunha Lamb, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 2006

CARDOSO, PEDRO MOTTA

Tornando a Programação em Lógica Indutiva (ILP) Escalável a Bases de Dados Arbitrariamente Grandes [Rio de Janeiro] 2006

XI, 70 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2006)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1 - Aprendizado de máquinas

2 - ILP

3 - Inferência estatística

4 - Bases de dados grandes

I. COPPE/UFRJ II. Título (série)

A todos aqueles que nunca deixaram de acreditar que este trabalho ficaria pronto.

Agradecimentos

Em primeiro lugar, agradeço a Deus, fonte de sabedoria e fé.

Agradeço também à minha mãe, meu pai e minha irmã por sempre estarem dispostos a me ajudar.

Agradeço à toda a minha família, por tudo que fizeram por mim.

Agradeço à minha namorada Priscila por todo carinho e ajuda.

Agradeço ao grande mestre (e doutor) Gerson Zaverucha, sem o qual esse trabalho nunca teria sido feito. Agradeço principalmente sua atenção, ajuda e, claro, paciência, muita paciência que ele teve que demonstrar ao longo deste trabalho.

Agradeço ao professor Vitor Costa por toda a ajuda e também pelo sistema Yap e pela base de dados CORA.

Agradeço também a todos os demais professores que ajudaram a formar a pessoa que sou hoje, principalmente (em ordem alfabética): Adriano Cruz, Celina Miraglia, Cláudio Esperança, Inês Dutra, Mário Benevides, Pedro Manoel e Rolci Cipolatti.

Agradeço ao professor Luís Lamb por aceitar o convite de participar da banca.

Agradeço a todos os meus amigos que me acompanham desde a graduação por toda a ajuda tangível e intangível, principalmente (em ordem alfabética): Alexandre Stauffer, o homem que simplesmente sabe as respostas antes de

se fazer as perguntas; Bruno Braga, meu parceiro de planos para dominar o mundo; Luís Rigo, além de artilheiro do Power Guido ainda me ajudou muito no labIA; Thiago Cordeiro, simplesmente meu parceiro (no sentido de programação XP) desde o 3º período da faculdade, ainda foi praticamente o responsável, junto com o Gerson, pelo tema deste trabalho; Vinicius Ramos que dentre outros incentivos, foi o goleiro no qual marquei meus únicos 2 gols pelo Power Guido.

Agradeço também ao meus amigos Aloísio Pina, Aline Paes, Aline Pina, Ana Luisa Dubboc e Carina Lopes por toda a ajuda.

Agradeço aos professores Ashwin Srinivasan, Filip Zelezny, Nada Lavrac e Pedro Domingos por tornarem seus sistemas (ALEPH, RSD, VFDT) públicos.

Agradeço a Richard Stallman e Linus Torvald por serem os pais do sistema operacional Linux.

Agradeço a CAPES pela ajuda financeira.

Agradeço ao virtual Mario Bros que sem dúvida alguma é un grande responsável por eu ter escolhido a carreira de computação.

Agradeço ao mestre Yoda por me ensinar a usar a força, ao Professor Xavier por me ensinar a controlar os meus poderes, ao tio Ben por me ensinar que grandes poderes requerem grandes responsabilidades, ao MacGyver por me ensinar que nada está perdido enquanto existir no mundo um canivete e fita adesiva e ao Bozo por me ensinar que "... Prá viver, é melhor sempre rir ...".

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

TORNANDO A PROGRAMAÇÃO EM LÓGICA INDUTIVA (ILP) ESCALÁVEL A BASES DE DADOS ARBITRARIAMENTE GRANDES

Pedro Motta Cardoso

Junho/2006

Orientador: Gerson Zaverucha

Programa: Engenharia de Sistemas e Computação

Atualmente, sistemas de informação estão em toda parte. Como resultado desse cenário, bases de dados muito grandes se tornaram bastante comuns. Ao trabalhar com tais bases, um grande problema surge: os algoritmos tradicionais de aprendizado não são capazes de trabalhar com tantos exemplos. Este trabalho propõe uma família de sistemas ILP, chamada VFILP (Very Fast ILP), que podem processar bases de dados relacionais muito grandes. Esta família é composta por 3 sistemas: VFILPh, VFILPpprog e VFILPprog. O dois primeiros sistemas utilizam proposicionalização para transformar os dados relacionais em uma representação proposicional. A fim de construir a teoria, VFILPh usa o sistema VFDT, que é um algoritmo de árvore de decisão preparado para processar milhares de exmplos. Os demais sistemas utilizam amostragem progressiva para melhorar a performance do aprendizado. Todos os sistemas foram testados em 2 bases de dados, uma sintética com 1250000 exemplos, e outra real com 622382 exemplos. Os resultados obtidos mostram que não apenas os sistemas obtiveram uma boa acurácia, como também geraram a teoria desejada para a base sintética.

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

SCALING UP INDUCTIVE LOGIC PROGRAMMING (ILP) FOR ARBITRARILY LARGE DATABASES

Pedro Motta Cardoso

June/2006

Advisor: Gerson Zaverucha

Department: Systems Engineering and Computer Science

Nowadays, information systems are everywhere. As a result of this, very large data sets have become very common. Working with these data sets, there is a major problem: the well-known learning algorithms are not able to deal with so many examples. This work proposes a family of ILP systems, named VFILP (Very Fast ILP), that can process very large relational data sets. This family is composed by 3 systems: VFILPh, VFILPpprog and VFILPprog. The first two systems use propositionalization to translate the relational data to a propositional representation. In order to construct a theory, VFILPh uses the VFDT system, which is a decision tree algorithm prepared to process very large data sets. The others systems use progressive sampling to scale up the learning task. All the systems were tested in 2 data sets, one synthetic with 1250000 examples, and one real with 622382 examples. The results show that not only the systems have obtained a good accuracy, but also they generated the desired theory for the synthetic data set.

Sumário

1 Introdução 1

2 Fundamentos Teóricos 6

2.1 Lógica de primeira ordem 6

2.2 Programação em Lógica Indutiva (ILP) 10

2.3 Proposicionalização 13

2.4 Medindo a performance de uma teoria 25

2.5 Inferência estatística 28

2.6 VFDT 32

3 Os sistemas VFILP 35

3.1 VFILPh 35

3.2 VFILPpprog 39

3.3 VFILPprog 41

4 Resultados Experimentais 43

4.1 East-West Train Problem 43

4.1.1 VFILPh 46

4.1.2 VFILPpprog 47

4.1.3 VFILPprog 48

4.2 Cora 49

4.2.1 VFILPh 51
4.2.2 VFILPpprog 51
4.2.3 VFILPprog 54
4.2.4 Comparação 56

5 Conclusão **58**

A ILP Modal **61**

Lista de Figuras

2.1 Exemplo de criação da tabela proposicional 14
2.2 Busca para a geração dos atributos 18
2.3 Divisão, real e pela teoria, dos exemplos 25
2.4 Curva de aprendizado (learning curve) 27
2.5 Curva de aprendizado com convergência rápida e lenta. 28
2.6 Tabela de contagem de exemplos em um nó da VFDT 33

3.1 Exemplo de árvore de decisão 37

4.1 Formulação original do problema. 44
4.2 Curva de treinamento para o "East-West train problem"no sistema VFILPh. 47
4.3 Convergência da acurácia para o "East-West train problem"no sistema VFILPpprog. 48
4.4 Curva de treinamento para o problema Cora no sistema VFILPh. 52
4.5 Evolução da precisão e da revogação para o problema Cora no sistema VFILPh. 52
4.6 Convergência da acurácia para o problema Cora no sistema VFILPpprog. 53
4.7 Evolução da precisão e da revocação para o problema Cora no sistema VFILPpprog. 54

4.8 Convergência da acurácia para o problema Cora no sistema VFILPprog. 55

4.9 Evolução da precisão e da revocação para o problema Cora no sistema VFILPprog. 56

Capítulo 1

Introdução

"Do or do not. There is no try!"
Mestre Yoda

Atualmente, os computadores estão em todas as partes. Até mesmo simples restaurantes universitários já contam com sistemas de informação para gerenciar seus principais processos. Uma tarefa muito comum feita por diversos sistemas é armazenar informações para uma analise posterior, gerando assim grandes repositórios de informação. Dessa forma, trabalhar com grandes bases de dados se tornou um problema de grande importância. Além disso, muitas organizações, como bancos e companhias telefônicas possuem bases de dados que crescem sem limites a taxa de milhares de registros por dia.

Bases de dados com um milhão de exemplos são consideradas bases muito grandes [18]. Apenas armazenar esse dados seria um desperdício de informação. Contudo devidos as limitações de tempo e espaço, algoritmos tradicionais não podem ser usados neste cenário. Muitas vezes, até mesmo algoritmos com complexidade O(n) não podem ser usados, já que o tempo de processamento seria proibitivo devido ao número, possivelmente infinito, de exemplos [19].

Uma técnica muito utilizada quando se trabalha com tais bases de dados é a técnica de inferência estatística. Tal metodologia consiste em analisar somente um subgrupo dos dados, diminuindo assim o tempo de processamento e a memória utilizada. Acredita-se que, em certas circunstâncias, um algoritmo preparado para utilizar somente um subconjunto dos dados é capaz de obter uma acurácia melhor do que um algoritmo simples que trabalhe com toda a base de dados [13].

A questão central ao se trabalhar com amostragem é determinar o tamanho do subconjunto dos dados que será analisada e como os dados serão selecionados para o subgrupo [41]. Essas escolha muitas vezes depende de fatores que não são conhecidos a priori. O ambiente de aprendizado teórico PAC [46, 16] utiliza uma técnica baseada na complexidade da amostra. Tal abordagem contudo, exige amostras muito grandes, o que não traria ganho de eficiência aos algoritmos de aprendizado.

Domingos e Hulten desenvolveram um método baseado em amostras para aumentar a performance de algoritmos de aprendizado de máquinas [12, 9] . Nesse método, a escolha do tamanho do subgrupo analisado é feita pelo limite de Hoeffding [17] . Dessa maneira, pode-se garantir que o modelo obtido analisando-se o subgrupo dos exemplos não é muito diferente do modelo que seria obtido a partir da análise de toda a base de dados.

Originalmente esse método foi aplicado a algoritmos proposicionais, como árvores de decisão (VFDT, CVFT) [9, 20], EM (VFEM) [11], K-Means (VFKM) [10] e Redes Bayesianas (VFBN) [14]. As vezes porém, representações proposicionais não são boas o bastante pois nelas não há uma maneira geral de se representar a relação entre os valores dos atributos [32]. Programação em lógica indutiva (ILP) [25] é conhecida por aprender a partir

de dados relacionais, porém está técnica costuma ser ineficiente para bases de dados grandes.

Em [44], um sistema ILP, CProgol + SS (*Sub Sampling*), que utiliza amostragem é apresentado. Porém, mesmo fazendo amostragem, o tempo de processamento cresce diretamente com o número de exemplos. Isso ocorre por que, depois que uma regra é gerada, no algoritmo de cobertura, todos os exemplos são analisados para descobrir os exemplos que não foram cobertos pela regra. Dessa forma este sistema não é capaz de trabalhar com base de dados muito grandes. Contudo, mesmo só tendo sido testado com bases de dezenas de milhares de exemplos, os resultados mostram que amostragem é uma abordagem promissora para ILP.

Uma técnica de ILP que vem sendo alvo de muitos estudos [25, 28, 26] é a transformação da representação relacional dos dados para uma representação proposicional. Tal técnica é chamada de proposicionalização e já se mostrou uma técnica bastante promissora para extração de conhecimento a partir de dados relacionais [23].

O objetivo deste trabalho é propor uma família de sistema ILP (VFILP, Very Fast Inductive Logic Programming) [4] capaz de trabalhar com bases de dados muito grandes (um milhão de exemplos ou mais). Esta família é composta por três sistemas, VFILPh, VFILPpprog e VFILPprog. Os dois primeiros são baseados na abordagem proposicional, ou seja, o problema relacional é transformado em um problema relacional, ao qual será aplicado um algoritmo de aprendizado para a extração de uma teoria. No final, essa teoria é novamente transformada para a lógica de primeira ordem. Já o sistema VFILPprog não efetua a proposicionalização.

Para garantir que os sistemas VFILPh e VFILPpprog sejam capazes de trabalhar com bases de dados muito grandes, o algoritmo de proposi-

cionalização não deve efetuar nenhuma seleção de atributos, o que seria muito custoso nesse cenário.

O sistema de proposicionalização utilizado é o sistema RSD, pois se a modelagem do problema for livre de constantes, a geração de atributos é feita a partir somente da linguagem dos *modes*. Essa restrição dos dados não poderem conter constantes não limita os problemas que podem ser abordados. Em [48], é apresentado um método de eliminação de constantes que não restringe o problema.

No sistema VFILPh, o algoritmo de aprendizado utilizado é o VFDT (*Very Fast Decision Tree*) [9]. Tal algoritmo foi desenvolvido para trabalhar com bases de dados muito grandes e apresenta uma performance muito superior aos algoritmos tradicionais.

Um sistema para aprendizado relacional, VFREL (*Very Fast RELational*) semelhante aos propostos nesse trabalho já foi descrito em [19]. Duas grandes diferenças, contudo existem entre o sistema VFREL e a família VFILP. A primeira é que no sistema VFREL a amostragem é feita no espaço dos atributos e o aprendizado é feito pelo algoritmo C4.5. Enquanto isso, nos sistemas VFILP a amostragem é feita no espaço dos exemplos pelo algoritmo VFDT ou pela amostragem progressiva. Dessa forma os sistemas VFILP são teoricamente capazes de processar um número maior de exemplos. A limitação no espaço dos atributos na família VFILP é feita pela limitação do tamanho máximo de cada atributo, ou de cada cláusula (VFILPprog). Essa limitação, contudo também existe no VFREL.

A segunda grande diferença é que o sistema VFREL transforma os dados relacionais em proposicionais utilizando uma abordagem de banco de dados, enquanto o VFILP utiliza uma abordagem lógica. Uma comparação entre essas duas abordagens pode ser encontrada em [23]. A escolha da abordagem

lógica foi feita pois o objetivo deste trabalho é criar um sistema ILP, que tem como objetivo criar uma teoria expressa em lógica de primeira ordem. E esse objetivo só poderia ser alcançado por esta abordagem.

Os sistemas VFILP foram testado em duas bases de dados. A primeira é uma base sintética com 1250000 para o problema "East-West train"proposto por Larson e Michalski em [24]. Esta base foi criada a partir de um gerador aleatório. A segunda base é uma real que tem como objetivo determinar quando duas citações diferentes referem-se ao mesmo trabalho científico. Este problema foi originado do sistema CORA proposto em [29, 30].

O trabalho está organizado na seguinte maneira. No próximo capítulo os fundamentos necessários para a compreensão dos sistemas VFILP serão apresentados. No capítulo 3 os sistemas VFILP serão apresentados com mais detalhes. No capítulo 4 os resultados experimentais são mostrados e por fim no capitulo 5 é feita a conclusão e propostas para trabalhos futuros são apresentadas.

Capítulo 2

Fundamentos Teóricos

"Dê-me uma alavanca e um ponto
de apoio, e eu moverei o mundo."
Arquimedes (matemático grego)

Neste capítulo serão apresentados os fundamentos necessários que servem de base para os sistemas VFILP. Na seção 2.1 será feita uma pequena introdução à lógica de primeira ordem. A seguir, na seção 2.2 os conceitos básicos de programação em lógica indutiva serão apresentados. Depois, na seção 2.3 serão explicados os conceitos básicos de proposicionalização assim como o algoritmo utilizado neste trabalho. A seção 2.4 comenta técnicas de mensuração da performance de uma teoria aprendida. Por fim, conceitos de inferência estatística serão apresentados na seção 2.5, e a seção 2.6 apresentará mais detalhes do algoritmo VFDT.

2.1 Lógica de primeira ordem

A lógica de primeira ordem, também conhecida como cálculo de predicados, é uma extensão da lógica proposicional com maior poder de expressão.

Nessa seção, será feita uma pequena introdução a está linguagem, maiores detalhes podem ser encontrados em [5].

Na lógica de primeira ordem, ao invés de se trabalhar com proposições, trabalha-se com predicados. Estes por sua vez, permitem que relações entre informações sejam representadas na própia linguagem. Por exemplo, sejam dadas duas proposições na lógica proposicional:

p: João é pai de José.

q: Bruno é pai de Marcos.

Embora essas duas proposições (p e q) compartilhem informações, elas são completamente independentes. Já na lógica de primeira ordem, essas duas informações podem ser representadas compartilhando um mesmo predicado:

Épai(João.José).

Épai(Bruno,Marcos).

Mais formalmente, pode-se definir um alfabeto para a lógica de primeira ordem como:

Definição 2.1 Alfabeto para a lógica de primeira ordem (padrão Edinburgh Prolog):

- Um conjunto de predicados, com aridade ≥ 1. Chama-se aridade de um predicado o número de termos que ele possui. No exemplo acima, *Épai(X,Y)* possui aridade 2.
- Um conjunto de constantes, normalmente representadas com a letra inicial minúscula.
- Um conjunto de simbolos funcionais com valência ≥ 1.
- Um conjunto finito de variáveis, normalmente representadas com a letra inicial maiúscula.

- Os operadores lógicos, também chamados de conectivos, $\wedge$ (e), $\vee$ (ou), $\rightarrow$ (se), $\leftrightarrow$ (se somente se), $\neg$ (não).
- Os quantificadores, existencial $\exists$ e universal $\forall$.
- Os operadores de escopo '(' e ')'.

Tendo-se definido o alfabeto da linguagem, pode-se definir um termo e uma fórmula bem formada (*well formed formulas, wwf*).

Definição 2.2 Um termo é definido como:

- Toda constante é um termo.
- Toda variável é um termo.
- Qualquer expressão $f(t_1,...,t_n)$ com $n \geq 1$, sendo que todo t_i é um termo, e f um simbolo funcional com aridade n, é um termo.
- Nada mais é termo.

Definição 2.3 Fórmula bem formada (*well formed formulas, wwf*):

- O predicado $\mathrm{P}(t_1,...,t_n)$ é uma fórmula bem formada, se sua aridade é n e todos os t_i são termos. Uma fórmula com apenas um predicado é chamada de fórmula atômica, átomo ou literal positivo.
- Se φ é uma fórmula bem formada, $\neg\ \varphi$ também é.
- Se φ e ψ são fórmulas, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \rightarrow \psi)$ e $(\varphi \leftrightarrow \psi)$ também são fórmulas.
- Se φ é uma fórmula com uma variável X, $\forall\ X\ \varphi$ e $\exists\ X\ \varphi$ também são.
- Nada mais é fórmula.

Quando se trabalha com aplicações computacionais utilizando-se a lógica de primeira ordem, comumente trabalha-se apenas com um subconjunto dela, chamado de cláusulas de Horn. Uma cláusula de Horn é uma disjunção de literais, com no máximo um literal positivo e todas as variáveis quantificas universalmente (eg. $\forall X_1 \ldots \forall X_n$ $(L_1 \vee \neg L_2 \vee \ldots \vee \neg L_m)$, onde $X_1,\ldots,X_n$ são todas as variáveis ocorrendo em $L_1, \neg L_2,\ldots,\neg L_m$).

As cláusulas de Horn são comumente escritas utilizando-se o operador de implicação (se, $\rightarrow$) e o operador de conjunção (e, $\wedge$), os quantificadores não são representados, e a quantificação universal fica implícita. A cláusula de Horn descrita acima normalmente é representada como:

$$L_1 \leftarrow L_2 \wedge \ldots \wedge L_m$$

O literal L_1 é chamado de cabeça da cláusula, e os predicados $L_2,\ldots,L_m$ formam o corpo da mesma. Em Prolog, o operador $\leftarrow$ é representado como :- e o operador $\wedge$ com ',':

$$L_1 \text{ :- } L_2,\ldots,L_m$$

Uma cláusula de Horn, por ter apenas um predicado na cabeça, é chamada de cláusula definida. Um conjunto de cláusulas definidas é chamado de programa lógico definido.

Um resultado importante a respeito da lógica de primeira ordem, é o Teorema de completude de Gödel [15]. Este teorema afirma que o cálculo de predicados é suficiente para provar qualquer fórmula logicamente válida. Outro resultado que tornou a lógica de primeira ordem ainda mais aceita foi a demostração de Robinson [43] que mostrou que apenas uma regra de inferência, chamada resolução, é correta e refutacionalmente completa para provar fórmulas nessa linguagem.

2.2 Programação em Lógica Indutiva (ILP)

Os resultados citados acima, serviram para provar que a dedução em lógica de primeira ordem poderia ser usada para resolver praticamente qualquer problema. Porém, uma questão ainda devia ser resolvida. Qual seria a origem do conhecimento usados na dedução? A resposta mais aceita a essa pergunta é que teorias lógicas que representam generalização de conhecimento, podem ser construídos a partir de fatos usando raciocínio indutivo [35].

A programação em lógica indutiva surgiu como a intersessão de aprendizado de máquinas com a programação em lógica. Seu objetivo é criar programas lógicos automaticamente usando raciocínio indutivo. Dessa forma, ela pode ser definida como um método de se construir uma teoria expressa em lógica de primeira ordem a partir de exemplos positivos e negativos e de um conhecimento preliminar [37]. Mais precisamente [44]:

Definição 2.4 Programação em lógica Indutiva (ILP):

- Seja B um conjunto finito de cláusulas, $B=\{C_1,C_2,...\}$. B será chamado de conhecimento preliminar (*background knowledge*).
- Seja $\mathcal{L}$ uma linguagem que especifica os predicados que podem ser usado para a construção da teoria. Essa linguagem também classifica os argumentos desses predicados, como de entrada, de saída ou constante. Um argumento de entrada deve ser uma variável já instanciada enquanto que um argumento de saída representa uma variável que será instanciada pelo predicado em questão.
- Seja E um conjunto finito de exemplos $E=E^+ \cup E^-$, onde E^+ são os exemplos positivos e E^- os exemplos negativos.

- H será a saída do algoritmo, ela será um conjunto finito de cláusulas de primeira ordem, H=$\{D_1$,D_2,...$\}$, que obedecem a linguagem $\pounds$.

Além disso, B, E e H devem obedecer:

- Necessidade primária: $B \not\models E^+$
- Suficiência fraca: Para cada D_i em H, $B \cup H \models e_1 \wedge e_2 \wedge ...$, onde $\{e_1, e_2, ...\} \subseteq E^+$
- Suficiência forte: $B \cup H \models E^+$
- Consistência fraca: $B \cup H \not\models \Box$
- Consistência forte: $B \cup H \cup E^- \not\models \Box$

A consistência forte e a suficiência forte, são normalmente relaxadas. Isso é feito porque os exemplos podem conter ruído, por isso é permitido que a teoria cubra alguns exemplos negativos e que deixe alguns exemplos positivos não cobertos.

A linguagem $\pounds$ é normalmente escrita na forma de declaração de *modes*. *Mode* é um predicado especial que possui 2 argumentos. O primeiro diz quantas vezes o predicado em questão pode aparecer em uma cláusula, e o segundo define o predicado. Nesta definição, os argumentos dele devem ser tipados e classificados como entrada +, saída − ou constante #. Abaixo há uma parte da declaração de *modes* do problema dos trens que será estudado adiante nesse trabalho.

mode(,has_car(+train,−car))*

mode(1,long(+car))

A primeira declaração define o predicado *has_car* com 2 argumentos, o primeiro do tipo *train* e de entrada, e o segundo do tipo *car* e de saída. Este predicado pode ser usado infinitas vezes em uma cláusula. A segunda

declaração define o predicado *long* com um argumento de entrada do tipo *car*. Este predicado pode ser usado apenas uma vez em casa cláusula da teoria.

Tendo-se definido o problema de programação em lógica indutiva, e a linguagem dos *modes*, pode-se propor um algoritmo para resolver o problema.

Algoritmo 1 Algoritmo guloso de programação em lógica indutiva (*greedy cover-set algorithm*)

1: ILP(B,$\pounds$,E):
2: Seja B um conhecimento preliminar
3: Seja $\pounds$ uma declaração de *modes* para os predicados que podem ser usados na teoria
4: Seja E um conjunto de exemplos
5: i=0
6: $E_i^+ = E^+, H_i = \emptyset$
7: **enquanto** $E_i^+ = \emptyset$ **faça**
8: i=i+1
9: $Treino_i = E_{i-1}^+ \cup E^-$
10: Usando $Treino_i$, seja D_i a "melhor" cláusula definida em $\pounds$ tal que $B \cup H_{i-1} \cup \{D_i\} \models \{e_i^+\}$ *onde* $\{e_i^+\} \subseteq E_{i-1}^+, B \cup H_{i-1} \cup \{D_i\} \not\models \Box$ *e* $B \cup H_{i-1} \cup \{D_i\} \cup E^- \not\models \Box$
11: $H_i = H_{i-1} \cup \{D_i\}$
12: $E_p = \{e_p | e_p \in E_{i-1}^+$ tal que $B \cup H_i \models \{e_p\}\}$
13: $E_i^+ = E_{i-1}^+ \setminus E_p$
14: **fim enquanto**
15: retorna H_i,

Esse algoritmo serve para mostrar por que o sistema CProgol+SS, mesmo fazendo amostragem, não é capaz de trabalhar com bases de dados muito grandes. Na linha 10 do algoritmo, o teste de cobertura é aplicado a todos os exemplos ainda não cobertos para garantir a consistência forte e a suficiência forte. Para evitar esse problema, o sistema VFILPh apenas se preocupa em fazer a melhor separação entre as classes, sem a preocupação de manter a consistência e a suficiência forte. Esse algoritmo será estudado melhor, na seção 2.6

2.3 Proposicionalização

A seção 2.1 introduziu a lógica de primeira ordem, e comentou suas vantagens sobre a lógica proposicional. Este capitulo irá tratar de como transformar problemas em lógica de primeira ordem para a lógica proposicional. O motivo dessa transformação será explicado adiante.

O processo de proposicionalização pode ser entendido como a transformação de um problema relacional para uma representação atributo-valor, que pode ser usada em um sistema de data-mining tradicional [22]. A parte principal do processo é a criação e a seleção dos atributos. Após esta parte, os dados podem ser representados em uma única tabela, sendo as colunas os atributos criados e as linhas os exemplos. Esses atributos normalmente tem a forma de um conjunto de literais de primeira ordem compartilhando variáveis. A figura 2.1 mostra graficamente um exemplo de criação da tabela proposicional.

O primeiro sistema ILP que utilizou uma tradução do problema em lógica de primeira ordem para a lógica proposicional e depois utilizou um algoritmo de aprendizado proposicional foi o sistema LINUS [25]. Desde então, muitos outros trabalhos seguiram o mesmo caminho. Mesmo sendo a linguagem proposicional, em princípio, menos poderosa, esta abordagem obteve bons resultados em muitos casos além de permitir um aumento da performance. Além disso, ao se traduzir o problema para a lógica proposicional, um número maior de algoritmos podem ser usados para o aprendizado. Por fim, na prática, em certos casos, para a construção de uma teoria, é suficiente procurar em um subespaço definido pelos atributos gerados [23].

A principal desvantagem ao se trabalhar com está abordagem é que a proposicionalização só deve ser efetuada em problemas centrados em indivíduos (*individual-centered problem*). Nesses problemas, há uma forte noção de indivíduo. Contudo, classificação e aprendizado de conceitos são proble-

```
eastbound(trem1).              eastbound(trem2).
has_car(trem1,carro1_1).       has_car(trem2,carro2_1).
long(carro1_1).                short(carro2_1).
open(carro1_1).                open(carro2_1).
...                            ...

atributo01(T):-has_car(T,C),long(C).
atributo02(T):-has_car(T,C),short(C).
atributo03(T):-has_car(T,C),long(C),open(C).
...
```

exemplo	atributo01	atributo02	atributo03	...
trem1	verdadeiro	falso	verdadeiro	...
trem2	falso	verdadeiro	falso	

Figura 2.1: Exemplo de criação da tabela proposicional

mas normalmente centrados em indivíduos. Já a síntese de programas não, pois quase sempre envolve algum cálculo para se determinar o valor de algum argumento [26].

A abordagem utilizada para a proposicionalização adotada neste trabalho foi a criação de atributos expressos como um conjunto de literais compartilhando uma variável. Para se entender o processo de criação de atributo, antes algumas definições devem ser feitas [26].

Definição 2.5 Variável global:

- Uma variável é dita global se somente se ela ocorre na cabeça da cláusula. Uma variável global é quantificada universalmente e o escopo do quantificador é a cláusula.

Definição 2.6 Variável local:

- Uma variável é dita local se ela ocorre somente no corpo da cláusula. Uma variável local é quantificada existencialmente e o escopo do quantificador é o corpo da cláusula.

Definição 2.7 Predicado estrutural:

- Um predicado é dito estrutural se ele introduz uma ou mais variáveis locais. Ele pode usar variáveis globais ou variáveis locais já introduzidas. Na linguagem de declaração dos *modes*, um predicado estrutural é um predicado que possui uma ou mais variáveis de saída.

Definição 2.8 Predicado utilitário:

- Um predicado utilitário é aquele que não introduz novas variáveis. Na linguagem de declaração dos *modes*, é um predicado com somente variáveis de entrada.

Definição 2.9 Atributo inválido:

- Um atributo é inválido se ele possui variáveis locais que foram usadas somente no predicado que a introduziu.

A partir dessas definições, o problema de construção de atributos pode ser visto com um problema de busca. Começando com as variáveis globais, e um atributo vazio, um predicado pode ser incluído no atributo se as variáveis de entrada do mesmo já foram introduzidas no atributo. Para cada predicado estrutural incluído, novas variáveis são introduzidas e novos predicados podem ser incluidos. No final, um atributo válido é criado se todas as variáveis introduzidas foram utilizadas em outros predicados além daqueles que a introduziram. Um algoritmo genérico para a construção de atributos é descrito a seguir 2.

Algoritmo 2 Algoritmo de criação de atributos

1: Seja $\pounds$ uma declaração de modes para os predicados que podem ser usados na teoria
2: Gera_atributos(profundidade,atributo_atual,variáveis_disponíveis):
3: **se** profundidade=0 **então**
4:     **se** não existe variável em variáveis_disponíveis que ocorra apenas no predicado que a introduziu **então**
5:         retorna atributo_atual
6:     **fim se**
7: **fim se**
8: Atributos=$\emptyset$
9: **para todo** predicado P_i definido em $\pounds$ **faça**
10:     seja V_i o conjunto das variáveis de entrada do predicado P_i
11:     **se** $V_i \subseteq$ variáveis_disponíveis **então**
12:         $Saída_i$ = variáveis de saída do predicado P_i
13:         Atributos = Atributos $\cup$ Gera_atributos(profundidade-1, atributo_atual+P_i, variáveis_disponíveis $\cup$ $Saída_i$)
14:     **fim se**
15: **fim para**
16: retorna Atributos

Abaixo há um exemplo de atributos criados, a partir da declaração dos modes para o problema dos trens.

modeh(1,eastbound(+train))

modeb(,has_car(+train,−car))*

modeb(,long(+car))*

modeb(,short(+car))*

atributo01(Train):-has_car(Train,Car).

atributo02(Train):-has_car(Train,Car), short(Car).

atributo03(Train):-has_car(Train,Car), long(Car), short(Car).

O primeiro atributo é invalido, pois a variável *Car* só é usada no predicado *has_car* que foi o predicado que a introduziu. Os demais atributos são válidos.

Se algum predicado estrutural pode aparecer infinitas vezes em uma cláusula, o número de atributos que podem ser gerados é infinito também, pois este predicado poderia ser incluído infinitas vezes introduzindo sempre novas variáveis. Para evitar tal problema, normalmente limita-se o tamanho máximo dos atributos, limitando o número de predicados que podem ocorrer nele.

Mesmo com o tamanho limitado, o número de atributos que podem ser gerados para um problema, assim como o custo computacional da criação dos mesmo, depende da complexidade do problema. Em alguns casos, tanto o número de atributos, quanto a complexidade podem ser polinomiais no tamanho máximo permitido para um atributo. Porém em certos problemas, ambos podem ser exponenciais [48]. Como, por motivos práticos que serão descritos adiante, o sistema VFILp trabalha com atributos de tamanho relativamente pequenos, mesmo que o problema seja exponencial, isto não será crítico para o sistema.

A figura 2.2 mostra graficamente a geração de atributos para o problema dos trens. O último nó do terceiro nível da árvore mostra o problema já descrito do número infinito de atributos, já que o predicado *has_car* pode aparecer infinitas vezes em um atributo e cada vez que ele aparece ele introduz novas variáveis do tipo *car* que poderão ser usada pelos predicados utilitários.

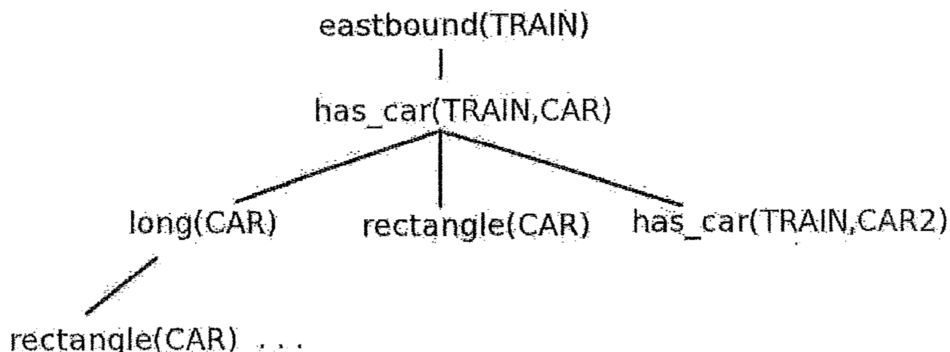


Figura 2.2: Busca para a geração dos atributos

Muitas técnicas existem para a escolha dos atributos mais significativos do problema. Uma dessas técnicas, utilizada pelo sistema RSD é baseada na teoria da relevância. Para se compreender essa técnica, algumas definições devem ser feitas [27].

Definição 2.10 Par p/n:

- Dados o conjunto de exemplos E=$E^+ \cup E^-$, onde E^+ são os exemplos positivos e E^- os exemplos negativos, um par p/n é um par de exemplos tal que $p \in E^+$ e $n \in E^-$

Definição 2.11 Cobertura de um atributo:

- Seja A um conjunto de atributos. Um atributo $a \in A$ cobre um par p_i/n_i se ele é verdade para p_i e falso para n_i.

Estes conceitos de par p/n e cobertura podem ser usados para provar um importante teorema sobre completude e consistência de descrição de conceitos.

Teorema 2.1 : Seja um conjunto de exemplos E, e um conjunto de atributos A, tal que uma teoria possa ser induzida de E e A. Seja $A' \subseteq A$. Uma teoria completa e consistente H pode ser descoberta usando-se apenas A' e E se para todo par p_i/n_i possível de exemplos existe pelo menos um atributo de A' que cobre este par.

Prova : Necessidade: Vamos supor que exista um par p_i/n_i que não é coberto por nenhum atributo de A'. Logo, nenhuma teoria pode ser criada usando-se apenas os atributos de A' que separe esses dois exemplos. Assim, é impossível gerar uma teoria que seja completa e consistente.

Suficiência: A partir de um exemplo positivo p_i, selecionar o subconjunto de atributos de A' que são verdade para p_i. Esse processo pode ser repetido várias vezes até que todos os exemplos positivos sejam cobertos. Em cada passo, uma regra H_i é criada sendo H_i=$A_{i,1} \wedge ... \wedge A_{i,n}$, onde os $A_{i,1}$,...,$A_{i,n}$ são os atributos que cobrem o exemplo em questão. No final uma teoria pode ser construída na forma H=$H_1 \vee ... \vee H_k$, sendo k o número de exemplos positivos. Está teoria é completa e consistente. □

Este teorema é muito importante para a teoria da relevância. Ele indica que a relevância de um atributo depende apenas dos pares p/n que ele cobre. A partir dessa observação, uma importante propriedade pode ser definida, a cobertura de atributos.

Definição 2.12 Cobertura de atributos:

- Seja $E(a)$ o conjunto de todos os pares p/n cobertos pelo atributo a. Um atributo a, cobre um atributo a' se $E(a') \subseteq E(a)$.

Além dessa, pode-se definir outra propriedade que será útil para detecção de atributos irrelevantes.

Definição 2.13 Custo de um atributo:

- Seja a um atributo, $c(a)$ é o custo deste atributo. Na teoria da relevância, a maneira como esse custo é calculada não tem importância. Este valor pode ser resultado de algum custo prático para o obtenção do valor do atributo (eg. exame laboratorial de alto custo monetário), ou da complexidade do atributo (eg. número de predicados) ou ainda ter o valor 1 para todos os atributos do problema, representando que todos têm igual custo.

Tendo essas definições, pode-se agora definir com precisão um atributo irrelevante.

Definição 2.14 Atributo irrelevante:

- Um atributo a' é irrelevante se existe um outro atributo a que cobre a' ($E(a') \subseteq E(a)$) e o custo de a é menor ou igual ao custo de a' ($c(a) \leq c(a')$).

Lema 2.1 : Sejam a e a' dois atributos. $P(a)$ representa o subconjunto dos exemplos positivos onde a tem valor verdadeiro. $N(a)$ representa o subconjunto dos exemplos negativos onde a tem valor falso. $E(a') \subseteq E(a)$ implica que $P(a') \subseteq P(a)$ e $N(a') \subseteq N(a)$. O reverso também é verdade, ou seja $P(a') \subseteq P(a)$ e $N(a') \subseteq N(a)$ implicam que $E(a') \subseteq E(a)$.

Prova : Por contradição. $E(a') \subseteq E(a)$. Porém $P(a')$ possui um exemplo positivo p_i que não pertence a $P(a)$. Escolhe-se agora um exemplo

negativo n_i para o qual a' e a tem valor falso. Contudo o par p_i/n_i é coberto por a', mas não é coberto por a. Logo a condição $E(a') \subseteq E(a)$ não pode ser atendida gerando assim uma contradição. □

Teorema 2.2 : Se um atributo a' é irrelevante, então para toda teoria H induzida de A e E, existe uma teoria H' induzia de A' e E (A'=$A \setminus \{a'\}$), com custo menor ou igual a teoria H ($c(H') \leq c(H)$).

Prova : Se a' é irrelevante, existe um atributo a tal que $E(a') \subseteq E(a)$ e $P(a') \subseteq P(a)$ e $N(a') \subseteq N(a)$. Sendo assim, em qualquer teoria completa e consistente H que utiliza a', a substituição de a' por a continuará mantendo a teoria completa e consistente. Como $c(a) \leq c(a')$, logo o custo da nova teoria será pelo menos igual ao custo da teoria inicial. □

De maneira análoga, pode-se definir uma teoria da relevância para exemplos. O objetivo é eliminar exemplos irrelevante, tal que a teoria completa e consistente induzida de A e E' ($E' \subseteq E$) também seja completa e consistente para o conjunto original de exemplos E.

Definição 2.15 Atributos de um exemplo, A():

- Seja um exemplo positivo p, então $A(p)$ é o subconjunto dos atributos de A que tem valor verdadeiro para p. Para os exemplos negativos n, A(n) é o subconjunto dos atributos de A que tem valor falso para n.

Definição 2.16 Cobertura de exemplos:

- Um exemplo e cobre outro exemplo e' se $A(e') \subseteq A(e)$ e se e e e' possuem a mesma classificação (positivos ou negativos).

Definição 2.17 Exemplos irrelevante:

- Um exemplo e é dito irrelevante se existe outro exemplo $e' \in E$ que cobre e.

Teorema 2.3 : Se um exemplo positivo $p \in E$ é irrelevante, então toda teoria completa e consistente H induzida de A e E' (E'=$E \setminus \{p\}$) também cobre p. Se um exemplo negativo $n \in E$ é irrelevante, então toda teoria completa e consistente H induzida de A e E' (E'=$E \setminus \{p\}$) também não cobre n.

Prova : Seja e' um exemplo irrelevante coberto pelo exemplo e. Se um atributo $a \in A$ tem valor verdadeiro para o exemplo e, ele também tem esse valor para e'. Então, uma representação conjuntiva/disjuntiva de H a partir dos atributos que são verdadeiros para e, também tem valor verdadeiro para e'. Isso prova que H cobre e'. □

Esses teoremas provam uma importante propriedade já descrita acima. Que é suficiente procurar por uma teoria H a partir de um subconjunto de exemplos e um subconjunto de atributos. Contudo, a seleção de atributos e exemplos feitas nos teoremas são baseadas em buscas exaustivas que por razões práticas são impossíveis de serem aplicadas em bases de dados muito grandes.

Dessa forma, ao se trabalhar com tais bases de dados, deve-se utilizar métodos estatísticos para a filtragem de atributos e exemplos [3]. Contudo, tais métodos geralmente introduzem uma incerteza nos valores obtidos. Em [19] uma escolha de atributos é feita a partir de resultados estatísticos garantindo porém que o modelo aprendido não é muito diferente do modelo que seria aprendido com todos os atributos, contudo não é aplicada nenhuma

eliminação de exemplos. Isso ocorre porque, na teoria, não é claro se o modelo aprendido será parecido com o modelo que seria aprendido, ao se aplicar métodos estatísticos de eliminação em duas etapas diferentes do algoritmo.

Por isso, nesse trabalho, será aplicado métodos estatísticos para a redução do número de exemplos. O número de atributos será limitado apenas pelo tamanho máximo permitido para um atributo.

Esta seção termina com uma breve análise do sistema RSD (*Relational Sub-group Discovery*) que é utilizado no sistema VFILP para a construção dos atributos.

RSD

O sistema RSD (*Relational Sub-group Discovery*) [28] foi originalmente desenvolvido para descobrir subgrupos nos dados de entrada. Ele possui três módulos básicos, mas o sistema VFILP utiliza apenas o primeiro que cria os atributos para o problema consultando apenas a declaração dos *modes*. Este modulo não é capaz de criar atributos com predicados que contenham constantes, este tipo de atributo só é construído pelo segundo módulo. Contudo, este segundo módulo não pode ser usado pelo sistema VFILP, pois ele efetua seleção de atributos, e como dito anteriormente esta funcionalidade não deve ser utilizada. Por isso o sistema VFILP tem a restrição de que os predicados não podem conter constantes. Contudo, como já dito anteriormente, essa restrição não limita os problemas que podem ser tratados pelo sistema, e em [48] é apresentado um método de se eliminar as constantes de um problema sem restringi-lo.

O segundo módulo termina o processo de proposicionalização selecionando os melhores atributos e criando a representação proposicional para os dados. Como ele não pode ser utilizado, um módulo PROLOG foi uti-

lizado para a criação da representação proposicional. Este módulo verifica o valor de cada atributo para cada exemplo e cria uma tabela proposicional para o problema.

O terceiro módulo do sistema RSD é o responsável pela descoberta dos subgrupos, tarefa não relacionada a este trabalho. Por isso este módulo também não foi utilizado.

A geração de atributos do sistema RSD é baseada em uma busca em profundidade. Uma característica interessante dessa busca é que ela não gera um atributo que possa ser decomposto em dois ou mais atributos. Como exemplo, o atributo:

atributo1(A):-hasCar(A,B),hasCar(A,C),short(B),open(C).

Não será gerado por este sistema, pois este atributo pode ser decomposto em dois diferentes atributos:

atributo1a(A):-hasCar(A,B),short(B).

atributo1b(A):-hasCar(A,B),open(B).

Essa restrição de não gerar um atributo que possa ser dividido acaba por, em alguns casos, limitar o numero de atributos. No caso do problema dos trens, o sistema RSD não geraria infinitos atributos, pois apenas um predicado *has_car* aparece em cada atributo, já que se mais de um aparecesse, o atributo poderia ser dividido. Porém, se existisse um predicado que tivesse duas variáveis de entrada do tipo *car* o problema dos infinitos atributos voltaria a existir, já que agora, o atributo não poderia mais ser dividido.

Além dessa verificação, o sistema RSD também não permite a criação de atributos com os predicados em ordem diferente. Ou seja, apenas um dos seguintes atributos seria criado, já que a única diferença entre eles é a ordem dos predicados.

atributo1(A):-hasCar(A,B),open(B),short(B),rectangle(B).

atributo2(A):-hasCar(A,B).short(B),open(B),rectangle(B).

atributo3(A):-hasCar(A,B),short(B),rectangle(B),open(B).

2.4 Medindo a performance de uma teoria

Dado uma teoria H, a maneira mais usada para medir sua performance é a acurácia. Essa medida é a proporção de exemplos classificados corretamente pelo tamanho do conjunto de exemplos. Contudo, quando o problema tem uma classe com um número significativamente maior de exemplos do que as demais classes, a acurácia se torna insuficiente.

Isso ocorre, pois se uma classe é majoritária, mesmo que a teoria seja tendenciosa para essa classe, a acurácia não será baixa. Como exemplo, se 90% dos exemplos pertencem a uma classe, classificar todos os exemplos como desta classe, gera uma acurácia de 90%, contudo, todos os exemplos das demais classes foram classificados errados. Isso é um sério problema quanto o objetivo é detectar elementos das classes minoritárias.

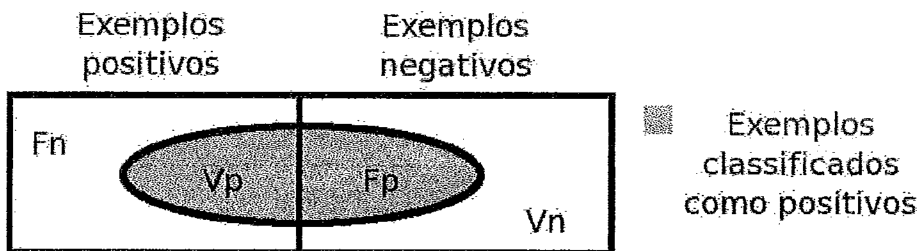


Figura 2.3: Divisão, real e pela teoria, dos exemplos

Nessa trabalho todos os problemas analisados se resumem a duas classes (exemplos positivos e negativos), e assim este caso será estudado com mais detalhes. A figura 2.3 mostra a separação do conjunto de exemplos em classes

gerada pela teoria e a separação real. Fp, Fn, Vp e Vn são abreviações para falsos positivos, falsos negativos, verdadeiros positivos e verdadeiros negativos respectivamente. A partir dessa divisão, pode-se introduzir duas novas medidas que serão usadas nesse trabalho, precisão (*precision*) e revocação (*recall*).

Definição 2.18 Revocação:

$$Revocacão = \frac{Vp}{Vp + Fn} \tag{2.1}$$

Definição 2.19 Precisão:

$$Precisão = \frac{Vp}{Vp + Fp} \tag{2.2}$$

Precisão e revocação são normalmente inversamente proporcionais. Isso ocorre pois se seu algoritmo de aprendizado tem como objetivo cobrir o maior número possível de exemplos positivos, ele acabará cobrindo também um número grande de exemplos negativos. Logo a revocação será alto e a precisão baixa. Por outro lado, se o algoritmo tem como objetivo minimizar o número de exemplos negativos cobertos certamente ele acabará deixando de cobrir exemplos positivos, logo a precisão será alto e a revocação baixo.

Uma outra maneira de se medir a performance de um algoritmo de aprendizado é através de uma curva de aprendizado (*learning curve*). A figura 2.4 mostra uma curva de aprendizado padrão. A curva é construída gerando várias teorias para subconjuntos dos exemplos. O tamanho de cada subconjunto é representado no eixo das abscissas, e a acurácia gerada com esses exemplos é representada no eixo das coordenadas.

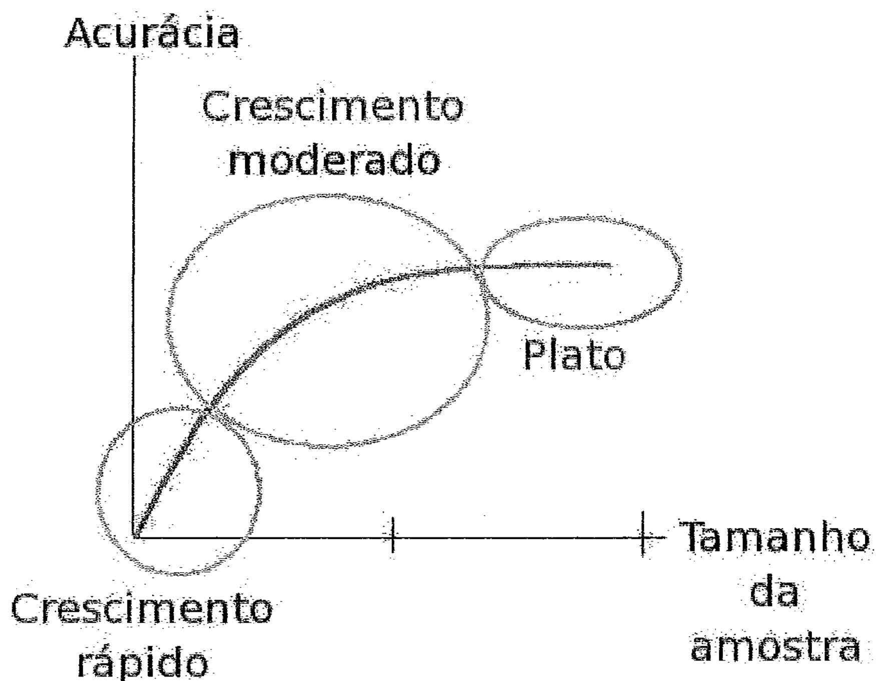


Figura 2.4: Curva de aprendizado (learning curve)

Curvas de aprendizado normalmente apresentam três regiões características que são marcadas na figura 2.4. Na primeira parte da curva, ela apresenta um crescimento acentuado. Na região intermediária, o crescimento da acurácia vai diminuindo, até estabilizar, na região denominada platô.

O tamanho de cada uma dessas regiões depende do problema. Em alguns casos, a parte intermediária é quase inexistente, com o crescimento da acurácia rápido até atingir o platô. Em outros casos, a curva é quase toda tomada pela região intermediária. A figura 2.5, mostra dois exemplos um com convergência rápida e outro com convergência lenta.

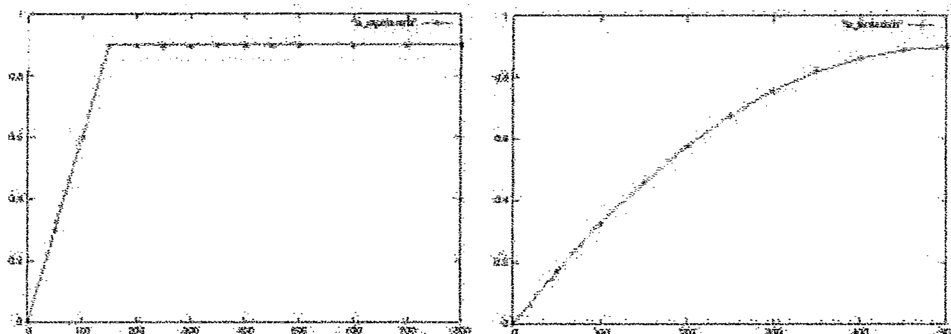


Figura 2.5: Curva de aprendizado com convergência rápida e lenta.

2.5 Inferência estatística

Para-se definir precisamente inferência estatística, primeiro uma definição deve ser feita [7].

Definição 2.20 População e amostra:

- Todos os elementos que deveriam ser analisados formam a população. Amostra é o nome dado a qualquer subconjunto da população.

A inferência estatística pode ser entendida como a área da matemática que estuda o problema de se afirmar certa propriedade sobre uma população a partir de informações contidas apenas em amostras. Ela é usada em muitos casos, como quando o custo da medição em toda a população é muito alto, ou quando a medição é destrutiva, sendo assim impossível de ser feita em toda a população (eg. tempo de vida de uma lâmpada).

Na computação, tais técnicas são bastante utilizadas pois permitem uma economia de memória e processamento ao analisar apenas um subconjunto do conjunto original dos exemplos. Além disso, a inferência estatística é praticamente obrigatória quando se trabalha com bases de dados muito grandes.

Uma questão central da inferência estatística é determinar como gerar uma amostra, e qual o tamanho da amostra. Para a primeira questão, duas abordagens são as mais utilizadas, a amostragem aleatória simples e a estratificada [6].

Na primeira, os exemplos são escolhidos aleatoriamente. Já na segunda, a classe de cada amostra é analisada, para que a distribuição de exemplos em cada classe seja igual tanto na população quanto na amostra. Esse tipo de abordagem é fundamental em problemas onde uma classe tem um número significativamente maior de representantes. Isso ocorre pois, ao se selecionar indivíduos de uma população com uma classe majoritária, a probabilidade de se escolher um indivíduo desta classe é maior, logo, existe a possibilidade dessa classe ser ainda mais majoritária na amostra, ou ainda, que nenhum individuo das classes minoritárias seja representado na amostra. Porém, em certos problemas, não se sabe a distribuição das classes a priori (eg. divisão dos habitantes de uma cidade em classes sociais). Contudo este tipo de problema não será abordado neste trabalho.

Todavia, determinar o tamanho da amostra muitas vezes requer conhecimento que não se tem a priori. Muitas técnicas existem para determinar o tamanho da amostra utilizando somente o conhecimento inicial do problema. Dentre várias técnicas, pode-se citar a amostragem progressiva [40], e os limites estatísticos.

A amostragem progressiva cria uma amostra aleatória pequena e mede a acurácia da teoria gerada a partir desses dados. Depois o algoritmo vai

aumentando o tamanho da amostra, até que a acurácia comece a convergir. Duas questões centrais dessa abordagem são, determinar a seqüencia de amostras e como determinar a convergência da acurácia.

Em [40] é provado que uma das melhores abordagens para a geração da seqüencia de amostras é a amostragem geométrica. Nessa técnica, as amostras seguem uma progressão geométrica, formando, para uma constante a, um conjunto da forma:

$$S = \{n_0, n_0 * a, n_0 * a^2, n_0 * a^3, ..., n_0 * a^k\} \tag{2.3}$$

Embora seja provado que a amostragem geométrica seja assintóticamente ótima, ainda não existe uma técnica definitiva para a detecção da convergência da acurácia. A técnica utilizada neste trabalho é similar a utilizada em [40]. Para cada amostra, outras amostras com tamanho próximo também são analisadas, e a convergência é detectada quando a reta gerada por regressão linear nesses pontos tem inclinação zero. Este critério é utilizado pois uma reta com inclinação zero deveria indicar a convergência na acurácia.

Já as abordagens baseadas em limites estatísticos determinam um tamanho para a amostra dada um limite para incerteza que a análise dessa amostra irá gerar. Os limites que consideram a distribuição da população como normal ($N(\mu, \sigma^2/n)$), utilizam os resultados dessa distribuição para determinar um tamanho de amostra a partir da incerteza (ϵ) e da confiança (δ) desejada.

$$P(|\bar{X} - \mu| \leq \epsilon) \geq \delta \tag{2.4}$$

$$P(\epsilon \leq \bar{X} - \mu \leq \epsilon) = P(\frac{-\sqrt{n}\delta}{\sigma} \leq Z \leq \frac{\sqrt{n}\delta}{\sigma}) \approx \delta \tag{2.5}$$

$$\frac{\sqrt{n}\delta}{\sigma} = z_y \tag{2.6}$$

$$n = \frac{\sigma^2 z_y^2}{\delta^2} \tag{2.7}$$

Nesse trabalho, contudo, essa suposição de distribuição normal não será feita. Por isso, o sistema VFILP usa o limite de Hoeffding [17]. A escolha deste limite se deve ao fato que o sistema VFDT, que serve de núcleo para o sistema VFILP, utiliza este limite, que será descrito com detalhes a baixo.

Definição 2.21 Limite de Hoeffding:

- Seja uma variável aleatória r com imagem R (eg. R=1 para uma distribuição de probabilidade e R=log(c) se r é o ganho de informação e c o número de classes). Depois de n observações independentes de r, $\overline{r}$ é a média das observações. o limite de Hoeffding diz que, com probabilidade $1 - \delta$, o valor verdadeiro da média de r está dentro do intervalo $(r - \epsilon \ , \ r + \epsilon)$, onde:

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{2.8}$$

O limite de Hoeffding tem a vantagem de poder ser usado para qualquer distribuição de r. Contudo, por ser geral, ele é conservativo, ou seja, para garantir a probabilidade $1-\delta$ e o intervalo $(r-\epsilon \ , \ r+\epsilon)$ ele necessita de mais exemplos do que limites específicos para uma distribuição de probabilidade. O sistema CProgol + SS utiliza um limite específico para a distribuição normal semelhante ao descrito acima [7].

2.6 VFDT

O algoritmo VFDT (*Very Fast Decision Tree*) [9] usa o limite de Hoeffding para criar um algoritmo de árvore de decisão escalável para grandes bases de dados. Através do limite de Hoeffding, o algoritmo pode garantir com certa certeza (entrada do algoritmo) que a árvore aprendida não é muito diferente da árvore que seria aprendida utilizando todos os exemplos de treinamento.

O algoritmo de construção da árvore é bastante semelhante a um algoritmo de árvore de decisão tradicional. Em cada nó, um atributo é escolhido para ser a consulta daquele nó. Depois da escolha do melhor atributo, os nós filhos são criados, um para cada valor possível do atributo escolhido, e a construção continua nos nós filhos.

Para se escolher o melhor atributo em cada nó, a heurística utilizada normalmente em problemas discretos é o ganho de informação. Por padrão a VFDT utiliza o ganho de informação, porém devido ao uso do limite de Hoeffding, apenas um subconjunto dos exemplos é usado no cálculo do ganho.

Para se calcular o ganho de informação gerado por cada atributo, o algoritmo armazena apenas uma matriz em cada nó. Essa matriz conta para cada valor possível de cada atributo o número de exemplos já vistos de cada classe com aquele valor. Dessa forma o algoritmo é econômico em memória. Para se entender melhor esse cálculo, a figura 2.6 representa um exemplo com duas classes, cinco atributos (A1, ... ,A5), todos eles podendo ter o valor 0 ou 1. Depois de n exemplos, existem $n1$ exemplos da classe 0 sendo X com $A1$ igual a 0 e Y com $A1$ igual a 1 ($n1 = X + Y$) e $n2$ exemplos da classe 1, analogamente $n2 = W + Z$. O conjunto de todos os exemplos já analisados pelo algoritmo será chamado de S. Assim, o ganho de informação do primeiro atributo será calculado como:

Classe 0

	A1	A2	A3	A4	A5
0	X	...	...	...	...
1	Y	...	...	...	...

Classe 1

	A1	A2	A3	A4	A5
0	W	...	...	...	...
1	Z	...	...	...	...

Figura 2.6: Tabela de contagem de exemplos em um nó da VFDT

$$
\begin{aligned}
G(S, A1) = \ & E(X+Y, W+Z) \\
& -\tfrac{X+W}{N} E(X, W) \\
& -\tfrac{Y+Z}{N} E(Y, Z)
\end{aligned}
$$

Onde G é o ganho de informação e E é entropia que pode ser calculada como:

$$
\begin{aligned}
E(P, N) = \ & -\tfrac{P}{P+N} log_2(\tfrac{P}{P+N}) \\
& -\tfrac{N}{P+N} log_2(\tfrac{N}{P+N})
\end{aligned}
$$

O limite de Hoeffding é usado na escolha do melhor atributo. Depois de N exemplos, e com uma probabilidade $1 - \delta$ (entrada do algoritmo), o limite de Hoeffding garante com essa probabilidade que o verdadeiro ganho de informação do atributo está dentro do intervalo $(G - \epsilon \ , G + \epsilon)$, sendo ϵ calculado pela formula 2.8.

Assim, se a diferença do melhor atributo para o segundo melhor for maior que ϵ, o limite de Hoeffding garante com certeza $1 - \delta$ que este melhor atributo também seria escolhido como o melhor se todos os exemplos fossem analisados.

Além de ser econômico armazenando apenas as matrizes de contagem, o algoritmo VFDT possui outras técnicas para aumentar a performance.

Detecção de empates. Se dois ou mais atributos têm um ganho de informação muito próximo, um grande número de exemplos seria necessário para decidir qual o melhor. Entretanto, isto seria um desperdício, pois a diferença gerada ao se escolher um deles seria pequena. Sendo assim, ao detectar um empate, o algoritmo escolhe um dos atributos empatados.

Numero de exemplos. O cálculo do ganho de informação é a parte mais dispendiosa do algoritmo, logo recalculá-la para cada novo exemplo visto não valeria a pena. Assim, esse calculo pode ser feito apenas após a análise de (n_{min}) exemplos, sendo esse número definido pelo usuário.

Após a análise dos fundamentos do sistema VFILP, a analise do sistema VFILP pode ser iniciada.

Capítulo 3

Os sistemas VFILP

"Para bom entendedor meia palavra ba."
"Para quem sabe ler, um ponto é letra."
Ditados populares brasileiros

Tendo-se apresentado todos os fundamentos para a compreensão dos sistemas VFILP, eles serão agora introduzidos. Ao todo a família de sistemas VFILP possui três diferentes implementações, tendo todas o mesmo objetivo de ser um sistema de aprendizado ILP preparado para bases de dados muito grandes. Neste capítulo, cada uma das três será analisada em sua respectiva seção.

3.1 VFILPh

O sistema VFILPh tem esse nome pois utiliza o limite de Hoeffding como método para a amostragem. Ele possui quatro módulos principais que serão descritos abaixo. Antes de se usar o sistema, as constantes do problema devem ser eliminadas, devido à técnica de proposicionalização descrita na seção 2.3.

Módulo 1: Criação dos atributos

Para realizar essa tarefa, o primeiro módulo do sistema RSD é utilizado. Ele cria as fórmulas para cada atributo a partir somente da declaração dos modes.

Módulo 2: Geração da tabela proposicional

A partir dos atributos gerados no módulo 1, e do conhecimento preliminar, um programa em Prolog gera a tabela proposicional para o problema, como mostra a figura 2.1. Esse módulo, junto com o primeiro realizam o processo de proposicionalização.

Módulo 3: Aprendizado

Com a tabela proposicional criada, ela pode ser dada como entrada para o algoritmo de aprendizado VFDT. Esse algoritmo descrito na seção 2.6 já aplica a amostragem, utilizando o limite de Hoeffding para garantir que a teoria aprendida não difere muito da teoria que seria aprendia se todos os exemplos fossem analisados. A saída do algoritmo é uma árvore que será processada no próximo módulo.

Módulo 4: Construção das regras

A partir da árvore de decisão gerada no módulo 3, e dos atributos gerados no módulo 1, as regras de primeira ordem que formam a teoria podem ser geradas. Esta é a etapa final do sistema VFILPh. Como exemplo, dada a árvore da figura 3.1, podemos extrair as seguintes regras:

positivo(X):-atributo01(X),atributo09(X).

positivo(X):-não atributo01(X), não atributo05(X).

Utilizando as fórmulas dos atributos, pode-se chegar as regras finais simplesmente substituindo cada atributo por sua respectiva fórmula. Como exemplo, dada as seguintes fórmulas:

atributo01(X):-has_car(X,Y),long(Y).

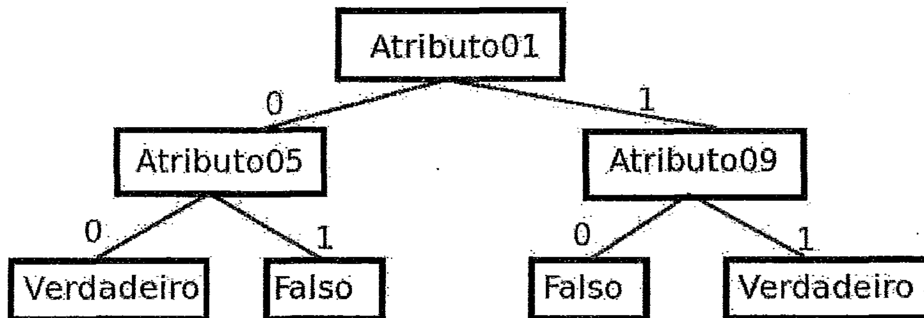


Figura 3.1: Exemplo de árvore de decisão

atributo05(X):-has_car(X, Y),load_triangle(Y).

atributo09(X):-has_car(X, Y),open(Y).

As seguintes regras seriam geradas:

positivo(X):-has_car(X, Y),long(Y),has_car(X,Z),open(Z).

positivo(X):-não (has_car(X, Y),long(Y)), não (has_car(X,Z), load_triangle(Z)).

O sistema VFILPh pode ser descrito mais formalmente pelo algoritmo 3.

Uma questão central nesse sistema é o tamanho máximo permitido para um atributo. Como dito na seção 2.3, esse é o único parâmetro que limita o número de atributos. Ao se escolher um número pequeno para esse parametro, a proposicionalização cria poucos atributos o que aumentará a performance do algoritmo de aprendizado. Contudo, a teoria que poderá ser aprendida nesse cenário também será limitada. Ao definir um tamanho muito grande, a teoria não fica tão limitada, mas o grande número de atributos gerados prejudicará a performance do aprendizado.

Algoritmo 3 Algoritmo VFILPh

1: VFILPh(B,$\pounds$,E,tamMax):
2: Seja B um conhecimento preliminar
3: Seja $\pounds$ uma declaração de modes para os predicados que podem ser usados na teoria
4: Seja E um conjunto de exemplos
5: Seja tamMax o tamanho máximo permitido para um atributo
6: A partir de $\pounds$ construir um conjunto de atributos A onde o tamanho máximo permitido para um atributo é tamMax
7: tamE=tamanho do conjunto E
8: tamA=tamanho do conjunto A
9: Seja M[tamE][tamA], a matriz que representa o problema na forma proposicional
10: **para todo** exemplo e $\in$ E **faça**
11:   **para todo** atributo a $\in$ A **faça**
12:     seja e_i o índice de e no conjunto E
13:     seja a_i o índice de a no conjunto A
14:     M[e_i][a_i]=valor do atributo a para o exemplo e.
15:   **fim para**
16: **fim para**
17: Executar o algoritmo VFDT passando M como entrada
18: Transforma árvore obtida em regras
19: Substituir toda ocorrência de um atributo nas regras por sua fórmula

Essa característica, contudo existe em vários algoritmos de aprendizado, como [19, 45, 23].

3.2 VFILPpprog

O sistema VFILPpprog utiliza amostragem progressiva sobre a representação proposicional do problema. Por isso receber esse nome (Proposicional com amostragem Progressiva). Ele também pode ser dividido em quatro módulos como o sistema VFILPh, sendo o terceiro módulo o único diferente. Esse módulo será descrito abaixo.

Módulo 3: Aprendizado VFILPpprog

Este módulo utiliza amostragem progressiva para determinar quando o aprendizado deve terminar. Um conjunto crescente de amostras é gerado, e para cada amostra uma teoria é construída e testada. Quando a acurácia converge para um valor, o aprendizado é interrompido.

Como algoritmo de aprendizado proposicional, o sistema VFILPpprog utiliza o algoritmo C4.5 [42]. Este sistema constrói uma árvore de decisão para os dados de entrada. A árvore gerada por este algoritmo será também transformada em regras pelo módulo 4.

A geração das amostras é feita de maneira geométrica, obedecendo a fórmula:

$$S = \{n_0, n_0 * a, n_0 * a^2, n_0 * a^3, ..., n_0 * a^k\} \tag{3.1}$$

Como esse sistema realiza a proposicionalização dos dados, ele introduz o parâmetro do tamanho máximo de um atributo, como discutido na seção anterior.

O sistema VFILPpprog pode ser descrito mais formalmente pelo algoritmo 4.

Algoritmo 4 Algoritmo VFILPpprog

1: VFILPpprog(B,$\mathcal{L}$,E,tamMax,tamInicial,a):
2: Seja B um conhecimento preliminar
3: Seja $\mathcal{L}$ uma declaração de modes para os predicados que podem ser usados na teoria
4: Seja E um conjunto de exemplos
5: Seja tamMax o tamanho máximo permitido para um atributo
6: Seja tamInicial o tamanho da primeira amostra
7: Seja a o fator de crescimento da amostra
8: A partir de $\mathcal{L}$ construir um conjunto de atributos A onde o tamanho máximo permitido para um atributo é tamMax
9: tamE=tamanho do conjunto E
10: tamA=tamanho do conjunto A
11: Seja M[tamE][tamA], a matriz que representa o problema na forma proposicional
12: **para todo** exemplo e $\in$ E **faça**
13:    **para todo** atributo a $\in$ A **faça**
14:       seja e_i o índice de e no conjunto E
15:       seja a_i o índice de a no conjunto A
16:       M[e_i][a_i]=valor do atributo a para o exemplo e.
17:    **fim para**
18: **fim para**
19: tamAmostra=tamInicial;
20: Seja A uma amostra de tamAmostra linhas de M
21: **enquanto** a acurácia não convergir e tamAmostra < tamE **faça**
22:    Executar o algoritmo C4.5 passando A como entrada
23:    tamAmostra=a *tamAmostra;
24:    Seja A uma amostra de tamAmostra linhas de M
25: **fim enquanto**
26: Transforma árvore obtida em regras
27: Substituir toda ocorrência de um atributo nas regras por sua fórmula

3.3 VFILPprog

O sistema VFILPprog utiliza amostragem progressiva da mesma maneira que o sistema VFILPpprog. Contudo, ele não realiza as etapas de proposicionalização. Logo, este sistema pode ser entendido como a aplicação da técnica de amostragem progressiva sobre um sistema de aprendizado ILP tradicional. O sistema ILP utilizado neste trabalho foi o sistema ALEPH [45].

Como o sistema VFILPprog não realiza proposicionalização, ele não introduz o parametro para o tamanho máximo de um atributo. Contudo, os sistemas de aprendizado ILP também limitam o tamanho máximo para uma cláusula para assim limitar o tamanho da busca pela melhor teoria. No sistema ALEPH, esse limite é definido pelo parametro *clauselength*. Desse modo, fica evidente que limitar o tamanho máximo de um atributo não é uma característica específica dos sistemas VFILPh e VFILPpprog.

Como no sistema VFILPpprog, a geração das amostras é feita de maneira geométrica, obedecendo a fórmula:

$$S = \{n_0, n_0 * a, n_0 * a^2, n_0 * a^3, ..., n_0 * a^k\} \tag{3.2}$$

A amostragem é feita apenas nos exemplos. Dessa forma a teoria preliminar, é a mesma em todas as execuções do sistema ALEPH. Isso diminui a performance do sistema, pois obriga a leitura e o armazenamento na memória de dados que não serão utilizados. Mas essa abordagem, torna desnecessário o pré-processamento dos dados, para permitir que a amostragem dos exemplos também divida o conhecimento preliminar.

O sistema VFILPprog pode ser descrito mais formalmente pelo algoritmo 5.

Algoritmo 5 Algoritmo VFILPprog

1: VFILPprog(B,$\mathcal{L}$,E,tamInicial,a):
2: Seja B um conhecimento preliminar
3: Seja $\mathcal{L}$ uma declaração de modes para os predicados que podem ser usados na teoria
4: Seja E um conjunto de exemplos
5: Seja tamInicial o tamanho da primeira amostra
6: Seja a o fator de crescimento da amostra
7: tamAmostra=tamInicial;
8: Seja A uma amostra de exemplos de E com tamanho tamAmostra
9: **enquanto** a acurácia não convergir e tamAmostra < tamE **faça**
10: Executar o algoritmo ALEPH passando como entrada B,$\mathcal{L}$,A
11: tamAmostra=a*tamAmostra;
12: Seja A uma amostra de exemplos de E com tamanho tamAmostra
13: **fim enquanto**

Tendo-se definido os sistemas da familia VFILP, pode-se estudar os resultados obtidos pelos mesmo em dois problemas. Este estudo será realizado no próximo capítulo.

Capítulo 4

Resultados Experimentais

"Depois disse a Tomé: "Introduz aqui o teu dedo,
e vê as minhas mãos. Põe a tua mão no meu lado.
Não seja incrédulo, mas homem de fé"."
João 20, 27

Os sistemas da família VFILP foram testados em dois problemas. O primeiro problema, assim como os resultados para ele serão objeto de estudo da próxima seção. Já o segundo problema será analisado na seção seguinte.

Em todos os testes com os sistemas que utilizam amostragem progressiva (VFILPpprog e VFILPprog), o tamanho inicial da amostra foi de 100 exemplos e a amostra foi duplicada ($a = 2$) a cada iteração do algoritmo.

4.1 East-West Train Problem

Este problema foi introduzido por Larson e Michalski em [24]. O objetivo do problema era criar a teoria mais concisa possível que separasse um conjunto de dez trens em duas classes, trens que estão indo para o leste e trens que estão indo para o oeste. Cada classe possuía cinco exemplos. Cada trem

tinha um ou mais vagões, e para cada vagão, várias propriedades existiam, como aberto ou fechado, longo ou curto... A figura 4.1 apresenta a conjunto original de trens.

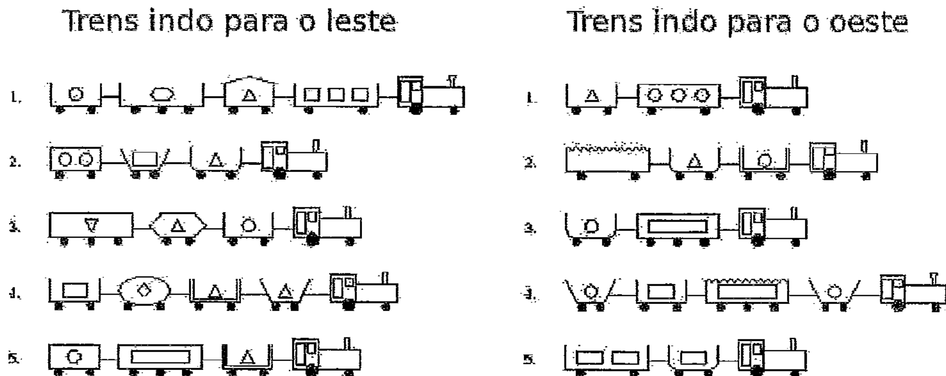


Figura 4.1: Formulação original do problema.

Em [31] uma nova formulação foi apresentada. Novos trens foram introduzidos, e um dos objetivos dessa nova formulação era criar uma teoria que obtivesse a melhor performance em um conjunto de teste.

Neste trabalho, utilizou-se um gerador de exemplos para esse problema a fim de se gerar 1250000 exemplos para o mesmo. Os exemplos foram gerados a partir da seguinte declaracão de modes:

modeh(1,eastbound(+train)).	modeb(*,has_car(+train,-car)).
modeb(1,bucket(+car)).	modeb(1,retangle(+car)).
modeb(1,u_shape(+car)).	modeb(1,hexagon(+car)).
modeb(1,elipse(+car)).	modeb(1,long(+car)).
modeb(1,short(+car)).	modeb(1,double(+car)).
modeb(1,none(+car)).	modeb(1,peaked(+car)).
modeb(1,flat(+car)).	modeb(1,arc(+car)).
modeb(1,jagged(+car)).	modeb(1,twowheels(+car)).
modeb(1,threewheels(+car)).	modeb(1,load_num0(+car)).
modeb(1,load_num1(+car)).	modeb(1,load_num2(+car)).
modeb(1,load_num3(+car)).	modeb(1,load_circle(+car)).
modeb(1,load_hexagon(+car)).	modeb(1,load_triangle(+car)).
modeb(1,load_retangle(+car)).	modeb(1,load_utriangle(+car)).
modeb(1,load_diamond(+car)).	

Todos os exemplos foram corretamente classificados pela seguinte teoria:

eastbound(T):-has_car(T,C),bucket(C),none(C),load_triangle(C).

eastbound(T):-has_car(T,C),long(C),double(C),threewheels(C).

eastbound(T):-has_car(T,C),u_shape(C),load_num3(C), load_diamond(C).

Do total de 1250000 exemplos metade foi gerada como positiva e a outra metade como negativa. Sendo assim a base de dados é balanceada e a acurácia é suficiente para medir a performance dos algoritmos. Os exemplos foram divididos em 5 grupos de 250000 cada um. Isso permite que os algoritmos sejam testados em uma validação cruzada de 5 grupos. Sendo assim cada algoritmo é executado 5 vezes, e em cada execução um dos grupos é utilizado

para teste e os outros para treinamento. A acurácia final do algoritmo é a média das acurácias obtidas em casa execução.

Nos sistemas VFILPp e VFILPpprog que utilizam amostragem progressiva, o conjunto de treinamento é novamente dividido, sendo que 750000 são usados para o treinamento, ou seja, o conjunto que será a população, e 250000 exemplos serão usados para validação. Esse conjunto de validação que será usado para detectar a convergência da acurácia.

4.1.1 VFILPh

O sistema RSD gerou 2625 atributos com tamanho máximo de quatro predicados. Esse tamanho máximo foi escolhido, pois 15275 atributos foram gerados com tamanho máximo igual a cinco, e com tantos atributos, só o processo de proposicionalização demorou mais de um dia de processamento. A geração dos 2625 atributos demorou menos de 1 s para ser feita. O modulo de proposicionalização demorou 70 min para proposicionalizr todos os exemplos.

Para o sistema VFILPh foi gerada a curva de treinamento mostrada na figura 4.2.

O sistema obteve uma acurácia de 100% e foi capaz de gerar a teoria original em todas as execuções da validação cruzada. O tempo de processamento foi de 115 min sendo 35 min para o aprendizado (VFDT) e 70 min para a proposicionalização. Contudo, pela figura 4.2, pode-se ver que não seria necessário o processamento de todos os exemplos para atingir essa acurácia. Porém o sistema VFILPh não tem mecanismos para a detecção deste cenário. Por fim, a detecção do número mínimo de exemplos necessário para gerar a acurácia final é uma caracteristica dos outros sistemas da família VFILP.

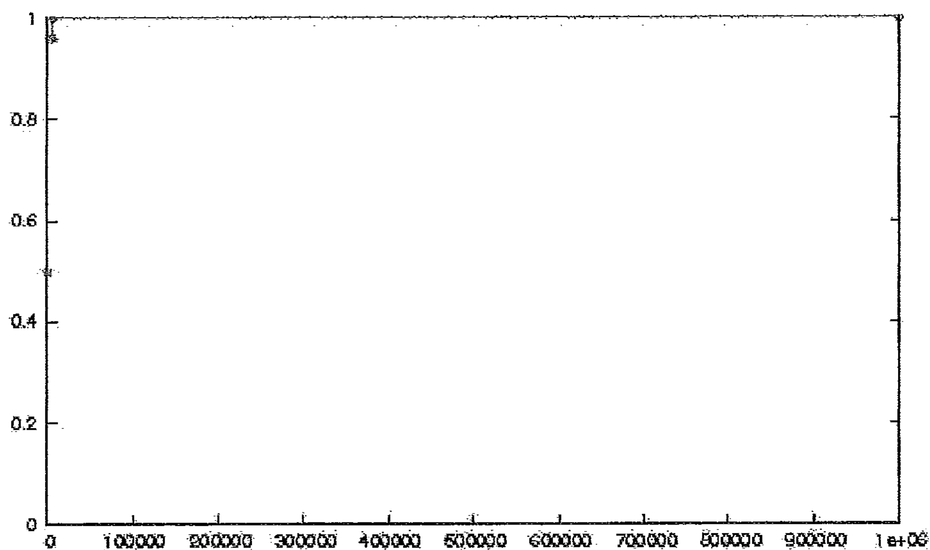


Figura 4.2: Curva de treinamento para o "East-West train problem"no sistema VFILPh.

4.1.2 VFILPpprog

Como dito anteriormente, o sistema VFILPpprog pega uma amostra com 100 exemplos, aplica o algoritmo C4.5, e dobra o tamanho da amostra até a acurácia convergir. A detecção de convergência é feita de maneira similar a realizada em [40]. Desse modo, tal sistema é capaz de parar o treinamento com um número pequeno de exemplos, se o problema assim permitir. Isso acontece no problema em questão.

A figura 4.3 mostra a convergência da acurácia, e com uma amostra de apenas 200 exemplos, a acurácia já havia convergido para 100%, e a teoria gerada era também a teoria desejada. O tempo total de processamento foi

71 min sendo 70 min para a proposicionalização e 1 min para o aprendizado (C4.5).

O tempo da proposicionalização foi alto, pois ela é feita em todos os exemplos antes de se iniciar o aprendizado. Essa abordagem foi escolhida, pois como a proposicionalização é um processo custoso, deseja-se evitar que um mesmo exemplos seja proposicionalizado mais de uma vez.

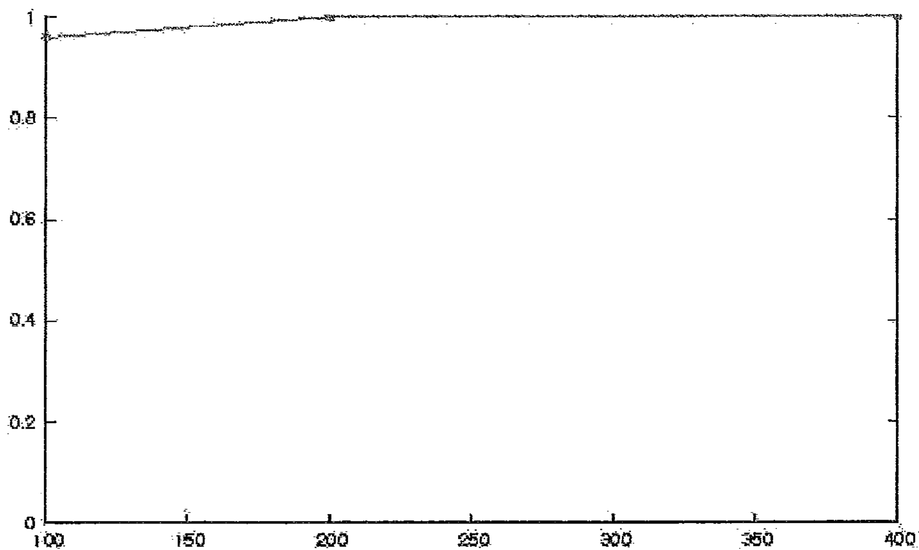


Figura 4.3: Convergência da acurácia para o "East-West train problem" no sistema VFILPpprog.

4.1.3 VFILPprog

Assim como o sistema VFILPpprog, o sistema VFILPprog realiza também amostragem progressiva. Contudo, nesse sistema, a primeira amostra com

apenas 100 exemplos já foi capaz de obter uma acurácia de 100%, gerando também a teoria desejada. Isso ocorreu em todas as cinco execuções da validação cruzada.

O tempo total de processamento para cada execução foi de 12 min. Sendo assim, esse sistema foi o melhor para o "East-East train problem", pois atingiu a mesma acurácia dos demais sistemas em menos tempo. Contudo deve-se ressaltar que os gráficos mostrados nas figuras 4.2 e 4.3, mostram que esse problema exige poucos exemplos para atingir a acurácia máxima.

4.2 Cora

A base de dados Cora foi proposta em [29, 30]. Ela contem informação sobre vários trabalhos científicos, incluindo suas referências. Um dos problemas propostos sobre essa base é determinar quando duas diferentes referências se referem ao mesmo trabalho [1]. Este será o problema analisado neste trabalho.

A base de dados usada é a mesma usada por Kok e Domingos em [21], contudo no referido trabalho, a métrica utilizada é a CLL e a AUC (área sobre a curva $precision - recall$), e como neste trabalho a métrica é a acurácia, a precisão e a revocação separados, os resultados não puderam ser comparados.

A base de dados contém 1295 referências para 112 trabalhos científicos na área de computação. No total, existem 25072 exemplos positivos e 597310 exemplos negativos. Esses exemplos foram agrupados em cinco grupos e cada grupo foi dividido em dois subgrupos. Sendo assim, os dados estão separados para a realização de uma validação cruzada 5x2 [8].

O problema possui a seguinte declaração de modes:

```
modeh(1,samePaper(+paper,+paper)).

modeb(*,paperTitle(+paper ,-title )).
modeb(*,paperYear(+paper ,-year )).
modeb(*,paperAuthor(+paper ,-author )).
modeb(*,paperVenue(+paper ,-venue )).

modeb(*,paperAuthor(-paper ,+author )).
modeb(*,paperVenue(-paper ,+venue )).
modeb(*,paperTitle(-paper ,+title )).

modeb(*,commonWordsInTitle100(+title ,+title )).
modeb(*,commonWordsInTitle80(+title ,+title )).
modeb(*,commonWordsInTitle60(+title ,+title )).
modeb(*,commonWordsInTitle40(+title ,+title )).
modeb(*,commonWordsInTitle20(+title ,+title )).
modeb(*,commonWordsInTitle0(+title ,+title )).

modeb(*,commonWordsInVenue100(+venue ,+venue )).
modeb(*,commonWordsInVenue80(+venue ,+venue )).
modeb(*,commonWordsInVenue60(+venue ,+venue )).
modeb(*,commonWordsInVenue40(+venue ,+venue)).
modeb(*,commonWordsInVenue20(+venue ,+venue )).
modeb(*,commonWordsInVenue0(+venue ,+venue )).

modeb(*,commonWordsInAuthor100(+author ,+author )).
modeb(*,commonWordsInAuthor80(+author ,+author )).
modeb(*,commonWordsInAuthor60(+author ,+author )).
modeb(*,commonWordsInAuthor40(+author ,+author )).
modeb(*,commonWordsInAuthor0(+author ,+author )).
```

A declaração foi dividida em 6 grupos. O primeiro define o predicado que deve aparecer na cabeça das regras aprendidas. O segundo grupo atribui um autor ou título ou editora ou ano para um determinado trabalho. O terceiro

grupo de predicados, atribuem um artigo dado um título ou autor ou editora. O quarto grupo mede a similaridade entre dois títulos. Esses predicados medem a porcentagem de palavras em comum entre os dois títulos. O quinto grupo mede a similaridade entre editoras e o último grupo a similaridade entre autores.

4.2.1 VFILPh

O sistema RSD gerou 599 atributos com tamanho máximo de quatro predicados. Novamente esse valor foi escolhido, pois com tamanho máximo igual a cinco, 7759 atributos foram construídos, e novamente o tempo de processamento da proposicionalização com tantos atributos passou de um dia. A geração dos 599 atributos demorou novamente menos de 1 s para ser feita. O modulo de proposicionalização demorou 40 minutos para proposicionalizar todos os exemplos.

Para o sistema VFILPh foi gerada a curva de treinamento mostrada na figura 4.4.

Analogamente a uma curva de treinamento, podemos traçar uma curva que mostra a evolução da precisão e da revocação. Essas curvas são mostradas na figura 4.5.

O sistema obteve, na média de todas as execuções da validação cruzada, uma acurácia final de 96%, com precisão 39% e revocação 28%. O tempo total de processamento para cada grupo da validação cruzada foi de 60 min sendo 40 min para a proposicionalização e 20 min para o aprendizado (VFDT).

4.2.2 VFILPpprog

Para realizar a amostragem progressiva, cada conjunto de treinamento foi dividido em duas partes de mesmo tamanho. Uma foi utilizada como

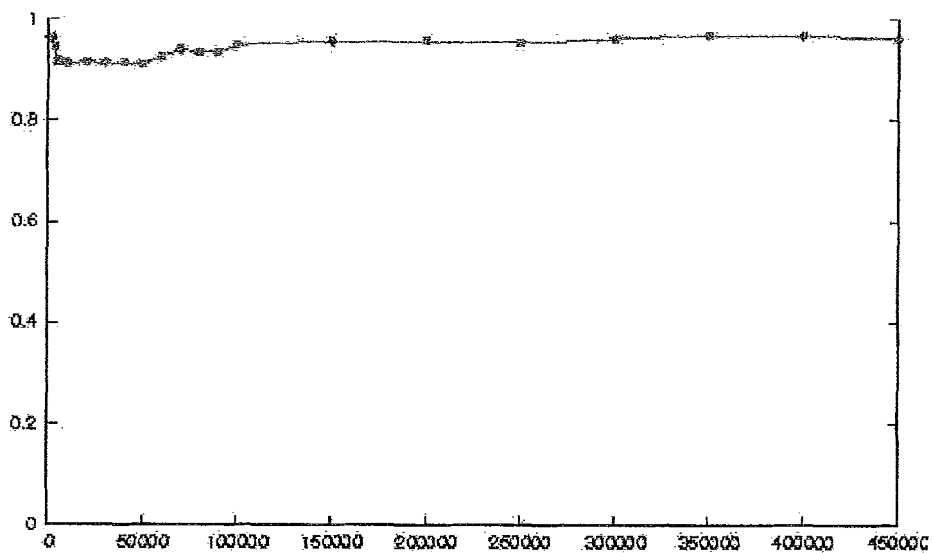


Figura 4.4: Curva de treinamento para o problema Cora no sistema VFILPh.

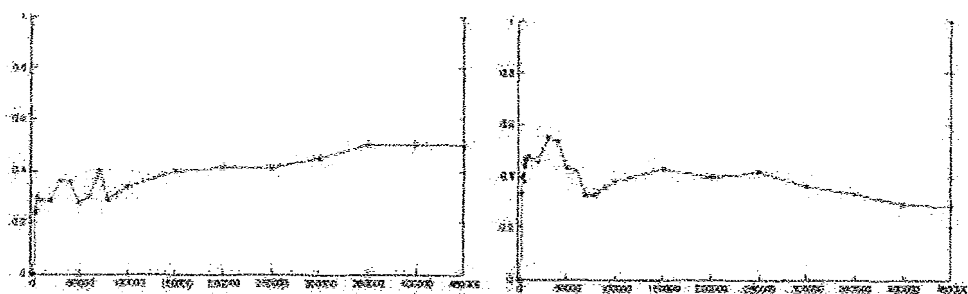


Figura 4.5: Evolução da precisão e da revogação para o problema Cora no sistema VFILPh.

população para as amostras, e a outra parte (validação) foi utilizada para detecção da convergência da acurácia. A acurácia final é calculada sobre o conjunto de teste. Essa divisão em três partes é necessária para que o classificador não seja tendencioso, já que o aprendizado é parado em função da acurácia obtida no conjunto de validação.

A figura 4.6 mostra a convergência da acurácia. Analogamente, a figura 4.7 mostra a evolução da precisão e da revocação.

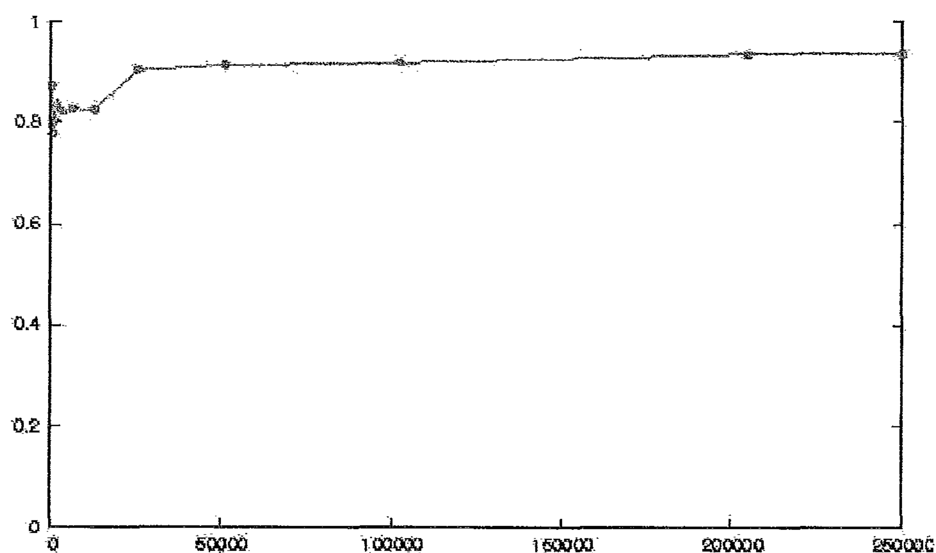


Figura 4.6: Convergência da acurácia para o problema Cora no sistema VFILPp-prog.

A convergência da acurácia foi detectada na amostra de tamanho 204800. Com uma amostra desse tamanho a acurácia foi de 94%, a precisão de 33% e a revocação de 38%.

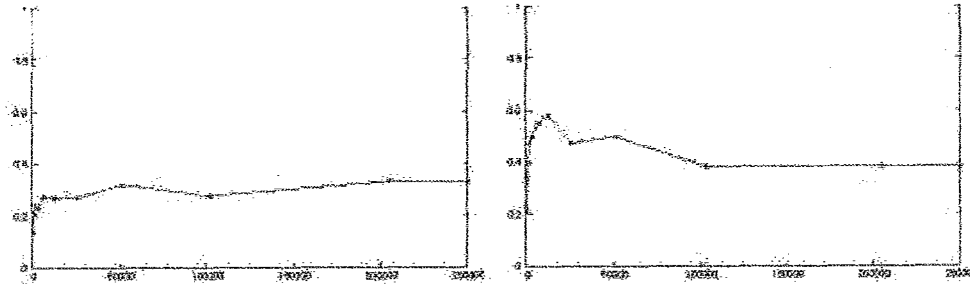


Figura 4.7: Evolução da precisão e da revocação para o problema Cora no sistema VFILPpprog.

O tempo de processamento total para cada grupo da validação cruzada foi de 62 minutos, sendo 40 min para a proposicionalização e 22 min para o aprendizado.

4.2.3 VFILPprog

Novamente, a convergência da acurácia e a evolução da precisão e da revocação são mostradas em gráficos, nas figuras 4.8 4.9 respectivamente.

Pelos gráficos, pode-se perceber, que o sistema ALEPH prioriza a Revocação sobre a acurácia. Outro ponto de atenção sobre o sistema VFILPprog, é que o processamento do sistema ALEPH foi limitado no tempo, pois se essa limitação não fosse feita, a partir de 25600 exemplos, o tempo de processamento do sistema ALEPH, passava de dois dias. O tempo dado para o sistema ALEPH processar é igual ao tempo de processamento total do sistema VFILPh, já que o objetivo deste trabalho é a comparação entre os sistemas. Essa limitação de tempo é baseada na idéia de que o sistema gera cláusulas mais gerais no início, e depois vai tentando cobrir os exemplos que ainda

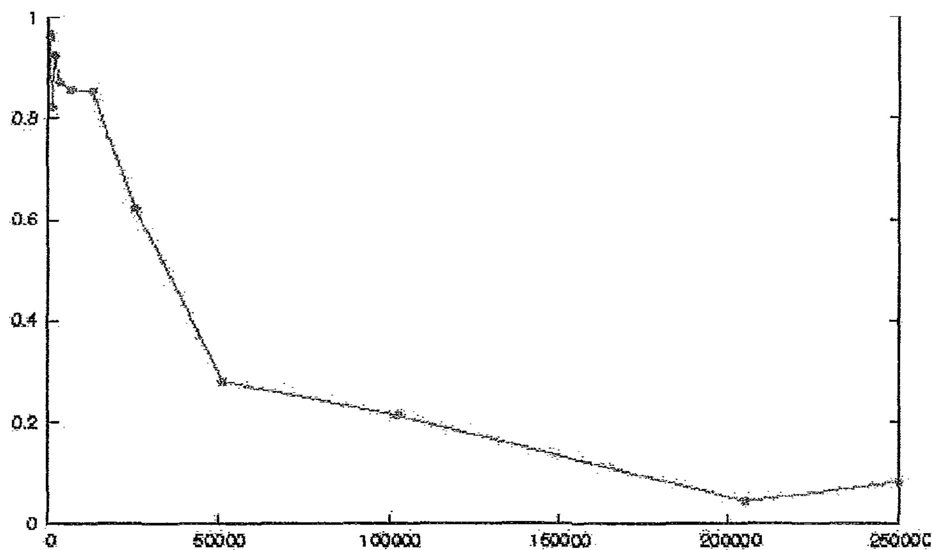


Figura 4.8: Convergência da acurácia para o problema Cora no sistema VFILPprog.

não foram cobertos. Dessa forma, ao se limitar o tempo de processamento, teremos como resposta só as primeiras cláusulas geradas que costumam ser mais gerais.

Não houve convergência da acurácia, logo, a acurácia final foi calculada utilizando-se toda a população para o treinamento. Nesse cenário, obteve-se 8% de acurácia, 4% de precisão e 99% de revocação.

O tempo de processamento foi de 4 horas e meia, isso por que cada execução do sistema ALEPH teve o tempo limitado como explicado acima. Isso mostra que esse sistema, embora tenha obtido uma excelente revocação,

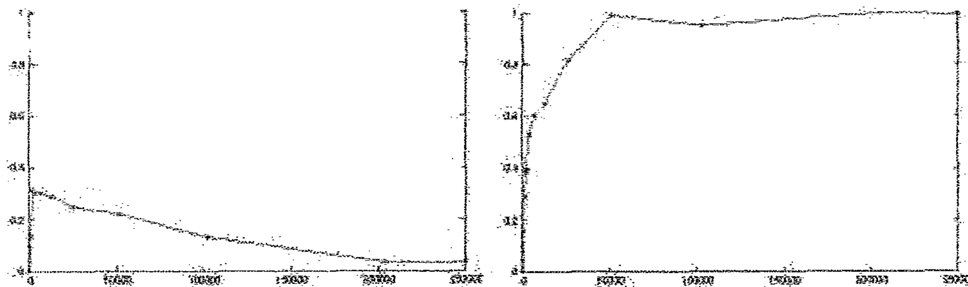


Figura 4.9: Evolução da precisão e da revocação para o problema Cora no sistema VFILPprog.

não tem uma performance promissora, já que mesmo com o tempo de processamento limitado, este tempo foi muito mais alto que os dos demais sistemas.

Usando a acurácia como objetivo, o sistema ALEPH não foi capaz de gerar uma teoria para os dados. Isso ocorreu pois como o número de exemplos negativos é muito maior que o de positivos, classificar os exemplos como negativos gera um acurácia de 96%, valor melhor do que qualquer teoria que ele tenha gerado no tempo limite.

4.2.4 Comparação

O sistema VFILPprog prioriza a revocação e por isso não pode ser comparado com os demais, já que sua acurácia é muito menor que a dos demais, e sua revocação maior. Além disso seu tempo de processamento é muito maior que os demais.

A diferença ente os sistemas VFILPpprog e VFILPh será calculada utilizando-se t-test com 95% de confiança para cada uma das medidas. Neste cenário temos a seguinte tabela, onde apenas a diferença da revocação se

mostrou estatisticamente significativa em favor do sistema VFILPpprog (* na tabela).

	VFILPh	VFILPpprog
Acurácia	96%	94%
Precisão	39%	33%
Revocação	28%	38% *

Capítulo 5

Conclusão

"No fim tudo dá certo, se ainda não deu certo
é porque ainda não chegou ao fim."
Fernando Sabino

Este trabalho apresentou uma família de sistemas de ILP capazes de trabalhar com bases de dados muito grandes. Os resultados mostraram que quando a teoria é simples e a base de dados não tem ruído, os sistemas que realizam amostragem progressiva levam vantagem sobre o sistema VFILPh. Esse é o caso do problema dos trens, onde os exemplos foram gerados a partir de um gerador que classificou todos os exemplos corretamente.

Contudo em uma base de dados com ruido, como o caso do problema Cora, o sistema VFILPprog apresentou uma performance muito inferior aos demais. Esse sistema é o único que não realiza proposicionalização. Deste modo, fica claro que sistemas proposicionais têm realmente uma performance melhor como sugerido em [23].

Porém, uma vantagem do sistema VFILPprog é que ele prioriza a revocação sobre a acurácia. Desse modo seus resultados podem ser mais interessantes para problemas com número de exemplos desbalanceados. Uma

idéia para trabalho futuro é colocar essa prioridade para a revocação (ou a precisão), nos demais sistemas da família VFILP.

Uma outra melhoria possível nos sistemas VFILPh e VFILPpprog seria determinar o tamanho máximo para um atributo dinamicamente a partir da declaração dos *modes*. Uma outra abordagem seria a criação de um filtro para atributos preparado para bases de dados muito grandes. Esse filtro seria semelhante ao sistema VFREL [19]. Contudo, como já dito nesse trabalho, ao aplicar inferência estatística em dois pontos diferentes do algoritmo, não fica claro que a teoria aprendida continua semelhante a teoria que seria aprendida com todos os exemplos. Mesmo assim, tal sistema poderia gerar bons resultados.

O sistema VFREL, mesmo fazendo amostragem nos atributos, também poderia ter sido usado nas comparações. Contudo, este sistema não é publico, e também não foi cedido para testes.

Em [2] é apresentado um sistema ILP escalável chamado TILDE. Tal sistema é uma extensão das árvores de decisão para a lógica de primeira ordem. Porém ele só foi testado em uma base de dados sintética com 100000 exemplos de treinamento. O uso das técnicas de amostragem descritas neste trabalho já estão sendo aplicadas em tal sistema para aumentar ainda mais sua performance.

Outro trabalho futuro, que também já está sendo realizado, é a implementação do limite de Hoeffding e de outros limites estatísticos no sistema ALEPH, assim como no sistema PROGOL [36].

Por fim, utilizando-se a idéia de tradução, um sistema ILP para lógica modal também já está sendo implementado. Neste sistema o problema representado em lógica modal é traduzido para a lógica de primeira ordem. Uma grande motivação dessa implementação é trabalhar com aprendizado

e inferência em problemas de múltiplos agentes inteligentes. O apêndice A contém uma análise inicial desse sistema.

Apêndice A

ILP Modal

"Por isso lhes falo por parábolas; porque eles, vendo, não vêem; e ouvindo, não ouvem nem entendem."

Mateus 13, 13

Neste apêndice, um sistema ILP para a lógica modal será apresentado. Esse sistema contudo, ao invés de trabalhar diretamente com a lógica modal, realiza a tradução do problema para a lógica de primeira ordem e depois realiza o aprendizado. Por fim as regras aprendidas são novamente traduzidas para a lógica modal. Essa abordagem é semelhante a abordagem dos sistemas VFILPh e VFILPpprog.

No capítulo 2 foi apresentado um método de se traduzir problemas expressos em lógica de primeira ordem para a lógica proposicional. Essa idéia de traduzir uma lógica em outra é bastante usada pois apresenta algumas vantagens.

A primeira é que uma lógica nova ou complexa pode ser definida a partir de uma lógica já bastante conhecida e mais simples. Além disso, a tradução pode levar a um ganho de performance como dito em [23]. Por fim, o uso da tradução permite que probelmas expressos em lógicas mais complexas possam

ser processados por sistemas computacionais para lógicas mais simples. Neste apêndice a tradução será feita da lógica modal para a lógica de primeira ordem.

Embora a lógica de primeira ordem, como dito na seção 2.1, seja suficiente para a maioria dos problemas, ela não é adequada para problemas de conhecimento que envolvam conceitos como crença, tempo, ação, necessidade, obrigação... A lógica modal é uma família de lógicas, mais poderosa que a lógica de primeira ordem, que pode ser usada para tais problemas.

Somente a lógica K, a mais simples da família modal, será usada neste apêndice. Nessa lógica os operadores existentes são: $\sim$ para a negação, $\rightarrow$ para o condicional se, e $\Box$ com o significado "É necessário que". Os operadores $\vee$ (ou), $\wedge$ (e) e $\leftrightarrow$ (se somente se), podem ser definido a partir dos anteriores. A escolha da lógica K se deve ao fato dela ser suficiente para o problema estudado.

A tradução da lógica modal para lógica de primeira ordem é um assunto bastante estudado [33, 47, 34, 38]. Contudo, três grandes problemas podem ocorrer nesta abordagem. O primeiro é a uma explosão exponencial no número de cláusulas geradas. O segundo é o algoritmo de tradução entrar em uma repetição infinita derivada de uma relação simétrica. O ultimo grande problema é que as cláusulas geradas podem não mais estar no universo das cláusulas de Horn [39].

Para se traduzir a lógica modal, três abordagens podem ser feitas, a sintática, a semântica e a funcional. A abordagem escolhida para ser usada foi a funcional, pois sobre certas condições, os três problemas descritos acima são evitados. Além disso, esta abordagem pode gerar uma melhor performance nos sistemas computacionais de primeira ordem [39].

A tradução funcional é baseada em uma semântica alternativa para a lógica de primeira ordem. A idéia principal é usar um conjunto de funções (γ) para representar cada relação binária. Se o operador $\Box$ é serial, a tradução funcional (TF) pode ser definida como [39]:

$$
\begin{aligned}
TF(t, p) &= p(t) \\
TF(t, \sim A) &= \sim TF(t, A) \\
TF(t, A \vee B) &= TF(t, A) \vee TF(t, B) \\
TF(t, A \wedge B) &= TF(t, A) \wedge TF(t, B) \\
TF(t, \Diamond A) &= \exists \gamma TF([t\ \gamma], A) \\
TF(t, \Box A) &= \forall \gamma TF([t\ \gamma], A) \\
TF(t, [X]A) &= \forall \gamma_X TF([t\ \gamma_X], A)
\end{aligned}
$$

A última linha da tradução é usada para o operador $\Box$ indexado, muito utilizado em problemas de múltiplos agentes. Nesse caso, a operação "agente X sabe p", é traduzida para $[X]p$. Essa representação é usada no problema estudado.

Uma vez que o conhecimento do problema esteja traduzido para a lógica de primeira ordem, o sistema ALEPH [45] foi utilizado para o aprendizado.

O sistema foi aplicado no problema dos sábios (*Wise Men Puzzle*) [39], cujo objetivo é obter um conjunto de regras para determinar se um sábio possui ou não uma marca na testa. O único conhecimento que ele possui é se os outros sábios possuem ou não marcas nas suas testas e que pelo menos um sábio possui uma marca.

O problema foi modelado para três sábios a partir de 2 predicados, $limpo(S)$ e $sujo(S)$, sendo um a negação do outro. Como exemplo de tradução pode-se

citar a fórmula $[a]limpo(b)$, que significa que o sábio a sabe que a testa do sábio b está limpa. Sua tradução fica $limpo([w\ \gamma_a]\ ,\ b)$.

Para o aprendizado 14 exemplos foram utilizados. Eles foram gerados a partir dos casos possíveis para este problema. Este conjunto foi dividido em 10 exemplos para aprendizado e 4 para teste. O algoritmo de ILP foi executado 11 vezes variando os exemplos do conjunto de testes e assim uma acurácia de 88,68% foi obtida.

Assim, essa abordagem mostrou-se bastante promissora, pois atingiu uma boa acurácia, além disso as fórmulas aprendidadas puderam ser traduzidas novamente para a lógica modal e seu significado foi bastante elucidativo. Como exemplo temos a fórmula [a]sujo(a)←[a]limpo(b),[a]limpo(c). O seu significado é que um sábio pode concluir que está com a testa suja se os outros 2 estiverem com a testa limpa, o que óbvio já que pelo menos 1 sábio tem a testa suja.

Referências Bibliográficas

[1] BILENKO, M., E MOONEY, R. J. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)* (2003), pp. 39–48.

[2] BLOCKEEL, H., RAEDT, L. D., JACOBS, N., E DEMOEN, B. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery 3*, 1 (1999), 59–93.

[3] CANTÚ-PAZ, E., NEWSAM, S., E KAMATH, C. Feature selection in scientific applications. In *KDD* (2004), pp. 788 – 793.

[4] CARDOSO, P. M., E ZAVERUCHA, G. Comparative evaluation of approaches to scale up ilp. *ILP 2006 - http://ilp06.doc.ic.ac.uk/* (2006).

[5] CASANOVA, M., GIORNO, F., E FURTADO, A. *Programação em Lógica e a Linguagem Prolog.* Edgard Blücher, 1987.

[6] COCHRAN, W. G. *Sampling techniques, Trird Edition.* John Wiley and Sons, 1977.

[7] DE O. BUSSAB, W., E MORETTIN, P. A. *Estatística Básica.* Saraiva, 2004.

[8] DIETTERICH, T. G. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation 10*, 7 (1998), 1895–1923.

[9] DOMINGOS, P., E HULTEN, G. Mining high-speed data streams. In *KDD* (2000), pp. 71–80.

[10] DOMINGOS, P., E HULTEN, G. A general method for scaling up machine learning algorithms and its application to clustering. In *Proc. 18th International Conference on Machine Learning (ICML)* (2001), pp. 106–113.

[11] DOMINGOS, P., E HULTEN, G. Learning from infinite data in finite time. In *Advances in Neural Information Processing Systems (NIPS) 14* (2002), pp. 673–680.

[12] DOMINGOS, P., E HULTEN, G. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics 12* (2003), 945–949(5).

[13] FRIEDMAN, J. H. Data mining and statistics: What's the connection? In *Proceedings of the 29th Symposium on the Interface Between Computer Science and Statistics* (1997).

[14] GROSSMAN, D., E DOMINGOS, P. Learning bayesian network classifiers by maximizing conditional likelihood. In *Proceedings of the Twenty-First International Conference on Machine Learning* (2004), pp. 361–368.

[15] GÖDEL, K. *Über die Vollständigkeit des Logikkalküls.* PhD thesis, University Of Vienna, Vienn, 1929.

[16] HAUSSLER, D. Quantifying inductive bias : Ai learning algorithms and valiant's learning framework. In *Artificial Intelligence, an International Journal* (1988), pp. 177–221.

[17] HOEFFDING, W. Probability inequalities for sums of bounded random variables. In *J. Amer. Statist. Assoc.* (1963), pp. 13–30.

[18] HUBER, P. *Massive data sets workshop: The morning after.* National Academy Press pg. 169-184, 1996.

[19] HULTEN, G., DOMINGOS, P., E ABE, Y. Mining massive relational databases. *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data* (2003), 53–60.

[20] HULTEN, G., SPENCER, L., E DOMINGOS, P. Mining time-changing data streams. In *Proc. 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2001), pp. 97–106.

[21] KOK, S., E DOMINGOS, P. Learning the structure of markov logic networks. In *Proceedings of the Twenty-Second International Conference on Machine Learning* (2005), ACM Press, pp. 441–448.

[22] KRAMER, S., LAVRAC, N., E FLACH, P. Propositionalization approaches to relational data mining. In *Relational Data Mining* (2001), pp. 262–291.

[23] KROGEL, M., RAWLES, S., ZELEZNY, F., FLACH, P., LAVRAC, N., E WROBEL, S. Comparative evaluation of approaches to propositionalization. In *Proceedings of the 13th International Conference on Inductive Logic Programming (ILP 2003), vol. 2835 of Lecture Notes in Artificial Intelligence* (October 2003), Springer-Verlag Heidelberg, pp. 194–217.

[24] LARSON, J., E MICHALSKI, R. Inductive inference of vl decision rules. *Invited paper for the Workshop in Pattern-Directed Inference Systems, Hawaii, and published in SIGART Newsletter, ACM, No. 63* (1977), 38–44.

[25] LAVRAC, N., E DZEROSKI, S. *Inductive Logic Programming: Techniques and Applications.* EH, 1994.

[26] LAVRAC, N., E FLACH, P. A. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic 2*, 4 (October 2001), 458–494.

[27] LAVRAC, N., GAMBERGER, D., E JOVANOSK, V. A study of relevance for learning in deductive databases. *Journal of logic Programming, vol 40 (2/3)* (1999), 215–249.

[28] LAVRAC, N., ZELEZNY, F., E FLACH, P. Rsd: Relational subgroup discovery through first-order feature construction. In *Proceedings of the 12th International Conference on Inductive Logic Programming (ILP 2002), vol. 2583 of Lecture Notes in Artificial Intelligence* (2002), Springer-Verlag, pp. 149–165.

[29] MCCALLUM, A., NIGAM, K., RENNIE, J., E SEYMORE, K. Building domain-specific search engines with machine learning techniques. In *AAAI Spring Symposium on Intelligent Agents in Cyberspace* (1999).

[30] MCCALLUM, A. K., NIGAM, K., RENNIE, J., E SEYMORE, K. Automating the construction of internet portals with machine learning. *Information Retrieval 3*, 2 (2000), 127–163.

[31] MICHIE, D., MUGGLETON, S., PAGE, D., E SRINIVASAN, A. To the international computing community: A new East-West challenge.

Relatório técnico. Oxford University Computing laboratory, Oxford,UK, ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP/trains.tar.Z, 1994.

[32] MITCHELL, T. *Machine Learning.* McGraw Hill, 1997.

[33] MOORE, R. C. Reasoning about knowledge and action. In *Proc. of the 5th IJCAI* (Cambridge, MA, 1977), pp. 223–227.

[34] MORREAU, M., E KRAUS, S. Syntactical treatments of propositional attitudes. *Artif. Intell. 106*, 1 (1998), 161–177.

[35] MUGGLETON, S. Inductive logic programming. *New Generation Computing, 8(4)* (1991), 295–318.

[36] MUGGLETON, S. Inverse entailment and Progol. *New Generation Computing 13* (1995), 245–286.

[37] MUGGLETON, S., E RAEDT, L. D. Inductive logic programming: Theory and methods. In *Jnl. Logic Programming, 19,20* (1994), pp. 629–679.

[38] NONNENGART, A. First-order modal logic theorem proving and functional simulation. *IJCAI* (1993), 80–87.

[39] OHLBACH, H. J., NONNENGART, A., DE RIJKE, M., E GABBAY, D. Encoding two-valued non-classical logics in classical logic. In *Handbook of Automated Deduction*, J. A. Robinson e A. Voronkov, Eds., vol. 1. MIT Press, 2001, pp. 1403–1486.

[40] PROVOST, F., JENSEN, D., E OATES, T. Efficient progressive sampling. In *KDD* (1999), pp. 23 – 32.

[41] PROVOST, F. J., E KOLLURI, V. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery 3*, 2 (1999), 131–169.

[42] QUINLAN, J. R. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993.

[43] ROBINSON, J. A. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM), 12(1)* (January 1965), 23–41.

[44] SRINIVASAN, A. A study of two sampling methods for analysing large datasets with ilp. In *Data Mining and Knowledge Discovery, 3(1)* (1999), pp. 95–123.

[45] SRINIVASAN, A. The aleph manual, 2005.

[46] VALIANT, L. G. A theory of the learnable. In *Communications of the ACM 27, 11* (1984), pp. 1134–1142.

[47] VAN BENTHEM, J. *Modal Correspondence Theory.* PhD thesis, Department of Mathematics, University of Amsterdam, 1978.

[48] ZELEZNY, F. Efficient construction of relational features. In *Proceedings of the 4th Int. Conf. on Machine Learning and Applications* (2005), IEEE Computer Society Press, pp. 259–264.