Marcos Aurélio Domingues

Implementação e Desenvolvimento de Técnicas de Descoberta de Conhecimento e Tratamento de Incerteza com Ênfase na Teoria de Conjuntos Aproximados

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtênção do título de Bacharel em Ciência da Computação.

Orientador
Prof. Joaquim Quinteiro Uchôa

Lavras
Minas Gerais - Brasil
2002

Marcos Aurélio Domingues

Implementação e Desenvolvimento de Técnicas de Descoberta de Conhecimento e Tratamento de Incerteza com Ênfase na Teoria de Conjuntos Aproximados

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtênção do título de Bacharel em Ciência da Computação.

Aprovada em *06 de Fevereiro de 2002*

Prof. Antônio Maria Pereira de Resende

Prof. Jones Oliveira de Albuquerque

Prof. Joaquim Quinteiro Uchôa
(Orientador)

Lavras
Minas Gerais - Brasil

À minha família.
Com carinho especial, a meus
pais Isabete e José Pedro e a meus
irmãos Marcelo e Márcio.

Agradecimentos

Agradeço a Deus por ter me ajudado a vencer mais uma etapa da minha vida.
Agradeço ao Prof. MSc. Joaquim Quinteiro Uchôa, pela dedicação e incentivo durante a orientação deste trabalho.
Agradeço aos meus tios Antônio e Rita, por terem me apoiado e incentivado durante os meus estudos.
Agradeço aos meus avós Osvaldo e Idalina, por terem me apoiado durante os meus estudos.
Agradeço aos meus familiares, pelo apoio e incentivo.
Agradeço ao PIBIC/CNPq, pelo apoio financeiro.

Resumo

A Teoria de Conjuntos Aproximados (TCA), tem sido utilizada em várias áreas de pesquisa, principalmente naquelas relacionadas com representação de conhecimento e aprendizado de máquina. Este trabalho tem por objetivo investigar a possibilidade do uso do formalismo de funções aproximadas (uma extensão dos conceitos básicos da TCA ao conceito de funções) para o desenvolvimento de métodos de Aprendizado Indutivo de Máquina, e apresentar uma *shell* de um sistema especialista.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Uma das mais importantes áreas de atuação da Inteligência Artificial é o desenvolvimento de Sistemas Baseados em Conhecimento (SBCs), sendo que estes se apresentam principalmente sob a forma de Sistemas Especialistas (SEs). Para alguns autores, inclusive, SBCs e SEs representam o mesmo tipo de software.

Em linha gerais, SBCs são definidos como programas de computador que resolvem problemas utilizando conhecimento representado explicitamente e que, não fosse essa representação, exigiriam um especialista humano no domínio do problema para a sua solução.

Sob este ponto de vista, SBCs devem ser capazes de representar, manipular e comunicar informações. É fato que tais sistemas devem estar preparados para modelar e tratar informações consideradas imperfeitas. Muitas vezes, o que se convenciona chamar de informações imperfeitas abrange informações imprecisas, inconsistentes, parcialmente ignoradas e mesmo incompletas. Como apontado em [Bonissone (1991)],

> a presença da incerteza em SBCs pode se originar de várias fontes: da confiabilidade parcial que se tem na informação, da imprecisão inerente da linguagem de representação na qual a informação é expressa, da não completeza da informação e da agregação/sumarização da informação que provêm de múltiplas fontes.

Observe que os problemas relacionados com incerteza acontecem em todo SBC. E apesar de muitas vezes o tratamento de incerteza ser realizado sem nenhuma fundamentação teórica, existem vários formalismos disponíveis, permitindo a

obtenção de resultados melhores e mais seguros. Dentre as várias técnicas existentes para o tratamento de incerteza em representação de conhecimento, bem como para descoberta de conhecimento, podem ser citadas:

- Teoria de Conjuntos Aproximados (TCA) [Pawlak (1982)], [Pawlak (1991)], [Nicoletti & Uchôa (1997)], [Uchôa (1998)];
- Teoria de Conjuntos Fuzzy (TCF) [Zadeh (1965)], [Klir & Yuan (1995)];
- Teoria de Dempster-Shaffer (TDS) [Shafer (1976)], [Uchôa et alii(1997)];
- Redes Neurais Artificiais (RNAs) [McCulloch & Pitts (1943)], [Fausett (1993)].

A Teoria de Conjuntos Aproximados (TCA) foi proposta em [Pawlak (1982)], como um novo modelo matemático para representação do conhecimento, tratamento de incerteza e classificação aproximada. Devido a estas características, tem-se utilizado esta teoria em Inteligência Artificial, especialmente nas áreas de:

- aprendizado de máquina;
- aquisição de conhecimento;
- raciocínio indutivo;
- descoberta de conhecimento em base de dados.

Em [Uchôa (1998)] pode ser verificado que a TCA pode ser utilizada com sucesso para a implementação de métodos de representação de conhecimento incerto, bem como um formalismo subsidiando aprendizado de máquina.

Este trabalho tem como objetivo propor o desenvolvimento de um algoritmo de aprendizado indutivo de máquina baseado em funções aproximadas (uma extensão dos conceitos básicos da TCA ao conceito de funções) e o desenvolvimento de uma *shell* para um sistema especialista que utilize este algoritmo.

O trabalho está organizado da seguinte forma: no Capítulo 2 são apresentados Sistemas Baseados em Conhecimento (SBCs), evidenciando a área de Aprendizado de Máquina (AM), como uma das maneiras de realizar Aquisição de Conhecimento para a construção da Base de Conhecimento em SBCs. No Capítulo 3 são apresentados os principais conceitos da TCA e descritas as medidas mais relevantes propostas na teoria. O Capítulo 4 demonstra a extensão dos conceitos básicos da TCA aos conceitos de relação e função aproximada, que serão utilizados como subsídio para o desenvolvimento de um algoritmo de aprendizado de máquina

baseado nesses conceitos. No Capítulo 5 é proposto o desenvolvimento do algoritmo de aprendizado indutivo denominado *lmurf - Learning Machine using ROugh Functions*, o qual se fundamenta na extensão dos conceitos da TCA ao conceito de funções. O Capítulo 6 apresenta uma *shell* de um sistema especialista desenvolvida para este algoritmo. No Capítulo 7 é apresentado conclusões e possíveis melhorias para o algoritmo *lmurf*. O Apêndice A apresenta o pseudocódigo dos algoritmos desenvolvidos.

Capítulo 2

Sistemas Baseados em Conhecimento

2.1 Introdução

Uma das mais proeminentes áreas em Inteligência Artificial é a de Sistemas Baseados em Conhecimento (SBCs), ou uma de suas principais formas, os Sistemas Especialistas (SEs).

O objetivo deste capítulo (baseado no capítulo 2 de [Uchôa (1998)]) é apresentar os principais componentes necessários a construção de SBCs e introduzir o conceito de Aprendizado e Máquina como um método de Aquisição de Conhecimento para a construção da Base de Conhecimento do SBC.

2.2 Sistemas Baseados em Conhecimento

SBCs são definidos formalmente como programas de computador que resolvem problemas utilizando conhecimento representado explicitamente e que, não fosse essa representação, exigiriam um especialista humano no domínio do problema para a sua solução. Conhecimento e processo de resolução de problemas são pontos críticos e essenciais durante o desenvolvimento de um SBC. A arquitetura básica de um SBC pode ser visualizada na Figura 2.1.

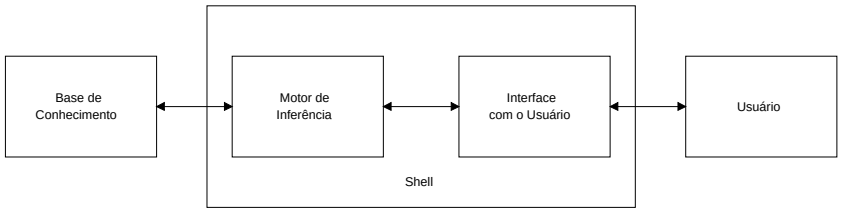


Figura 2.1: Arquitetura básica de um SBC

Um SBC possui, então, três módulos principais, a saber:

1. **Base de Conhecimentos (BC):** contém o conhecimento específico do domínio da aplicação. É composto de fatos sobre o domínio, regras que descrevem relações no domínio e métodos e heurísticas para resolução de problemas no domínio.

2. **Motor de Inferência (MI):** mecanismo responsável pelo processamento do conhecimento da BC, utilizando-se de alguma linha de raciocínio. Implementa as estratégias de inferência e controle do SBC. Quando o conhecimento do SBC está expresso como regras, as estratégias de controle empregadas pelo MI normalmente são encadeamento para trás *(backward chaining)* ou encadeamento para frente *(forward chaining)*. Quando o MI usa a estratégia de encadeamento para trás, uma lista de hipóteses é pesquisada procurando reunir evidências para viabilizar a conclusão da validade de alguma(s) dela(s). Esta estratégia corresponde à pergunta: É possível provar as hipóteses a partir dos dados disponíveis? Se a estratégia de controle for encadeamento para frente, o MI parte dos dados e, com base nas regras de conhecimento, deduz outras asserções procurando chegar à solução do problema. Essa estratégia corresponde à pergunta: O que é possível concluir a partir dos dados disponíveis?

3. **Interface com o Usuário (IU):** módulo responsável pela comunicação entre o usuário e o sistema. Deve fornecer, também, justificativas e explicações referentes às conclusões obtidas na BC, bem como do raciocínio utilizado.

Sistemas Especialistas (SEs) constituem uma importante subclasse dos SBCs. Um SE pode ser definido como um SBC que resolve problemas bem específicos do mundo real, problemas esses que requerem considerável habilidade, conhecimento

e heurísticas para sua resolução. Na literatura, os termos Sistemas Baseados em Conhecimento e Sistemas Especialistas são, muitas vezes, utilizados quase sem distinção. Adota-se essa abordagem nesse texto.

A grande maioria dos SBCs apresenta-se sob a forma de Sistemas Baseados em Regras (SBRs), que utilizam-se de regras para codificação do conhecimento. Uma regra é uma estrutura da forma $(a \Rightarrow b)$, e é lida `se` a `é verdade, então` b `também é verdade`, ou resumidamente `se` a`, então` b. O termo a é denominado antecedente da regra, e expressa as condições de execução da mesma. O termo b, por sua vez, é denominado conseqüente da regra e, caso satisfeitas as condições expressas no antecedente, torna-se um fato. Um fato é uma regra da forma $(\Rightarrow b)$, ou resumidamente (b), onde o antecedente é vazio, o que implica a verdade incondicional de b.

Observe que, em um SBR, o Motor de Inferência é um interpretador de regras. Os SBCs também podem apresentar-se sob a forma de Sistemas Baseados em Árvores de Decisão, que utilizam-se de uma árvore de decisão para codificação do conhecimento. Numa árvore de decisão, os nós não-folhas são considerados como os antecedentes da regra e sua análise permite que se percorra a árvore em busca de um nó folha que é considerado como o conseqüente da regra. Note que, em um SBC baseado em árvore de decisão, o Motor de Inferência é um algoritmo que percorre uma árvore em busca de um nó folha. Os nós são analisados, indicando o caminho que se deve seguir na árvore, sendo que os nós não-folha possuem um ou mais valores (atributos) e compõem o antecedente de uma regra (condições de uma regra), enquanto que os nós folhas que possuem também um ou mais valores irão compor o conseqüente da regra (conclusão de uma regra).

A capacidade de resolver problemas que demandam informação referente a um determinado domínio de conhecimento e explicar seu comportamento tornaram-se os aspectos principais de um SBC. Um SBC deve ser capaz de explicar seu comportamento e suas decisões ao usuário, respondendo o *porque* e *como* chegou a uma determinada solução. De uma maneira geral as perguntas *por que* referem-se a qual conhecimento respalda a conclusão; as perguntas *como*, por sua vez, referem-se aos passos de raciocínio seguidos para determinar a solução do problema. Esta característica é especialmente necessária quando o SBC lida com domínios incertos (diagnóstico médico, por exemplo); a explicação pode, de certa forma, aumentar o grau de confiança que o usuário deposita no sistema, ou então, ajudá-lo a encontrar alguma falha no raciocínio do sistema.

Uma outra característica frequentemente necessária é a habilidade de lidar com incertezas e informações incompletas: a informação a respeito do problema a ser

resolvido pode estar incompleta ou ser parcialmente confiável, bem como as relações no domínio do problema podem ser aproximadas. Espera-se, também, que um SBC seja flexível o suficiente para permitir, facilmente, a acomodação de novo conhecimento.

2.3 Aquisição de Conhecimento

Uma das principais atividades relacionadas ao desenvolvimento de Sistemas Baseados em Conhecimento consiste na transferência do conhecimento - informações e/ou formas de condução do raciocínio - do especialista humano (ou de qualquer outra fonte) à Base de Conhecimento do SBC. Este processo é conhecido como Aquisição de Conhecimento (AC) e é, reconhecidamente, o processo mais difícil durante o desenvolvimento de SBCs, exigindo um grande investimento em tempo e esforço.

Além da extração do conhecimento necessário, tal conhecimento deve ser traduzido para o esquema formal de representação usado pelo SBC e deve ser repetidamente refinado, até que o sistema atinja um grau de desempenho próximo ao de um especialista humano, quando da resolução do problema. Sobre esse processo, pode ser observado em [Nicoletti (1994)] que:

> ... o processo de extração do conhecimento do especialista envolve um intenso questionamento que, em certas situações, pode interferir na sua própria percepção de como elabora o raciocínio. Geralmente um especialista acha difícil detalhar descrições de seu conhecimento e de como o usa; por outro lado, muitas vezes é difícil para o engenheiro do conhecimento traduzir com exatidão aquele conhecimento, geralmente amplo e multifacetado, em uma linguagem de representação restrita. Devido a essas dificuldades, muitas vezes o conhecimento extraído pode ser inconsistente (como conseqüência de diferenças individuais entre especialistas), incompleto e impreciso.

Existe um vasto conjunto de métodos e ferramentas que facilitam a tarefa de AC. A disponibilidade dessa variedade reflete o fato da AC ser um processo multidimensional, podendo ocorrer em diferentes estágios do desenvolvimento de um SBC, e podendo envolver vários tipos de conhecimento. Uma classificação proposta em [Boose (1989)] agrupa os métodos e técnicas para AC em:

1. **manuais -** que tipicamente consistem de entrevistas e análise de protocolos;

2. **baseadas em computador:**

   (a) ***interativas*** **-** que englobam, principalmente, ferramentas que entrevistam o especialista, que fazem análise textual, que extraem e analisam conhecimento de múltiplas fontes separadamente e as combinam para uso;

   (b) ***baseadas em aprendizado*** **-** as quais, via de regra, generalizam situações específicas em conceitos.

2.4 Aprendizado de Máquina

Como observado em [Michalski & Tecuci (1993)], aprendizado pode ser caracterizado como um processo multidimensional que, via de regra, ocorre através da aquisição de conhecimento declarativo, do desenvolvimento de habilidades motoras e cognitivas através de instrução e prática, da organização do conhecimento existente em representações mais efetivas, da descoberta de novos fatos e/ou teorias através da observação e experimentação ou, então, através da combinação e/ou composição dessas dimensões.

Um sistema inteligente deve ser capaz de formar conceitos, i.e., classes de entidades unidas por algum princípio. Tal princípio pode ser o fato das entidades terem um uso ou objetivo comum ou, simplesmente, terem características perceptuais similares. Para a utilização do conceito, o sistema deve também desenvolver métodos eficientes de reconhecimento de pertinência de uma dada entidade ao conceito. O estudo e a modelagem de processos pelos quais o sistema adquire, refina e diferencia conceitos são objetivos da área de pesquisa chamada de *Aprendizado de Conceito*, uma subárea de AM.

Em Aprendizado de Conceito, o termo *conceito* usualmente indica uma classe de equivalência de entidades, que pode ser descrita via um conjunto relativamente limitado de expressões e que deve ser suficiente para a diferenciação entre conceitos. Entidades individuais na classe são chamadas de *instâncias* do conceito. O fato do conceito ser definido como uma classe de equivalência torna cada instância de uma classe igualmente representativa do conceito em questão; tal definição estabelece, também, os limites da descrição daqueles conceitos - uma entidade satisfaz ou não satisfaz o conceito. Mais detalhes sobre aprendizado de conceito podem ser encontrados em [Mitchell (1997)] e [Langley (1996)].

Observe que, dada a complexidade existente no processo de aprendizado, não há em relação a ele uma definição única e tampouco um entendimento suficiente.

Podem ser encontradas na literatura definições que se diferenciam entre si pela ênfase dada a diferentes aspectos do processo de aprendizado.

Uma abordagem mais limitada de AM, geralmente adotada por pessoas que trabalham especificamente com SBCs, é a de que aprendizado é a aquisição de conhecimento explícito. Muitos SBCs representam conhecimento como um conjunto de regras que necessitam ser adquiridas, organizadas e estendidas. Isto enfatiza a importância de tornar explícito o conhecimento adquirido, de maneira que ele possa facilmente ser verificado, modificado e explicado.

Na literatura há diversas taxionomias de sistemas de AM. Algumas das várias razões que dão origem a esta diversidade de classificações são: diferentes estratégias de aprendizado adotadas, existência de diferentes representações do conhecimento, domínios de aplicação variados, etc. Uma possível taxionomia de métodos de Aprendizado de Máquina classifica os paradigmas em:

Aprendizado Simbólico: aquisição de conceitos expressos em símbolos, através de conjuntos de exemplos;

Aprendizado Baseado em Instância: métodos que simplesmente armazenam as instâncias de treinamento;

Aprendizado Baseado em Algoritmos Genéticos: que inclui ambos: algoritmos genéticos, que induzem hipóteses descritas usando cadeias de bits, e programação genética, que induzem hipóteses descritas como programas;

Aprendizado Conexionista: buscam modelar o processo de funcionamento dos neurônios e/ou áreas do cérebro humano;

Aprendizado Analítico: aprendizado baseado em explicações e certas formas de métodos de aprendizado analógicos e baseados em casos.

Note que, dentre todos esses modelos, apenas o Aprendizado Analítico é de natureza dedutiva; todos os demais são indutivos.

2.4.1 Aprendizado Indutivo de Máquina

Entre os vários modelos existentes para aprendizado apontados anteriormente, o aprendizado simbólico conhecido como aprendizado indutivo baseado em exemplos é o que mais tem sido pesquisado e o que mais tem contribuído efetivamente para a implementação de sistemas de aprendizado de máquina. A partir de um conjunto de exemplos, expressões para tarefas classificatórias podem ser aprendidas

(induzidas) como, por exemplo, diagnóstico de doenças, previsão meteorológica, predição do comportamento de novos compostos químicos, predição de propriedades mecânicas de metais com base em algumas de suas propriedades químicas, etc. A Figura 2.2, extraída de [Shaw & Gentry (1990)], ilustra esse processo.

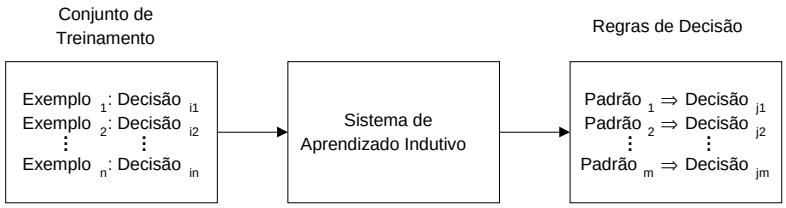


Figura 2.2: Esquema geral de aprendizado indutivo de regras

No aprendizado indutivo baseado em exemplos, também referenciado como aprendizado indutivo, o conjunto de exemplos, também denominado de *conjunto de treinamento*, é fornecido ao sistema por um instrutor ou pelo ambiente (base de dados, sensores, etc.). Esse conjunto de treinamento é geralmente composto de *exemplos positivos* (exemplos do conceito) e *exemplos negativos* (contra-exemplos do conceito). A indução do conceito corresponde a uma busca no espaço de hipóteses, de forma a encontrar aquelas que *melhor* classificam os exemplos. Nesse contexto, *melhor* pode ser definido em termos de precisão e compreensibilidade.

De uma maneira geral, um sistema que aprende a partir de exemplos recebe como dados informações na forma de situações específicas, cada uma delas devidamente classificadas (geralmente por especialista humano no domínio), caracterizando o que se convenciona chamar de *aprendizado supervisionado*, e produz, como resultado, uma(s) hipótese(s) que generaliza(m) aquelas situações inicialmente fornecidas. Entre as principais características de sistemas de aprendizado indutivo de máquina, encontram-se:

Incrementabilidade: num sistema incremental, a expressão do conceito vai sendo construída exemplo a exemplo e implica constante revisão do conceito (ou conceitos) até então formulado(s). Um novo exemplo pode, eventualmente, causar um rearranjo da expressão do conceito. A expressão do conceito vai se modificando à medida que os exemplos vão se tornando disponíveis. No caso não-incremental, o conjunto de treinamento deve estar totalmente disponível no início do processo de aprendizado, uma vez que o conceito é induzido considerando-se todos os exemplos de uma vez.

Teoria de Domínio: no caso em que o sistema não possui informação a respeito do problema de aprendizado sendo tratado, o sistema induz a expressão do conceito apenas a partir dos exemplos disponíveis. Entretanto, para que a solução de problemas difíceis de aprendizado sejam encontradas, é fundamental que um volume substancial de conhecimento sobre o problema esteja disponível, de maneira a subsidiar a indução do conceito. Esse conhecimento prévio existente é conhecido como Teoria de Domínio ou Conhecimento de Fundo.

Linguagem de Descrição: em inferência indutiva, os exemplos, a Teoria de Domínio e as hipóteses formuladas necessitam ser expressos em alguma linguagem e geralmente são utilizadas linguagens formais.

Uma vez desenvolvido o sistema de AM, é interessante avaliar o resultado deste sistema. De uma forma geral, entre os critérios mais usuais para se medir o sucesso de um sistema de AM encontram-se:

Precisão de classificação: é geralmente definida como o percentual de exemplos corretamente classificados pela hipótese induzida.

Transparência da descrição induzida do conceito: em muitos domínios de aplicação (por exemplo, diagnóstico médico), é fundamental que a descrição do conceito, gerada por um sistema de AM, seja compreensível pelo ser humano. O entendimento do conceito, por parte do ser humano, não apenas aumenta a credibilidade do sistema de AM, mas também permite que o conceito possa ser assimilado e utilizado pelo especialista humano. Em muitas situações a transparência da descrição é medida pelo número de descritores e operadores usados na descrição do conceito.

Complexidade computacional: está relacionada com os recursos computacionais necessários (tempo e espaço) para realizar o aprendizado.

Como já comentado, para a expressão de um modelo de aprendizado indutivo são necessárias linguagens que descrevam os exemplos de treinamento, assim como linguagens que descrevam os conceitos aprendidos. Vários formalismos lógicos têm sido usados em sistemas de aprendizado indutivo para a representação de exemplos e conceitos. Em geral, distinguem-se dois tipos de descrição: *descrição baseada em atributos* (ou proposicional) e *descrição relacional*. Em uma descrição baseada em atributos, exemplos são descritos como vetores de pares atributo-valor e uma classe associada. Em uma descrição relacional, exemplos do

conceito a ser aprendido são fornecidos como fatos. O aprendizado relacional se caracteriza também por viabilizar o uso da Teoria de Domínio; a incorporação desse tipo de conhecimento ao processo de aprendizado quase sempre contribui para uma expressão mais concisa e mais natural do conceito a ser aprendido.

2.4.2 Tratamento de Incerteza em Aprendizado de Máquina

Um dos principais problemas em Aprendizado de Máquina é o de representação de incerteza. É fato que todo SBC possui problemas relativos à incerteza. Esta pode se manifestar de diversas formas: imprecisão, incompleteza, inconsistência, etc. Torna-se, portanto, desejável que a estratégia de AM implementada, bem como o próprio SBC, possua mecanismos de tratamento da incerteza. Esses mecanismos freqüentemente surgem sob a forma de medidas de incerteza.

A Teoria de Conjuntos Aproximados, formalismo tratado neste trabalho, possui mecanismos para expressão de, entre outros, um tipo fundamental de incerteza: a indiscernibilidade. A indiscernibilidade surge quando não é possível distinguir objetos de um mesmo conjunto; representa a situação em que esses objetos parecem todos ser um único objeto.

Representação da indiscernibilidade, bem como técnicas para a classificação de objetos, usando a TCA, são as bases que sustentam o presente trabalho e serão abordadas em detalhes nos próximos capítulos.

Capítulo 3

Elementos da Teoria de Conjuntos Aproximados

3.1 Introdução

Um conjunto aproximado é um modelo matemático usado para tratar um tipo de incerteza - *a indiscernibilidade*. De uma forma simplista, conjuntos aproximados podem ser considerados conjuntos com fronteiras nebulosas, ou seja, conjuntos que não podem ser caracterizados precisamente utilizando-se o conjunto de atributos disponíveis. Uma das principais vantagens da Teoria de Conjuntos Aproximados (TCA) é a de não necessitar de qualquer informação adicional ou preliminar a respeito de dados, tais como: distribuição de probabilidade, atribuição de crenças, grau de pertinência ou possibilidade.

Este capítulo tem dois objetivo, a saber: o primeiro é apresentar os principais conceitos e medidas da Teoria de Conjuntos Aproximados (TCA), como subsídio para os estudos realizados e, conseqüentemente, para a compreensão dos próximos capítulos; o segundo é apresentar os principais conceitos de Sistemas de Representação de Conhecimento (SRCs) sob a ótica da TCA. Este capítulo foi baseado em [Pawlak (1991)] e em [Uchôa (1998)].

3.2 Teoria de Conjuntos Aproximados - TCA

3.2.1 Espaços Aproximados

Um *espaço aproximado* é um par ordenado $A = (U, R)$, onde:

- U é um conjunto não vazio, denominado *conjunto universo*;

- R é uma relação de equivalência sobre U, denominada *relação de indiscernibilidade*. Dados $x, y \in U$, se xRy então x e y são *indiscerníveis* em A, ou seja, a classe de equivalência definida por x é a mesma que a definida por y, i.e., $[x]_R = [y]_R$.

As classes de equivalência induzidas por R em U são denominadas *conjuntos elementares*. Se E é um conjunto elementar, $des(E)$ denota a descrição dessa classe de equivalência. Essa descrição é função do conjunto de atributos que define R. Note que, dados $x, y \in E$, onde E é um conjunto elementar em A, x e y são indiscerníveis, i.e., no espaço $A = (U, R)$ não se consegue distinguir x de y, pois $des(x) = des(y) = des(E)$.

3.2.2 Classificação Aproximada de um Conjunto

Dado um espaço aproximado $A = (U, R)$ e um conjunto $X \subseteq U$, com o objetivo de verificar o quão bem X é representado pelos conjuntos elementares de A, são definidas:

- a *aproximação inferior* de X em A, $A_{A-inf}(X)$, como a união de todos os conjuntos elementares que estão contidos em X:

$$A_{A-inf}(X) = \{x \in U \mid [x]_R \subseteq X\};$$

- a *aproximação superior* de X em A, $A_{A-sup}(X)$, como a união de todos os conjuntos que possuem intersecção não vazia com X:

$$A_{A-sup}(X) = \{x \in U \mid [x]_R \cap X \neq \varnothing\}.$$

Nas notações utilizadas, quando o espaço aproximado for conhecido e não houver risco de confusão, a referência ao espaço será abolida. Assim, por exemplo, $A_{sup}(X)$ será usado em substituição a $A_{A-sup}(X)$.

3.2.3 Regiões do Espaço Aproximado

Dado um espaço aproximado $A = (U, R)$ e $X \subseteq U$, as aproximações inferior e superior permitem a classificação do espaço aproximado em regiões:

1. *região positiva* de X em A, formada por todas as classes de equivalência de U contidas inteiramente no conjunto X e dada por,

$$pos_A(X) = A_{A-inf}(X)$$

2. *região negativa* de X em A, formada pelos conjuntos elementares de A que não estão contidos na aproximação superior de X e dada por,

$$neg_A(X) = U - A_{A-sup}(X)$$

3. *região duvidosa* de X em A, formada pelos elementos que pertencem a aproximação superior mas não pertencem à aproximação inferior e dada por,

$$duv_A(X) = A_{A-sup}(X) - A_{A-inf}(X)$$

Exemplo 1 *Seja U um conjunto universo e R uma relação de equivalência em U, definindo o espaço aproximado $A = (U, R)$. Seja também X, como ilustra a Figura 3.1. A aproximação inferior e a aproximação superior de X em $A = (U, R)$ são mostradas na Figura 3.2(a) e 3.2(b). A Figura 3.3, por sua vez, apresenta as regiões de X.*

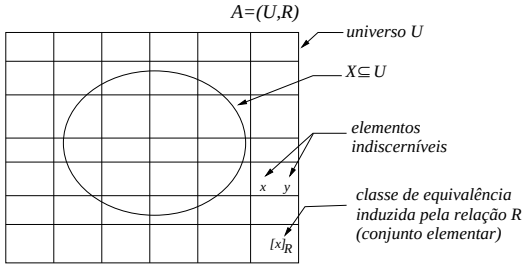
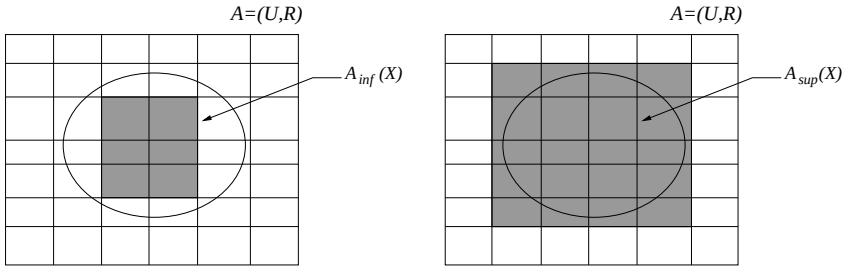


Figura 3.1: Conjunto X no espaço aproximado $A = (U, R)$.



(a) Aproximação Inferior de X. (b) Aproximação Superior de X.

Figura 3.2: Aproximações de X em A.

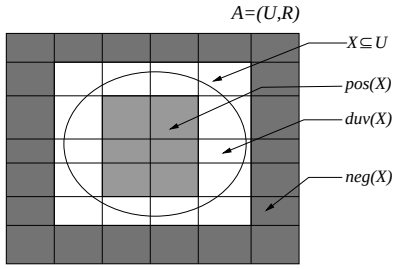


Figura 3.3: Regiões de X em A.

Seja $A = (U, R)$ um espaço aproximado e seja $X \subseteq U$. O conjunto X pode ou não ter suas fronteiras claramente definidas em função das descrições dos conjuntos elementares de A. Isso leva ao conceito de conjuntos aproximados: um *conjunto aproximado* em A é a família de todos os subconjuntos de U que possuem a mesma aproximação inferior e a mesma aproximação superior em A. Ou seja, possuem a mesma região positiva, negativa e duvidosa.

3.2.4 Medidas em um Espaço Aproximado

Dado um espaço aproximado $A = (U, R)$ e $X \subseteq U$, pode-se definir medidas que indiquem o quão bem $X \subseteq U$ pode ser representado pelo espaço aproximado

$A = (U, R)$.

Acuracidade de um Aproximação

Seja um espaço aproximado $A = (U, R)$. Com a finalidade de verificar o quão bem o conjunto $X \subseteq U$ pode ser representado por A, são definidas as seguintes medidas:

- *medida interna* de X em A, $\omega_{A-inf}(X) = |A_{A-inf}(X)|$;
- *medida externa* de X em A, $\omega_{A-sup}(X) = |A_{A-sup}(X)|$;
- *qualidade da aproximação inferior* de X em A, $\gamma_{A-inf}(X) = \frac{\omega_{A-inf}(X)}{|U|}$;
- *qualidade da aproximação superior* de X em A, $\gamma_{A-sup}(X) = \frac{\omega_{A-sup}(X)}{|U|}$.

A medida de *acuracidade* de X é definida pela relação entre a cardinalidade da aproximação inferior de X pela cardinalidade da aproximação superior de X, em símbolos:

$$\omega_A(X) = \frac{\omega A_{inf^A}(X)}{\omega A_{sup^A}(X)} = \frac{|A_{inf}(X)|}{|A_{sup}(X)|}.$$

Quando A é conhecido e não há risco de confusão, escreve-se ω_{inf}, ω_{sup}, γ_{inf}, γ_{sup} e ω em substituição a ω_{A-inf}, ω_{A-sup}, γ_{A-inf}, γ_{A-sup} e ω_A.

Índice Discriminante de Atributos

Seja o espaço aproximado $A = (U, R)$ e $X \subseteq U$. O número total de objetos em U que podem, com certeza, ser classificados em dois subconjuntos disjuntos, X e $U - X$, é igual ao número de objetos que não pertencem à região $duv(X)$. Esse número é, pois:

$$|U - duv(X)| = |U - (A_{sup}(X) - A_{inf}(X))| = |U| - |A_{sup}(X) - A_{inf}(X)|.$$

Dessa forma, a razão:

$$\alpha_R(X) = \frac{|U| - |A_{sup}(X) - A_{inf}(X)|}{|U|},$$

definida com *índice discriminante de R com relação a X*, fornece uma medida do grau de certeza na determinação da pertinência (ou não) de um elemento de U no conjunto X.

Função de Pertinência Aproximada

Em [Pawlak (1994)] é proposta um função de pertinência aproximada, que permite calcular a pertinência de um elemento do universo U a qualquer das regiões definidas por um conjunto $X \subseteq U$. Dado um espaço aproximado $A = (U, R), X \subseteq U$ e $x \in U$, a pertinência de x a X no espaço A é dada por:

$$\mu_X^A(x) = \frac{|[x]_R \cap X|}{|[x]_R|},$$

onde $[X]_R$ denota a classe de equivalência induzida pela relação de equivalência R, que contém o elemento x (e todos os seus equivalentes, por R). Quando A é conhecido e não há risco de confusão, escreve-se μ_X em substituição a μ_X^A.

3.3 Representação de Conhecimento

3.3.1 Sistemas de Representação de Conhecimento (SRCs)

Os conceitos da TCA são utilizados principalmente no contexto de Sistemas de Representação de Conhecimento. Um *Sistema de Representação de Conhecimento* (SRC) é uma quádrupla $S = (U, Q, V, \rho)$, onde U é o universo finito de S.

Os elementos de U são chamados objetos, que são caracterizados por um conjunto de atributos Q e seus respectivos valores. O conjunto de valores de atributos é dado por $V = \bigcup_{q \in Q} V_q$, onde V_q é o conjunto de valores do atributo q. Por sua vez, ρ: $U \times Q \to V$ é uma *função de descrição* tal que $\rho(x, q) \in V_q$, para $x \in U$ e $q \in Q$.

Dado um SRC $S = (U, Q, V, \rho)$, é importante observar que cada subconjunto de atributos $P \subseteq Q$ define um único espaço aproximado $A = (U, \tilde{P})$ onde $\tilde{P}$ é a relação de indiscernibilidade (equivalência) induzida por P.

Exemplo 2 *Seja o SRC representado pela Tabela 3.1, onde* $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$, $Q = \{a, b, c, d\}$ *e* $V = \{1, 2, 3, 4, 5, 6\}$. *Neste sistema de representação de conhecimento tem-se:*

- $\rho(x_1, a) = 4$;
- $\rho(x_2, b) = 1$;
- $\rho(x_5, d) = 1$;
- $\rho(x_3, c) = 5$.

Tabela 3.1: SRC onde $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$ e $Q = \{a, b, c, d\}$

U	a	b	c	d
x_1	4	1	3	1
x_2	6	1	4	1
x_3	6	2	5	2
x_4	6	1	5	1
x_5	4	2	4	1
x_6	4	2	4	2
x_7	4	2	3	1
x_8	4	1	4	1
x_9	6	1	4	2
x_{10}	4	1	4	2

Exemplo 3 *Seja S o SRC do Exemplo 2 e $P = \{d\}$. Neste caso tem-se que os elementos x_3, x_6, x_9 e x_{10} são indiscerníveis com relação a P pois possuem o mesmo valor. Além disto, $A = (U, \tilde{P})$ é um espaço aproximado e seus conjuntos elementares são $E_1 = \{x_1, x_2, x_4, x_5, x_7, x_8\}$, $E_2 = \{x_3, x_6, x_9, x_{10}\}$.*

O Exemplo 4 demonstra o cálculo dos principais conceitos e medidas da TCA.

Exemplo 4 *Seja S o SRC definido no Exemplo 2 e seja $P = \{a, b\}$, um subconjunto de atributos de Q. Nesse caso, os conjuntos elementares do espaco aproximado $A = (U, \tilde{P})$ são: $E_1 = \{x_1, x_8, x_{10}\}$, $E_2 = \{x_2, x_4, x_9\}$, $E_3 = \{x_3\}$ e $E_4 = \{x_5, x_6, x_7\}$. Seja $X = \{x_2, x_3\}$. Tem-se que:*

$$A_{A-inf}(X) = \{x_3\}$$
$$A_{A-sup}(X) = \{x_2, x_3, x_4, x_9\}$$

$$\alpha_P(X) = \frac{10-|4-1|}{10} = \frac{7}{10} = 0.7$$

$$\omega_{A-inf}(X) = 1$$
$$\omega_{A-sup}(X) = 4$$
$$\omega_A(X) = 0.25$$

$$\gamma_{A-inf}(X) = 0.1$$
$$\gamma_{A-sup}(X) = 0.4$$

$$\mu_X^A(x_3) = 1$$

3.3.2 Tabelas de Decisão

No contexto da TCA o interesse recai, principalmente, sobre tabelas de decisão, um tipo particular de SRC. Uma *tabela de decisão* é um SRC onde os atributos de Q são divididos em condições e decisões. Tem-se então $Q = C \cup D$, onde C é o conjunto das condições e D o conjunto das decisões. Como geralmente o conjunto D é unitário, uma tabela de decisão é descrita por $S = (U, C \cup \{\delta\}, V, \rho\}$, onde U, V e ρ são tais como num SRC, C é o conjunto de condições e δ é o atributo de decisão. Por $Class_S(\delta)$ entende-se a classificação de S, i.e., a família de conjuntos elementares do espaço aproximado induzido por $\{\delta\}$.

Exemplo 5 *Seja S o SRC do Exemplo 2. Seja d o atributo de decisão em S, i.e., $\delta = d$. S é uma tabela de decisão, onde $C = \{a, b, c\}$. Os conjuntos elementares do espaço aproximado induzido por C são $\{x_1\}$, $\{x_2, x_9\}$, $\{x_3\}$, $\{x_4\}$, $\{x_5, x_6\}$, $\{x_7\}$ e $\{x_8, x_{10}\}$. Por sua vez, a classificação de S é dada por $Class_S(\delta) = \{\{x_1, x_2, x_4, x_5, x_7, x_8\}, \{x_3, x_6, x_9, x_{10}\}\}$.*

Dada uma tabela de decisão $S = (U, C \cup \{\delta\}, V, \rho)$, é importante verificar o quão bem a família de conjuntos elementares induzidos pelas condições $P \subseteq C$ espelha a família de conjuntos elementares induzidos por $\{\delta\}$. Para isso, considerando o espaço aproximado induzido por P, são definidas:

- *região positiva* de δ induzida por P,

$$pos(P, \delta) = \bigcup_{X \in Class_S(\delta)} A_{P-inf}(X);$$

- *grau de dependência* de δ com relação a P,

$$\kappa(P, \delta) = \frac{|pos(P, \delta)|}{|U|};$$

- *fator de significância* de um atributo $a \in P$, com relação à dependência existente entre δ e P,

$$FS(a, P, \delta) = \frac{(\kappa(P, \delta) - (\kappa(P - \{a\}, \delta))}{\kappa(P, \delta)},$$

se $\kappa(P, \delta) > 0$.

Exemplo 6 *Seja S o SRC definido no Exemplo 2 e seja $P = \{a, b\}$. Os conjuntos elementares do espaço aproximado induzido por P são: $E_1 = \{x_1, x_8, x_{10}\}$, $E_2 = \{x_2, x_4, x_9\}$, $E_3 = \{x_3\}$ e $E_4 = \{x_5, x_6, x_7\}$. Por sua vez, $Class_S(d) = \{\{x_1, x_2, x_4, x_5, x_7, x_8\}, \{x_3, x_6, x_9, x_{10}\}\}$.*
Com isso, é possível verificar os seguintes resultados:

$$pos(P, d) = \bigcup_{X \in Class_S(d)} A_{P-inf}(X) =$$

$$= A_{P-inf}(\{x_1, x_2, x_4, x_5, x_7, x_8\}) \cup A_{P-inf}(\{x_3, x_6, x_9, x_{10}\}) =$$

$$= \{x_1, x_4, x_7\} \cup \{x_3\} = \{x_1, x_3, x_4, x_7\}$$

$$\kappa(P, d) = \frac{|pos(P, d)|}{|U|} = \frac{4}{10} = 0.4.$$

Para o cálculo de:

$$FS(a, P, d) = \frac{(\kappa(P,d) - (\kappa(P - \{a\}, d))}{\kappa(P,d)},$$

e de :

$$FS(b, P, d) = \frac{(\kappa(P,d) - (\kappa(P - \{b\}, d))}{\kappa(P,d)},$$

bem como o de :

$$FS(c, P, d) = \frac{(\kappa(P,d) - (\kappa(P - \{c\}, d))}{\kappa(P,d)},$$

é necessário calcular antes o valor de $\kappa(\{b, c\}, d)$, $\kappa(\{a, c\}, d)$ e $\kappa(\{a, b\}, d)$, respectivamente, para este exemplo, considerando-se que $P_1 = \{b, c\}$, $P_2 = \{a, c\}$ e $P_3 = \{a, b\}$, tem-se que as famílias dos conjuntos elementares dos espaços aproximados induzidos por P_1, P_2 e P_3 são dadas, respectivamente, por $\{\{x_1\}, \{x_2, x_8, x_9, x_{10}\}, \{x_5, x_6\}, \{x_7\}, \{x_3\}, \{x_4\}\}$, $\{\{x_1, x_7\}, \{x_5, x_6, x_8, x_{10}\}, \{x_3, x_4\}, \{x_2, x_9\}\}$ e $\{\{x_1, x_8, x_{10}\}, \{x_2, x_4, x_9\}, \{x_3\}, \{x_5, x_6, x_7\}\}$. Donde:

$$pos(P_1, d) = \{x_1, x_3, x_4, x_7\} \text{ e}$$
$$\kappa(P_1, d) = \frac{4}{10} = 0.4,$$

$$pos(P_2, d) = \{x_1, x_7\} \text{ e}$$
$$\kappa(P_2, d) = \frac{2}{10} = 0.2,$$

$$pos(P_3, d) = \{x_3\} \text{ e}$$
$$\kappa(P_3, d) = \frac{1}{10} = 0.1.$$

Portanto,

$$FS(a,P,d) = \frac{(\kappa(P,d)-(\kappa(\{b,c\},d))}{\kappa(P,d)} = \frac{\frac{4}{10}-\frac{4}{10}}{\frac{4}{10}} = 0,$$

$$FS(b,P,d) = \frac{(\kappa(P,d)-(\kappa(\{a,c\},d))}{\kappa(P,d)} == \frac{\frac{4}{10}-\frac{2}{10}}{\frac{4}{10}} = 0.5,$$

$$FS(c,P,d) = \frac{(\kappa(P,d)-(\kappa(\{a,b\},d))}{\kappa(P,d)} == \frac{\frac{4}{10}-\frac{1}{10}}{\frac{4}{10}} = \frac{3}{4} = 0.75.$$

Diz-se, ainda com respeito a uma tabela de decisão $S = (U, C \cup \{\delta\}, V, \rho\}$ e $P \subseteq C$ que P é *independente* com relação à dependência existente entre δ e P se, para todo subconjunto próprio $R \subset P$, for verdade que $pos(P,\delta) \neq pos(R,\delta)$, i.e., $\kappa(P,\delta) \neq \kappa(R,\delta)$. Caso haja algum $R \subset P$ tal que $pos(P,\delta) = pos(R,\delta)$, i.e., $\kappa(P,\delta) = \kappa(R,\delta)$, então P é dito ser *dependente* com relação à dependência existente entre δ e P.

3.3.3 Redução do Conjunto Inicial de Atributos

Um problema crucial no contexto de Sistemas de Representação de Conhecimento é o de encontrar subconjuntos do conjunto original de atributos com o mesmo poder discriminatório deste. A obtenção destes subconjuntos pode auxiliar tanto na redução de custo computacional em tarefas que utilizem-se de SRCs, como até mesmo custo temporal ou financeiro (caso atributos de difícil obtenção ou alto custo possam ser eliminados). No contexto da TCA, a obtenção de redutos está intimamente ligada à anàlise de dependência entre atributos.

Um conjunto $R \subset P$ é dito ser um *reduto* de P com relação à dependência existente entre δ e P se for independente com relação à dependência existente entre δ e R, e $pos(P,\delta) = pos(R,\delta)$, i.e., $\kappa(P,\delta) = \kappa(R,\delta)$.

Exemplo 7 *Seja S o SRC do Exemplo 2, onde d é o atributo de decisão em S. Como já visto, S e uma tabela de decisão, onde a classe de condições é dada por $C = \{a,b,c\}$ e a decisão é dada por $\{\delta\} = \{d\}$. A família $Class_S(d)$ foi determinada no Exemplo 5. Tem-se os seguintes resultados em S:*

$$\kappa(C,d) = \frac{|pos(C,d)|}{|U|} = \frac{|4|}{|10|} = 0.4$$

$$\kappa(\{a,b\},d) = \frac{|1|}{|10|} = 0.1$$

$$\kappa(\{a,c\},d) = \frac{|2|}{|10|} = 0.2$$

$$\kappa(\{b,c\},d) = \frac{|4|}{|10|} = 0.4$$

Neste caso C é dependente e $\{b,c\}$ é o único subconjunto de C a possuir um reduto de C, pois $\{b,c\}$ é o único subconjunto com o mesmo grau de dependência de C. Tem-se ainda:

$$\kappa(\{b\},d) = \frac{|0|}{|10|} = 0$$

$$\kappa(\{c\},d) = \frac{|2|}{|10|} = 0.2$$

Portanto, o conjunto $R = \{b,c\}$ é o único reduto do conjunto de condições com relação a dependência entre δ e C.

3.4 Comentários Finais

Neste capítulo, foram apresentados os principais conceitos e medidas da Teoria de Conjuntos Aproximados, com o objetivo de subsidiar os estudos de relações e funções aproximadas, conceitos que serão apresentados no Capítulo 4. Os conceitos foram utilizados no desenvolvimento de um algoritmo de aprendizado indutivo de máquina, apresentado no Capítulo 5.

Capítulo 4

Relações e Funções Aproximadas

4.1 Introdução

O objetivo principal deste capítulo é demonstrar que os conceitos básicos da Teoria de Conjuntos Aproximados podem ser estendidos aos conceitos matemáticos de relações e funções de conjuntos, como apresentado em [Uchôa (1998)] e em [Nicoletti; Uchôa & Baptistini (2001)]. O capítulo também pretende investigar a possibilidade do uso do formalismo de funções aproximadas como um subsídio para o desenvolvimento de algoritmos de aprendizado de máquina (AM).

4.2 Relações Aproximadas

Sejam $A_1 = (U_1, R_1)$ e $A_2 = (U_2, R_2)$ dois espaços aproximados. O produto de A_1 por A_2 é o espaço aproximado denotado por $A = (U, R)$, onde $U = U_1 \times U_2$ e a relação de indiscernibilidade $R \subseteq (U_1 \times U_2)^2$ é definida por $((x_1, y_1), (x_2, y_2)) \in R \Leftrightarrow (x_1, x_2) \in R_1$ e $(y_1, y_2) \in R_2$, onde $x_1, x_2 \in U_1$ e $y_1, y_2 \in U_2$. Os elementos (x_1, y_1) e (x_2, y_2) são indiscerníveis em R se, e somente se, os elementos x_1 e x_2 forem indiscerníveis em R_1 e y_1 e y_2 forem indiscerníveis em R_2. Isso implica que $[(x, y)]_R$, a classe de equivalência de R contendo (x, y), deve ser igual ao produto cartesiano de $[x]_{R_1}$ por $[y]_{R_2}$, como pode ser observado no Exemplo 8. Tem-se que R, da maneira como foi definida, é uma relação de equivalência (indiscernibilidade), pois, como mostrado em [Uchôa (1998)] e em [Nicoletti; Uchôa & Baptistini (2001)]:

é reflexiva: como R_1 e R_2 são relações de equivalência em U_1 e U_2, respectivamente, então $(u_1, u_1) \in R_1$, $\forall u_1 \in U_1$ e $(u_2, u_2) \in R_2$, $\forall u_2 \in U_2$. Pela

definição de R, entretanto, tem-se que se $(u_1, u_1) \in R_1$ e $(u_2, u_2) \in R_2 \Leftrightarrow ((u_1, u_2), (u_1, u_2)) \in R \Leftrightarrow R$ é reflexiva;

é simétrica: dado que R_1 e R_2 são relações de equivalência em U_1 e U_2, respectivamente, R_1 e R_2 são simétricas, i.e.,

- $(u_1, u_2) \in R_1 \Leftrightarrow (u_2, u_1) \in R_1$,
- $(v_1, v_2) \in R_2 \Leftrightarrow (v_2, v_1) \in R_2$,

então:
$((u_1, v_1), (u_2, v_2)) \in R \Leftrightarrow (u_1, u_2) \in R_1$ e $v_1, v_2 \in R_2 \Leftrightarrow (u_2, u_1) \in R_1$ e $v_2, v_1 \in R_2 \Leftrightarrow ((u_2, v_2), (u_1, v_1)) \in R \Leftrightarrow R$ é simétrica;

é transitiva: dado que R_1 e R_2 são relações de equivalência em U_1 e U_2, respectivamente, R_1 e R_2 são transitivas, i.e.,

- $(u_1, u_2) \in R_1$ e $(u_2, u_3) \in R_1 \Leftrightarrow (u_1, u_3) \in R_1$,
- $(v_1, v_2) \in R_2$ e $(v_2, v_3) \in R_2 \Leftrightarrow (v_1, v_3) \in R_2$,

então: $((u_1, v_1), (u_2, v_2)) \in R$ e $((u_2, v_2), (u_3, v_3)) \in R \Leftrightarrow (u_1, u_2) \in R_1$ e $(u_2, u_3) \in R_1$, $(v_1, v_2) \in R_2$ e $(v_2, v_3) \in R_2 \Leftrightarrow (u_1, u_3) \in R_1$ e $(v_1, v_3) \in R_2 \Leftrightarrow ((u_1, v_1), (u_3, v_3)) \in R \Leftrightarrow R$ é transitiva.

Exemplo 8 *Sejam $A_1 = (U_1, R_1)$ e $A_2 = (U_2, R_2)$ dois espaços aproximados, onde $U_1 = \{x_1, x_2, x_3, x_4\}$, $R_1 = \{(x_1, x_1), (x_2, x_2), (x_3, x_3), (x_4, x_4), (x_1, x_2), (x_2, x_1), (x_3, x_4), (x_4, x_3)\}$, $U_2 = \{a, b, c\}$ e $R_2 = \{(a, a), (b, b), (c, c), (a, b), (b, a)\}$. Seja $A = (U, R) = (U_1 \times U_2, R)$ o produto de A_1 por A_2, onde $R \subseteq (U_1 \times U_2)^2$. Neste caso, o conjunto de objetos U é dado por,*

$$U = \{(x_1, a), (x_1, b), (x_1, c), (x_2, a), (x_2, b), (x_2, c), (x_3, a), (x_3, b), (x_3, c), (x_4, a), (x_4, b), (x_4, c)\},$$

e como R é definida por $((x_1, y_1), (x_2, y_2)) \in R \Leftrightarrow (x_1, x_2) \in R_1$ e $(y_1, y_2) \in R_2$, tem se que

$$\begin{aligned} R = \{&((x_1, a), (x_1, a)), ((x_1, b), (x_1, b)), ((x_1, c), (x_1, c)), ((x_1, a), (x_1, b)), ((x_1, b), (x_1, a)),\\ &((x_2, a), (x_2, a)), ((x_2, b), (x_2, b)), ((x_2, c), (x_2, c)), ((x_2, a), (x_2, b)), ((x_2, b), (x_2, a)),\\ &((x_3, a), (x_3, a)), ((x_3, b), (x_3, b)), ((x_3, c), (x_3, c)), ((x_3, a), (x_3, b)), ((x_3, b), (x_3, a)),\\ &((x_4, a), (x_4, a)), ((x_4, b), (x_4, b)), ((x_4, c), (x_4, c)), ((x_4, a), (x_4, b)), ((x_4, b), (x_4, a)),\\ &((x_1, a), (x_2, a)), ((x_1, b), (x_2, b)), ((x_1, c), (x_2, c)), ((x_1, a), (x_2, b)), ((x_1, b), (x_2, a)),\\ &((x_2, a), (x_1, a)), ((x_2, b), (x_1, b)), ((x_2, c), (x_1, c)), ((x_2, a), (x_1, b)), ((x_2, b), (x_1, a)),\\ &((x_3, a), (x_4, a)), ((x_3, b), (x_4, b)), ((x_3, c), (x_4, c)), ((x_3, a), (x_4, b)), ((x_3, b), (x_4, a)),\\ &((x_4, a), (x_3, a)), ((x_4, b), (x_3, b)), ((x_4, c), (x_3, c)), ((x_4, a), (x_3, b)), ((x_4, b), (x_3, a))\}. \end{aligned}$$

Os conceitos da TCA podem ser facilmente estendidos a um relação, principalmente pelo fato que uma relação é também um conjunto. Assim, sejam $A_1 = (U_1, R_1)$ e $A_2 = (U_2, R_2)$ dois espaços aproximados e $A = (U, R) = (U_1 \times U_2, R)$ o produto de A_1 por A_2. Dada uma relação (ou um conjunto) $X \subseteq U_1 \times U_2$, podem ser definidas a aproximação inferior e a aproximação superior de X no espaço aproximado A:

$$A_{inf}(X) = \{(x, y) \in U_1 \times U_2 | [(x, y)]_R \subseteq X\},$$
$$A_{sup}(X) = \{(x, y) \in U_1 \times U_2 | [(x, y)]_R \cap X \neq \varnothing\}.$$

Exemplo 9 *Seja $A = (U, R) = (U_1 \times U_2, R)$ o espaço aproximado produto definido no Exemplo 8. Sejam também as relações $X, Y, Z \subseteq U_1 \times U_2$, tal que $X = \{(x_1, a), (x_1, b)\}$, $Y = \{(x_1, c), (x_2, c), (x_3, c), (x_4, c)\}$ e $Z = \{(x_1, a), (x_1, c), (x_3, a), (x_3, c), (x_4, c)\}$. Neste caso, tem-se:*

$$A_{inf}(X) = \varnothing,$$

$$A_{sup}(X) = [(x_1, a)]_R = [(x_1, b)]_R = \{(x_1, a), (x_1, b), (x_2, a), (x_2, b)\},$$

$$A_{inf}(Y) = A_{sup}(Y) = [(x_1, c)]_R \cup [(x_3, c)]_R = \{(x_1, c), (x_2, c), (x_3, c), (x_4, c)\} = Y,$$

$$A_{inf}(Z) = [(x_3, c)]_R = \{(x_3, c), (x_4, c)\},$$

$$A_{sup}(Z) = [(x_1, a)]_R \cup [(x_1, c)]_R \cup [(x_3, a)]_R \cup [(x_3, c)]_R = U.$$

Além das aproximações, os conceitos de regiões, bem como o de pertinência aproximada, também podem ser facilmente estendidos a relações:

$$pos_A(X) = A_{inf}(X) = \{(x, y) \in U_1 \times U_2 | [(x, y)]_R \subseteq X\},$$

$$neg_A(X) = U_1 \times U_2 - A_{sup}(X) = \{(x, y) \in U_1 \times U_2 | [(x, y)]_R \cap X = \varnothing\},$$

$$duv_A(X) = A_{sup}(X) - A_{inf}(X) = \{(x, y) \in U_1 \times U_2 | [(x, y)]_R \cap X \neq \varnothing \; | [(x, y)]_R \not\subseteq X\},$$

$$\mu_X^A((x, y)) = \frac{|[(x, y)]_R \cap X|}{|[(x, y)]_R|}.$$

Exemplo 10 *Seja S_1 o SRC fornecido pela Tabela 4.1 onde $U = \{x_1, \ldots, x_9\}$ e $Q = \{a, b, c, d\}$, cuja representação é fornecida pela Tabela 4.2. Seja S_2 o SRC fornecido pela Tabela 4.3 onde $U = \{x_1, \ldots, x_9\}$ e $Q = \{a, b, c\}$, cuja representação é fornecida pela Tabela 4.4. Considere $A_1 = (U_1, S_1)$ e $A_2 = (U_2, S_2)$ os espaços aproximados induzidos, respectivamente, por S_1 e S_2, sendo que o produto de A_1 por A_2 é dado por $A = (U, S)$. Seja também uma relação $X \subseteq U$, dada por $X = \{(x_1, x_1), (x_1, x_3), (x_1, x_7), (x_2, x_3), (x_2, x_7), (x_3, x_3), (x_6, x_7)\}$. A aproximação inferior e a aproximação superior de X em A são, portanto:*

$$A_{inf}(X) = \{(x_1, x_3), (x_1, x_7), (x_2, x_3), (x_2, x_7)\}$$

$$\begin{aligned} A_{sup}(X) = \{&(x_1, x_1), (x_1, x_2), (x_1, x_3), (x_1, x_6), (x_1, x_7), (x_2, x_1), (x_2, x_2), \\ &(x_2, x_3), (x_2, x_6), (x_2, x_7), (x_3, x_3), (x_3, x_7), (x_7, x_3), (x_7, x_7), (x_6, x_3), \\ &(x_6, x_7)\} \end{aligned}$$

Tabela 4.1: Exemplo de SRC, onde $U = \{x_1, \ldots, x_9\}$ e $Q = \{a, b, c, d\}$.

U	a	b	c	d
x_1	1	0	1	1
x_2	1	0	1	1
x_3	0	1	0	1
x_4	1	1	1	1
x_5	1	1	1	0
x_6	1	0	1	0
x_7	0	1	0	1
x_8	1	1	1	1
x_9	1	1	1	0

Tabela 4.2: Conjuntos elementares gerados a partir do SRC fornecido pela Tabela 4.1.

U/Q	a	b	c	d
$E_1 = \{x_1, x_2\}$	1	0	1	1
$E_2 = \{x_3, x_7\}$	0	1	0	1
$E_3 = \{x_4, x_8\}$	1	1	1	1
$E_4 = \{x_5, x_9\}$	1	1	1	0
$E_5 = \{x_6\}$	1	0	1	0

Tabela 4.3: SRC, onde $U = \{x_1, \ldots, x_9\}$ e $Q = \{a, b, c\}$.

U	a	b	c
x_1	1	0	1
x_2	1	0	1
x_3	0	1	0
x_4	1	1	1
x_5	1	1	1
x_6	1	0	1
x_7	0	1	0
x_8	1	1	1
x_9	1	1	1

Tabela 4.4: Conjuntos elementares gerados a partir do SRC fornecido pela Tabela 4.3.

U/Q	a	b	c
$E_1 = \{x_1, x_2, x_6\}$	1	0	1
$E_2 = \{x_3, x_7\}$	0	1	0
$E_3 = \{x_4, x_5, x_8 x_9\}$	1	1	1

Observe que as definições e conceitos relativos a um espaço aproximado produto binário $A_1 \times A_2$ podem ser facilmente estendidos a um espaço produto n-ário, da seguinte forma: seja $A_1, \ldots, A_n$ uma família de espaços aproximados onde cada $A_1 = (U_i, R_i)$, $1 \leq i \leq n$. Então $A = (U_1 \times, \ldots, \times U_n, R)$ é um espaço aproximado produto n-ário, onde R é uma relação de indiscernibilidade definida da seguinte forma: xRy, $x = (x_1, \ldots, x_n) \in U_1 \times, \ldots, \times U_n$ e $y = (y_1, \ldots, y_n) \in U_1 \times, \ldots, \times U_n \Leftrightarrow x_i R_i y_i$, para todo i, $1 \leq i \leq n$, e $x_i, y_i \in U_i$.

4.3 Funções Aproximadas

Sejam $A = (X, S)$ e $B = (Y, P)$ dois espaços aproximados quaiquer e seja uma função $f : X \to Y$. A representação de f em A é a função $f_{(A,B)} : X \to 2^Y$, tal que:

$$f_{(A,B)}(x) = \{y \in Y | z \in [x]_S, f(z) = y\},$$

ou seja, a (A, B)-aproximação de f em x é o conjunto formado por todos os elementos de y que são imagem dos elementos de X pertencentes à mesma classe

de equivalência de x. Como proposto em [Uchôa (1998)], a *aproximação inferior* e a *aproximação superior* de f em x, que são, respectivamente,as funções $f_{inf}(x) : X \to Y/P$ e $f_{sup}(x) : X \to Y/P$, definidas por:

$$f_{inf}(x) = A_{inf}(f_{(A,B)}(x)), \forall x \in X,$$

$$f_{sup}(x) = A_{sup}(f_{(A,B)}(x)), \forall x \in X.$$

Diz-se que uma função f é *exata* em $x \in X$ se e somente se $f_{inf}(x) = f_{sup}(x)$. Do contrário, f é *inexata*. O conjunto $f_{sup}(x) - f_{inf}(x)$ é o erro da aproximação de f em x, podendo ser medido por $\upsilon_f(x) = (f_{sup}(x) - f_{inf}(x))/f_{sup}(x)$. Como o interesse maior é aproximar todos os pontos de X, define-se a *aproximação inferior* e a *aproximação superior* de $f_{(A,B)}$ em X, em símbolos $A_{inf}(f) : X \to 2^Y$ e $A_{sup}(f) : X \to 2^Y$, respectivamente, da seguinte forma:

$$A_{inf}(f) = \{(x, [y]_S) \in X \times Y/P | y \in f_{inf}(x)\},$$

$$A_{sup}(f) = \{(x, [y]_S) \in X \times Y/P | y \in f_{sup}(x)\}.$$

Observe que, da forma como foram definidas, tanto $A_{inf}(f)$, como $A_{sup}(f)$ continuam sendo funções, o que pode ser observado no Exemplo 11.

Exemplo 11 *Seja o espaço aproximado $A = (U, R)$, onde $U = \{a, b, c, d, e, f, g\}$ e $U/R = \{\{a, b\}, \{c, d\}, \{e, f\}, \{g\}\}$. Seja f a função definida por $f = \{(a, a)$, $(b, a), (c, e), (d, g), (e, a), (f, b), (g, g)\}$. Para o cálculo de $A_{inf}(f)$ e $A_{sup}(f)$, é necessário obter $f_{inf}(x)$ e $f_{sup}(x)$ para todo $x \in U$, o que é fornecido pela Tabela 4.5.*

Tabela 4.5: Cálculo de $f_{inf}(x)$ e $f_{sup}(x)$.

x	$[x]_R$	$f_{(A,B)}(x)$	$f_{inf}(x)$	$f_{sup}(x)$
a	$\{a, b\}$	$\{a\}$	$\varnothing$	$\{a, b\}$
b	$\{a, b\}$	$\{a\}$	$\varnothing$	$\{a, b\}$
c	$\{c, d\}$	$\{e, g\}$	$\{g\}$	$\{e, f, g\}$
d	$\{c, d\}$	$\{e, g\}$	$\{g\}$	$\{e, f, g\}$
e	$\{e, f\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$
f	$\{e, f\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$
g	$\{g\}$	$\{g\}$	$\{g\}$	$\{g\}$

Nesse caso, a aproximação inferior e a aproximação superior de $f_{(A,B)}$ são dadas por:

$$A_{inf}(f) = \{(a, \varnothing), (b, \varnothing), (c, \{g\}), (d, \{g\}), (e, \{a, b\}), (f, \{a, b\}), (g, \{g\})\},$$

$$A_{sup}(f) = \{(a, \{a, b\}), (b, \{a, b\}), (c, \{e, f, g\}), (d, \{e, f, g\}), (e, \{a, b\}),$$
$$(f, \{a, b\}), (g, \{g\})\}.$$

Observe que $A_{inf}(f)$ e $A_{sup}(f)$ são, a bem da verdade, aproximações da função $f_{(A,B)}$ e não de f. Mas ela permite que se pense em estratégias para funções aproximadas num espaço aproximado qualquer. Uma possível extensão, por exemplo, é a definição de uma *função de classificação aproximada* de um conjunto $Z \subseteq X$ [Uchôa (1998)], baseada na função f, e nos espaços aproximados $A = (X, S)$ e $B = (Y, P)$, notada por $F_{(A,B)} : 2^X \rightarrow 2^Y$, tal que:

$$F_{(A,B)}(Z) = \bigcup_{z \in Z} f_{(A,B)}(z)$$

O significado desta função é claro: ela indica o quão bem o conjunto Z está sendo "visto" através da função f, em relação aos espaços aproximados A e B, ou seja, como Z é "classificado" por f em B. Quando os espaços A e B são conhecidos, e não há risco de confusão, utiliza-se $F(Z)$ em vez de $F_{(A,B)}(Z)$. A aproximação inferior e a aproximação superior da função de classificação aproximada de f, como proposto em [Uchôa (1998)] e notadas, respectivamente, por $F_{inf} : 2^X \rightarrow 2^Y$ e $F_{sup} : 2^X \rightarrow 2^Y$, são definidas como:

$$F_{inf}(Z) = \bigcup_{z \in Z} f_{inf}(z)$$

$$F_{sup}(Z) = \bigcup_{z \in Z} f_{sup}(z)$$

Impondo-se a condição de Z ser um conjunto elementar em A, $F_{(A,B)}(Z)$ mostra o quão bem f mapeia a classe Z no espaço aproximado B. Mais ainda, $F_{inf}(Z)$ indica a "certeza" com a qual os espaços aproximados A e B refletem $f(Z)$, pois é formada por uma união de conjuntos elementares de B, cujos elementos seguramente pertencem à $f(Z)$, se analisados através da classificação existente em B. $F_{sup}(Z)$, por sua vez, é formada pelos conjuntos elementares de B, cujos elementos possivelmente pertencem à $f(Z)$ analisados através da classificação existente em B. Isso pode ser melhor visualizado no Exemplo 12. Neste caso, (Z é um conjunto elementar em A), é intuitivo que $F_{inf}(Z) = A_{inf}(F(Z))$ e $F_{sup}(Z) = A_{sup}(F(Z))$.

Exemplo 12 *Sejam S o SRC fornecido pela Tabela 4.6 e f uma função que mapeia todo objeto de S em $\{0, 1, 2\}$ conforme fornecido pela Tabela 4.7. Obviamente f pode ser vista como uma classificação efetuada nos objetos de S.*

Tabela 4.6: Exemplo de SRC onde $U = \{x_1, \ldots, x_{10}\}$ e $Q = \{a, b\}$.

U	a	b
x_1	4	1
x_2	6	1
x_3	6	2
x_4	6	1
x_5	4	2
x_6	4	2
x_7	4	2
x_8	4	1
x_9	6	1
x_{10}	4	1

Tabela 4.7: Mapeamento dos objetos de S em $\{0, 1, 2\}$.

U	$y_i = f(x_i)$
x_1	$y_1 = 0$
x_2	$y_2 = 2$
x_3	$y_3 = 2$
x_4	$y_4 = 2$
x_5	$y_5 = 1$
x_6	$y_6 = 1$
x_7	$y_7 = 1$
x_8	$y_8 = 0$
x_9	$y_9 = 2$
x_{10}	$y_{10} = 0$

Os conjuntos elementares de S são $E_1 = \{x_1, x_8, x_{10}\}$, $E_2 = \{x_2, x_4, x_9\}$, $E_3 = \{x_3\}$ e $E_4 = \{x_5, x_6, x_7\}$. Neste caso, tem-se:

Para E_1:

$$F_{(A,B)}(E_1) = \{y_1, y_8, y_{10}\},$$
$$F_{inf}(E_1) = \{y_1, y_8, y_{10}\},$$
$$F_{sup}(E_1) = \{y_1, y_8, y_{10}\}.$$

Para E_2:

$$F_{(A,B)}(E_2) = \{y_2, y_4, y_9\},$$
$$F_{inf}(E_2) = \varnothing,$$

$$F_{sup}(E_2) = \{y_2, y_3, y_4, y_9\}.$$

Para E_3:

$$F_{(A,B)}(E_3) = \{y_3\},$$
$$F_{inf}(E_3) = \varnothing,$$
$$F_{sup}(E_3) = \{y_2, y_3, y_4, y_9\}.$$

Para E_4:

$$F_{(A,B)}(E_4) = \{y_5, y_6, y_7\},$$
$$F_{inf}(E_4) = \{y_5, y_6, y_7\},$$
$$F_{sup}(E_4) = \{y_5, y_6, y_7\}.$$

Observe que os objetos dos conjuntos elementares E_1 e E_4 foram bem classificados, de acordo com os resultados obtidos. O mesmo não pode ser dito dos objetos de E_2 e E_3 que possuem por aproximação inferior da classificação aproximada de f o conjunto vazio, indicando que esses conjuntos elementares não foram bem classificados por f, levando-se em conta os espaços aproximados existentes (um criado a partir do sistema de representação de conhecimento S e outro a partir dos valores de $f(x)$ em U). Note que isso, pode ser utilizado no desenvolvimento de algoritmos de aprendizado de máquina.

Capítulo 5

Algoritmo de Aprendizado Indutivo de Máquina Baseado em Funções Aproximadas

5.1 Introdução

No capítulo anterior foram apresentados alguns conceitos de funções aproximadas que podem ser utilizados no desenvolvimento de algoritmos de aprendizado indutivo de máquina.

O objetivo deste capítulo é apresentar o desenvolvimento de um Algoritmo de Aprendizado Indutivo de Máquina baseado em funções aproximadas.

5.2 O Algoritmo *lmurf* - Learning Machine using ROugh Functions

O algoritmo *lmurf* foi desenvolvido para demonstrar a possibilidade de desenvolvimento de algoritmos de aprendizado indutivo de máquina subsidiados por funções aproximadas. O algoritmo desenvolvido, produz uma árvore de decisão, cujas regras são obtidas percorrendo a árvore de sua raiz até um de seus nós folhas, como pode ser verificado no Exemplo 13. Ao percorrer a árvore, os nós são analisados indicando o caminho que se deve seguir, sendo que os nós não-folha possuem um ou mais valores (atributos), que compõem o antecedente de uma regra (condições de uma regra). Os nós folhas que possuem também um ou mais valores irão

compor o conseqüente da regra (conclusão de uma regra).

Exemplo 13 *Considere o SRC representado pela Tabela 5.1, onde* $U = \{x_1, x_2, x_3 \ldots, x_6\}$, *e* $Q = \{a, b, c, d\}$.

Tabela 5.1: SRC onde $U = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ e $Q = \{a, b, c, d\}$.

U	a	b	c	d
x_1	4	1	3	1
x_2	6	2	5	2
x_3	6	1	5	1
x_4	4	2	4	1
x_5	4	1	4	1
x_6	6	1	4	2

Aplicando o algoritmo lmurf *numa tabela de decisão* $S = (U, C \cup \{\delta\}, V, \rho\}$ *onde* $C = \{a, b, c\}$ *e* $\delta = d$*, tem-se:*

- *Árvore de Decisão*

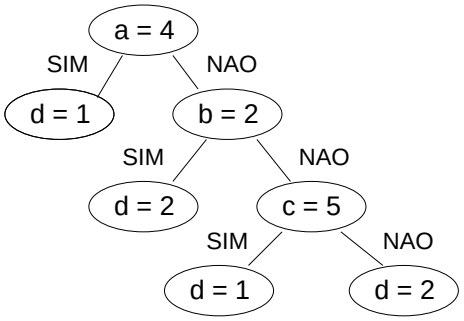


Figura 5.1: Árvore de Decisão.

- *Regras*
  (a = 4) $\Rightarrow$ (d = 1)
  (a $\neq$ 4) & (b = 2) $\Rightarrow$ (d = 2)
  (a $\neq$ 4) & (b $\neq$ 2) & (c = 5) $\Rightarrow$ (d = 1)
  (a $\neq$ 4) & (b $\neq$ 2) & (c $\neq$ 5) $\Rightarrow$ (d = 2)

O algoritmo *lmurf* utilizado no Exemplo 13 e que será descrito a seguir, foi dividido em vários passos, visando uma melhor legibilidade e compreensão. Durante os passos 5, 8, 11 e 15 do algoritmo *lmurf*, são realizadas chamadas da função *Inserir_nó*. Esta função insere nós (com proposição(ões) de condição ou proposição(ões) de decisão) na árvore de decisão. Uma descrição da função *Inserir_nó* é apresentada a seguir.

```
Inserir_nó(nó)
Início
  Se (nó_pai igual a Nulo) fazer
      nó_pai = nó;
  Senão
     Se (nó_atual.filho_esquerda = Nulo) fazer
        nó_atual.filho_esquerda = nó;
        nó.pai = nó_atual;
     Senão
        Se (nó_atual.filho_direita = Nulo) fazer
           nó_atual.filho_direita = nó;
           nó.pai = nó_atual;
        Senão
           nó_atual = nó_atual.pai;
           Inserir_nó(nó);
     Se (nó não é folha) fazer
        nó_atual = nó;
Fim
```

O algoritmo possui como entrada uma tabela de decisão $S = (U, C \cup \{\delta\}, V, \rho)$ e uma variável $pert = 0$. Com esta tabela, o algoritmo calcula $F_{inf}(X_r)$, (para todo $X_r \in Class_s(\delta)$, $1 \leq r \leq |Class_s(\delta)|$), com maior número de elementos e gera proposição(ões) consistentes para os nós da árvore de decisão. Se $F_{inf}(X_r) = \varnothing$, o algoritmo irá calcular $F_{sup}(X_r)$ (para todo $X_r \in Class_s(\delta)$, $1 \leq r \leq |Class_s(\delta)|$) e escolher $F_{sup}(X_r) \neq U$ com maior número de elementos para gerar proposição(ões) inconsistentes para os nós da árvore de decisão. Caso $F_{inf}(X_r) = \varnothing$ e $F_{sup}(X_r) = U$, o algoritmo calcula $\mu_{X_r}^{B_i}(x)$ para todo

$X_r \in Class_s(\delta)$, $1 \leq r \leq |Class_s(\delta)|$, e todo x pertencente a cada conjunto elementar de cada um dos espaços aproximados $B_i = (U, P_i)$, tal que P_i é a relação de equivalência induzida pelo atributo $c_i \in C$, $1 \leq i \leq |C|$. Se $|\mu_{X_r}^{B_i}(x)| \leq pert$ ou $|B_i| = 1$, uma proposição de conclusão é gerada e inserida em um novo nó da árvore de decisão, senão o algoritmo divide a tabela de decisão S em duas novas tabelas $S_1 = (U_1, C \cup \{\delta\}, V, \rho)$ e $S_2 = (U_2, C \cup \{\delta\}, V, \rho)$, através do maior valor obtido com o cálculo da função de pertinência aproximada $\mu_{X_r}^{B_i}(x)$. Uma proposição baseada na função de pertinência aproximada é gerada e inserida em um novo nó da árvore decisão. O algoritmo é então reiniciado recursivamente com S_1 e depois com S_2.

Dada uma tabela de decisão $S = (U, C \cup \{\delta\}, V, \rho\}$, pode-se verificar, detalhadamente, todos os passos do algoritmo *lmurf*:

1. Inicializar variáveis (fazer `pert=0`, `restart=false`, `nó_pai=NULO`, `nó_atual=nó_pai`)

2. Se U = $\varnothing$ ou `restart=true`, retornar a árvore de decisão (cujo nó raiz aponta para `nó_pai`) e terminar o algoritmo

3. Fazer `restart=true` e calcular $S = Class_s(\delta) = \{X_1, \ldots, X_n\}$, a família de conjuntos elementares do espaço aproximado $A = (U, \{\delta\})$. Se a cardinalidade de S for igual a 1 ($|S| = 1$), ir para o passo 15

4. Calcular $F_{inf}(X_r) = A_{inf}(F_{(A,B_i)}(X_r))$, para todo $X_r \in S$, $1 \leq r \leq |S|$, e todo $B_i = (U, P_i)$, tal que P_i é a relação de equivalência induzida pelo atributo $c_i \in C$, $1 \leq i \leq |C|$. $F_{(A,B_i)}(X_r)$ é a função de classificação aproximada do conjunto elementar X_r, baseada na função identidade f e no espaço aproximado $B_i = (U, P_i)$. Se $F_{inf}(X_r) = \varnothing$, para todo $1 \leq r \leq |S|$, ir para o passo 7

5. Fazer `restart=false`. Selecionar a $F_{inf}(X_r)$, $1 \leq r \leq |S|$, com maior número de elementos. Identificar os conjuntos elementares $E = \{E_1, \ldots, E_j\}$, $1 \leq j \leq |F_{inf}(X_r)|$, contidos em $F_{inf}(X_r)$. Para cada $E_k \in E$, $1 \leq k \leq j$, gerar uma proposição de condição para um nó não-folha da árvore de decisão e uma proposição de decisão para um nó folha da árvore. Cada nó da árvore tem a forma $(a = v_{ae_1}) ou \ldots ou (a = v_{ae_j})$, onde a é o atributo que gera o espaço aproximado B_i utilizado em $F_{inf}(X_r)$. Por sua vez, $e_k \in U$, $1 \leq k \leq j$, é um elemento qualquer do conjunto elementar E_k. Cada proposição de condição $(a = v_{ae_k})$ é constituída da seguinte forma: a recebe o nome do atributo que gera o espaço aproximado B_i

utilizado em $F_{inf}(X_r)$ e v_{ae_k}, $1 \leq k \leq j$, recebe o valor atribuído a cada elemento e_k por esse atributo. As proposições de decisão são constituídas de forma semelhantes, com a exceção de que a recebe o nome do atributo de decisão δ. Fazer `nó_não-folha=proposição(ões) de condição` e `nó_folha=proposição(ões) de decisão`. Executar a função `Inserir_nó(nó_não-folha)` e `Inserir_nó(nó_folha)`.
As conclusões tiradas a partir da árvore de decisão e que envolvam estes nó, são classificadas como conclusões consistentes.

6. Fazer $U = U - F_{inf}(X_r)$. Se para o novo conjunto U, o número de conjuntos elementares do espaço aproximado B_i utilizado em $F_{inf}(X_r)$ for igual a 1, remover o atributo $c_i \in C$, $1 \leq i \leq |C|$, que induziu o espaço aproximado B_i. Ir para o passo 2

7. Calcular $F_{sup}(X_r) = A_{sup}(F_{(A,B_i)}(X_r))$, para todo $X_r \in S$, $1 \leq r \leq |S|$, e todo $B_i = (U, P_i)$, tal que P_i é a relação de equivalência induzida pelo atributo $c_i \in C$, $1 \leq i \leq |C|$. $F_{(A,B_i)}(X_r)$ é a função de classificação aproximada do conjunto elementar X_r, baseada na função identidade f e no espaço aproximado $B_i = (U, P_i)$. Se $F_{sup}(X_r) = U$, para todo $1 \leq r \leq |S|$, ir para o passo 10

8. Fazer `restart=false`. Selecionar a $F_{sup}(X_r)$, $1 \leq r \leq |S|$, com maior número de elementos. Identificar os conjuntos elementares $E = \{E_1, \ldots, E_j\}$, $1 \leq j \leq |F_{sup}(X_r)|$, contidos em $F_{sup}(X_r)$. Para cada $E_k \in E$, $1 \leq k \leq j$, gerar uma proposição de condição para um nó não-folha da árvore de decisão e uma proposição de decisão para um nó folha da árvore. Cada nó da árvore tem a forma $(a = v_{ae_1}) ou \ldots ou (a = v_{ae_j})$, onde a é o atributo que gera o espaço aproximado B_i utilizado em $F_{sup}(X_r)$. Por sua vez, $e_k \in U$, $1 \leq k \leq j$, é um elemento qualquer do conjunto elementar E_k. Cada proposição de condição $(a = v_{ae_k})$ é constituída da seguinte forma: a recebe o nome do atributo que gera o espaço aproximado B_i utilizado em $F_{sup}(X_r)$ e v_{ae_k}, $1 \leq k \leq j$, recebe o valor atribuído a cada elemento e_k por esse atributo. As proposições de decisão são constituídas de forma semelhantes, com a exceção de que a recebe o nome do atributo de decisão δ. Fazer `nó_não-folha=proposição(ões) de condição`, `nó_folha=proposição(ões) de decisão`. Executar a função `Inserir_nó(nó_não-folha)` e `Inserir_nó(nó_folha)`.
As conclusões tiradas a partir da árvore de decisão e que envolvam estes nó, são classificadas como conclusões inconsistentes.

9. Fazer $U = U - F_{sup}(X_r)$. Se para o novo conjunto U, o número de conjuntos elementares do espaço aproximado B_i utilizado em $F_{sup}(X_r)$ for igual a 1, remover o atributo $c_i \in C$, $1 \leq i \leq |C|$, que induziu o espaço aproximado B_i. Ir para o passo 2

10. Calcular a pertinência aproximada $\mu_{X_r}^{B_i}(x) = \frac{|[x]_R \cap X_r|}{|[x]_R|}$, para todo $X_r \in S$, $1 \leq r \leq |S|$, e todo x pertencente a cada conjunto elementar de cada um dos espaços aproximados $B_i = (U, P_i)$, tal que P_i é a relação de equivalência induzida pelo atributo $c_i \in C$, $1 \leq i \leq |C|$. Selecionar a pertinência aproximada com o maior valor. Se a pertinência selecionada for menor ou igual a variavél `pert` ou se o número de conjuntos elementares do espaço aproximado B_i da pertinência aproximada de maior valor for igual a 1, ir para o passo 15

11. Fazer `pert` = $\mu_{X_r}^{B_i}(x)$. Para os elementos e_k, $1 \leq k \leq j$, contidos no conjunto elementar do espaço aproximado B_i da pertinência aproximada de maior valor, gerar uma proposição (condição) para um nó não-folha da árvore de decisão. A proposição de condição $(a = v_{ae_k})$ é constituída da seguinte forma: a recebe o nome do atributo que gerou o espaço aproximado B_i utilizado em $\mu_{X_r}^{B_i}(x)$ e v_{ae_k}, $1 \leq k \leq j$, recebe o valor atribuído a cada elemento e_k (do conjunto elementar do espaço aproximado B_i que gerou o maior valor para $\mu_{X_r}^{B_i}(x)$) por esse atributo. Fazer `nó_não-folha=proposição(ões) de condição`. Executar a função `Inserir_nó(nó_não-folha)`.

12. Dividir a tabela de decisão $S = (U, C \cup \{\delta\}, V, \rho)$ em duas novas tabelas: $S_1 = (U_1, C \cup \{\delta\}, V, \rho)$ e $S_2 = (U_2, C \cup \{\delta\}, V, \rho)$, onde U_1 é formado pelos elementos do conjunto elementar E_k, $1 \leq k \leq |B_i|$, do espaço aproximado $B_i = (U, P_i)$, tal que P_i é a relação de equivalência induzida pelo atributo $c_i \in C$, $1 \leq i \leq |C|$, que gerou a $\mu_{X_r}^{B_i}(x)$ de maior valor, e U_2 o conjunto formado pelos elementos que não pertence ao conjunto elementar E_k

13. Remover o atributo $c_i \in C$, $1 \leq i \leq |C|$, que induziu o espaço aproximado B_i da pertinência aproximada de maior valor, do conjunto de atributos C da tabela de decisão $S_1 = (U_1, C \cup \{\delta\}, V, \rho)$. Fazer `restart=false`. Realizar uma chamada recursiva com S_1, indo para o passo 2

14. Se com a tabela de decisão $S_2 = (U_2, C \cup \{\delta\}, V, \rho)$ o número de conjuntos elementares do espaço aproximado B_i que gerou o maior valor para $\mu_{X_r}^{B_i}(x)$ for igual a 1, remover o atributo c_i, $1 \leq i \leq |C|$, que induziu o espaço aproximado B_i, da tabela de decisão S_2. Fazer `restart=false`. Realizar uma chamada recursiva com S_2, indo para o passo 2

15. Para cada elemento $e_k \in U$, gerar uma proposição (decisão) para um nó folha da árvore de decisão. Cada nó folha da árvore tem a forma $(\delta = v_{ae_1}) ou \ldots ou (\delta = v_{ae_j})$, onde δ é o atributo de decisão e $e_k \in U$, $1 \leq k \leq j$, é um elemento qualquer do conjunto U. Cada proposição de decisão $(\delta = v_{ae_k})$ é constituída da seguinte forma: δ recebe o nome do atributo de decisão e v_{ae_k}, $1 \leq k \leq j$, recebe o valor atribuído a cada elemento

e_k por esse atributo. Fazer `nó_folha=proposição(ões) de decisão`. Executar a função `Inserir_nó(nó_folha)`. Fazer $U = \varnothing$ e ir para o passo 2

Exemplo 14 *Considere a tabela de decisão definida no Exemplo 5. Aplicando os passos do algoritmo* lmurf*, tem-se:*

1. Inicializar as variavéis pert = 0 e restart = false.

2. $U \neq \varnothing$ *e restart = false.*

3. Fazer restart = true. Calcular $Class_s(d) = \{X_1 = \{x_1, x_2, x_4, x_5, x_7, x_8\}, X_2 = \{x_3, x_6, x_9, x_{10}\}\}$ *e* $|Class_s(d)|$ *= 2.*

4. A Tabela 5.2 mostra os valores de $F_{inf}(X_r) = A_{inf}(F_{(A,B_i)}(X_r))$*, para os conjuntos elementares* X_1 *e* $X_2 \in A$ *(onde* A *é o espaço aproximado induzido pelo atributo de decisão* d*), com relação aos espaços aproximados* B_a*,* B_b *e* B_c*, induzidos pelos atributos de condição* a*,* b *e* c*.*

Tabela 5.2: Cálculo de $F_{inf}(X_r)$, para $X_1, X_2 \in A$, com relação a B_a, B_b e B_c.

	B_a	B_b	B_c
X_1	$\varnothing$	$\varnothing$	$\{x_1, x_7\}$
X_2	$\varnothing$	$\varnothing$	$\varnothing$

$F_{inf}(X_r) \neq \varnothing$.

5. Fazer restart = false, $F_{inf}(X_r) = \{x_1, x_7\}$ *e* $E_1 = \{x_1, x_7\}$.
Gerar proposição:

- *Proposição de condição: (c = 3);*
- *Proposição de decisão: (d = 1);*
- *Regra: (c = 3)* $\Rightarrow$ *(d = 1).*

6. Fazer $U = U - F_{inf}(X_r) = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\} - \{x_1, x_7\} = \{x_2, x_3, x_4, x_5, x_6, x_8, x_9, x_{10}\}$*. Com o novo conjunto* U *e com* c *(o atributo que induzido* B_c *utilizado para calcular* $F_{inf}(X_r)$*), o número de conjuntos elementares é igual a 2 (*$E_1 = \{x_3, x_4\}$ *e* $E_2 = \{x_2, x_5, x_6, x_8, x_9, x_{10}\}$*). Ir para o passo 2.*

2. $U \neq \varnothing$ *e restart = false.*

3. Fazer restart = true. Calcular $Class_s(d) = \{X_1 = \{x_2, x_4, x_5, x_8\}, X_2 = \{x_3, x_6, x_9, x_{10}\}\}$ *e* $|Class_s(d)|$ *= 2.*

4. A Tabela 5.3 mostra os valores de $F_{inf}(X_r) = A_{inf}(F_{(A,B_i)}(X_r))$*, para os conjuntos elementares* X_1 *e* $X_2 \in A$ *(onde* A *é o espaço aproximado induzido pelo atributo de decisão* d*), com relação aos espaços aproximados* B_a*,* B_b *e* B_c*, induzidos pelos atributos de condição* a*,* b *e* c*.*

Tabela 5.3: Cálculo de $F_{inf}(X_r)$, para $X_1, X_2 \in A$, com relação a B_a, B_b e B_c.

	B_a	B_b	B_c
X_1	$\varnothing$	$\varnothing$	$\varnothing$
X_2	$\varnothing$	$\varnothing$	$\varnothing$

Como $F_{inf}(X_r) = \varnothing$*, ir para o passo 7.*

7. A Tabela 5.4 mostra os valores de $F_{sup}(X_r) = A_{sup}(F_{(A,B_i)}(X_r))$*, para os conjuntos elementares* X_1 *e* $X_2 \in A$ *(onde* A *é o espaço aproximado induzido pelo atributo de decisão* d*), com relação aos espaços aproximados* B_a*,* B_b *e* B_c*, induzidos pelos atributos de condição* a*,* b *e* c*.*

Tabela 5.4: Cálculo de $F_{inf}(X_r)$, para $X_1, X_2 \in A$, com relação a B_a, B_b e B_c.

	B_a	B_b	B_c
X_1	U	U	U
X_2	U	U	U

Como $F_{sup}(X_r) = U$*, ir para o passo 10.*

10. A Tabela 5.5 mostra os valores de $\mu_{X_r}^{B_i}(x) = \frac{|[x]_R \cap X_r|}{|[x]_R|}$*, para os conjuntos elementares* X_1 *e* $X_2 \in A$ *(onde* A *é o espaço aproximado induzido pelo atributo de decisão* d*), e para todo* x *pertencente aos conjuntos elementares* $E_{(a=4)}$*,* $E_{(a=6)}$*,* $E_{(b=1)}$*,* $E_{(b=2)}$*,* $E_{(c=4)}$ *e* $E_{(c=5)}$ *(conjuntos elementares induzidos pelo atributo* a *com valor igual a* 4 *e igual a* 6*, pelo atributo* b *com valor igual a* 1 *e igual a* 2 *e pelo atributo* c *com valor igual a* 4 *e igual a* 5*), que pertencem aos espaços aproximados* B_a*,* B_b *e* B_c*, induzidos pelos atributos de condição* a*,* b *e* c*.*

Tabela 5.5: Cálculo de $\mu_{X_r}^{B_i}(x) = \frac{|[x]_R \cap X_r|}{|[x]_R|}$.

	$E_{(a=4)}$	$E_{(a=6)}$	$E_{(b=1)}$	$E_{(b=2)}$	$E_{(c=4)}$	$E_{(c=5)}$
X_1	0.5	0.5	0.6	0.33	0.5	0.5
X_2	0.5	0.5	0.4	0.66	0.5	0.5

Com $E_{(b=2)}$ e X_2, $\mu_{X_r}^{B_i}(x) = 0.66$. O valor $\mu_{X_r}^{B_i}(x) > pert$ e com o atributo b tem-se 2 conjuntos elementares $E_{(b=1)} = \{x_3, x_5, x_6\}$ e $E_{(b=2)} = \{x_2, x_4, x_8, x_9, x_{10}\}$.

11. Fazer pert = $\mu_{X_r}^{B_i}(x) = 0.66$.
Gerar proposição:

- *Proposição de condição: (b = 2).*

12. Dividir a tabela de decisão S em $S_1 = (\{x_3, x_5, x_6\}, C \cup \{\delta\}, V, \rho\}$ e $S_2 = (\{x_2, x_4, x_8, x_9, x_{10}\}, C \cup \{\delta\}, V, \rho\}$.

13. Fazer $C = C - b$, para S_1. Fazer restart = false. Ir recursivamente para o passo 2 com a tabela de decisão $S_1 = (\{x_3, x_5, x_6\}, C \cup \{\delta\}, V, \rho\}$.

14. Com a tabela de decisão $S_2 = (\{x_2, x_4, x_8, x_9, x_{10}\}, C \cup \{\delta\}, V, \rho\}$, e com o atributo b da pertinência aproximada temos apenas o conjunto elementar $E_{(b=2)} = \{x_2, x_4, x_8, x_9, x_{10}\}$. Como o número de conjuntos elementares é igual a 1, fazer $C = C - b$, para S_2. Fazer restart = false. Ir recursivamente para o passo 2 com a tabela de decisão $S_2 = (\{x_2, x_4, x_8, x_9, x_{10}\}, C \cup \{\delta\}, V, \rho\}$.

2. $U \neq \varnothing$, mas restart = true, retornar a árvore de decisão e terminar o algoritmo.

Inicio da primeira chamada recursiva

2. $U \neq \varnothing$ e restart = false.

3. Fazer restart = true. Calcular $Class_s(d) = \{X_1 = \{x_5\}, X_2 = \{x_3, x_6\}\}$ e $|Class_s(d)|$ = 2.

4. A Tabela 5.6 mostra os valores de $F_{inf}(X_r) = A_{inf}(F_{(A,B_i)}(X_r))$, para os conjuntos elementares X_1 e $X_2 \in A$ (onde A é o espaço aproximado induzido pelo atributo de decisão d), com relação aos espaços aproximados B_a e B_c, induzidos pelos atributos de condição a e c.

Tabela 5.6: Cálculo de $F_{inf}(X_r)$, para $X_1, X_2 \in A$, com relação a B_a e B_c.

	B_a	B_c
X_1	$\varnothing$	$\varnothing$
X_2	$\{x_3\}$	$\varnothing$

$F_{inf}(X_r) \neq \varnothing$.

5. Fazer restart = false, $F_{inf}(X_r) = \{x_3\}$ *e* $E_1 = \{x_3\}$.
Gerar proposição:

- *Proposição de condição: (a = 6);*
- *Proposição de decisão: (d = 2);*
- *Regra: (c* $\neq$ *3) & (b = 2) & (a = 6)* $\Rightarrow$ *(d = 2).*

6. Fazer $U = U - F_{inf}(X_r) = \{x_3, x_5, x_6\} - \{x_3\} = \{x_5, x_6\}$. *Com o novo conjunto* U *e com* a *(o atributo que induzido* B_a *utilizado para calcular* $F_{inf}(X_r)$*), o número de conjuntos elementares é igual a 1 (*$E_1 = \{x_5, x_6\}$*), o que consiste em fazer* $C = C - a$ *e ir para o passo 2.*

2. $U \neq \varnothing$ *e restart = false.*

3. Fazer restart = true. Calcular $Class_s(d) = \{X_1 = \{x_5\}, X_2 = \{x_6\}\}$ *e* $|Class_s(d)|$ *= 2.*

4. A Tabela 5.7 mostra os valores de $F_{inf}(X_r) = A_{inf}(F_{(A,B_i)}(X_r))$*, para os conjuntos elementares* X_1 *e* $X_2 \in A$ *(onde* A *é o espaço aproximado induzido pelo atributo de decisão* d*), com relação ao espaço aproximado* B_c*, induzidos pelos atributo de condição* c.

Tabela 5.7: Cálculo de $F_{inf}(X_r)$, para $X_1, X_2 \in A$, com relação a B_c.

	B_c
X_1	$\varnothing$
X_2	$\varnothing$

Como $F_{inf}(X_r) = \varnothing$*, ir para o passo 7.*

7. A Tabela 5.8 mostra os valores de $F_{sup}(X_r) = A_{sup}(F_{(A,B_i)}(X_r))$*, para os*

conjuntos elementares X_1 e $X_2 \in A$ (onde A é o espaço aproximado induzido pelo atributo de decisão d), com relação ao espaço aproximado B_c, induzidos pelos atributo de condição c.

Tabela 5.8: Cálculo de $F_{sup}(X_r)$, para $X_1, X_2 \in A$, com relação a B_c.

	B_c
X_1	U
X_2	U

Como $F_{sup}(X_r) = U$, ir para o passo 10.

10. A Tabela 5.9 mostra os valores de $\mu_{X_r}^{B_i}(x) = \frac{|[x]_R \cap X_r|}{|[x]_R|}$, para os conjuntos elementares X_1 e $X_2 \in A$ (onde A é o espaço aproximado induzido pelo atributo de decisão d), e para todo x pertencente ao conjunto elementar $E_{(c=4)}$ (conjunto elementar induzido pelo atributo c com valor igual a 4), que pertence ao espaço aproximado B_c, induzido pelo atributo de condição c.

Tabela 5.9: Cálculo de $\mu_{X_r}^{B_i}(x) = \frac{|[x]_R \cap X_r|}{|[x]_R|}$.

	$E_{(c=4)}$
X_1	0.5
X_2	0.5

Com $E_{(c=4)}$ e X_1, $\mu_{X_r}^{B_i}(x) = 0.5$. O valor $\mu_{X_r}^{B_i}(x) < pert$. Ir para o passo 15.

15. Gerar uma proposição com o atributo de decisão d:

- *Proposição de decisão: (d = 1 ou d = 2);*
- *Regra: (c ≠ 3) & (b = 2) & (a ≠ 6) ⇒ (d = 1 ou d = 2).*

Fazer $U = \varnothing$ e ir para o passo 2.

2. $U = \varnothing$, retornar uma árvore de decisão parcial e voltar ao passo 14, seguinte ao passo 13 da primeira chamada recursiva.

Inicio da segunda chamada recursiva

2. $U \neq \varnothing$ e restart = false.

3. Fazer restart = true. Calcular $Class_s(d) = \{X_1 = \{x_2, x_4, x_8\}, X_2 = \{x_9, x_{10}\}\}$

e $|Class_s(d)|$ = 2.

4. A Tabela 5.10 mostra os valores de $F_{inf}(X_r) = A_{inf}(F_{(A,B_i)}(X_r))$, para os conjuntos elementares X_1 e $X_2 \in A$ (onde A é o espaço aproximado induzido pelo atributo de decisão d), com relação aos espaços aproximados B_a e B_c, induzidos pelos atributos de condição a e c.

Tabela 5.10: Cálculo de $F_{inf}(X_r)$, para $X_1, X_2 \in A$, com relação a B_a e B_c.

	B_a	B_c
X_1	$\varnothing$	$\{x_4\}$
X_2	$\varnothing$	$\varnothing$

$F_{inf}(X_r) \neq \varnothing$.

5. Fazer restart = false, $F_{inf}(X_r) = \{x_4\}$ e $E_1 = \{x_4\}$.
Gerar proposição:

- Proposição de condição: (c = 5);
- Proposição de decisão: (d = 1);
- Regra: (c $\neq$ 3) & (b $\neq$ 2) & (c = 5) $\Rightarrow$ (d = 1).

6. Fazer $U = U - F_{inf}(X_r) = \{x_2, x_4, x_8, x_9, x_{10}\} - \{x_4\} = \{x_2, x_8, x_9, x_{10}\}$. Com o novo conjunto U e com c (o atributo que induzido B_c utilizado para calcular $F_{inf}(X_r)$), o número de conjuntos elementares é igual a 1 ($E_1 = \{x_2, x_8, x_9, x_{10}\}$), o que consiste em fazer $C = C - c$ e ir para o passo 2.

2. $U \neq \varnothing$ e restart = false.

3. Fazer restart = true. Calcular $Class_s(d) = \{X_1 = \{x_2, x_8\}, X_2 = \{x_9, x_{10}\}\}$ e $|Class_s(d)|$ = 2.

4. A Tabela 5.11 mostra os valores de $F_{inf}(X_r) = A_{inf}(F_{(A,B_i)}(X_r))$, para os conjuntos elementares X_1 e $X_2 \in A$ (onde A é o espaço aproximado induzido pelo atributo de decisão d), com relação ao espaço aproximado B_a, induzido pelo atributo de condição a.

Tabela 5.11: Cálculo de $F_{inf}(X_r)$, para $X_1, X_2 \in A$, com relação a B_a.

	a
X_1	$\varnothing$
X_2	$\varnothing$

Como $F_{inf}(X_r) = \varnothing$, ir para o passo 7.

7. A Tabela 5.12 mostra os valores de $F_{sup}(X_r) = A_{sup}(F_{(A,B_i)}(X_r))$, para os conjuntos elementares X_1 e $X_2 \in A$ (onde A é o espaço aproximado induzido pelo atributo de decisão d), com relação ao espaço aproximado B_a, induzido pelo atributo de condição a.

Tabela 5.12: Cálculo de $F_{sup}(X_r)$, para $X_1, X_2 \in A$, com relação a B_a.

	a
X_1	U
X_2	U

Como $F_{sup}(X_r) = U$, ir para o passo 10.

10. A Tabela 5.13 mostra os valores de $\mu_{X_r}^{B_i}(x) = \frac{|[x]_R \cap X_r|}{|[x]_R|}$, para os conjuntos elementares X_1 e $X_2 \in A$ (onde A é o espaço aproximado induzido pelo atributo de decisão d), e para todo x pertencente aos conjuntos elementares $E_{(a=4)}$ e $E_{(a=6)}$ (conjuntos elementares induzidos pelo atributo a com valor igual a 4 e igual a 6), que pertencem ao espaço aproximado B_a, induzido pelo atributo de condição a.

Tabela 5.13: Cálculo de $\mu_{X_r}^{B_i}(x) = \frac{|[x]_R \cap X_r|}{|[x]_R|}$.

	$E_{(a=4)}$	$E_{(a=6)}$
X_1	0.5	0.5
X_2	0.5	0.5

Com $E_{(a=4)}$ e X_1, $\mu_{X_r}^{B_i}(x) = 0.5$. O valor $\mu_{X_r}^{B_i}(x) < pert$. Ir para o passo 15.

15. Gerar uma proposição com o atributo de decisão d:

- *Proposição de decisão: (d = 1 ou d = 2);*

- *Regra: (c ≠ 3) & (b ≠ 2) & (c ≠ 5) ⇒ (d = 1 ou d = 2).*

Fazer $U = \varnothing$ e ir para o passo 2.

2. $U = \varnothing$, retornar uma árvore de decisão parcial e retornar ao passo seguinte à segunda chamada recursiva.

Resultado Final do Algoritmo:

- Árvore de Decisão

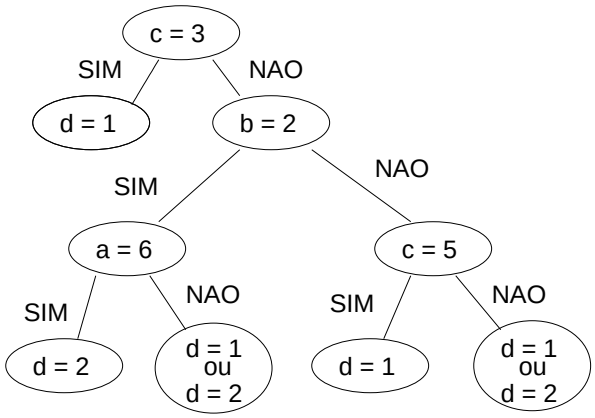


Figura 5.2: Árvore de Decisão para a Tabela de Decisão do Exemplo 5.

- Regras
  (c = 3) ⇒ (d = 1)
  (c ≠ 3) & (b = 2) & (a = 6) ⇒ (d = 1)
  (c ≠ 3) & (b = 2) & (a ≠ 6) ⇒ (d = 1 ou d = 2)
  (c ≠ 3) & (b ≠ 2) & (c = 5) ⇒ (d = 1)
  (c ≠ 3) & (b ≠2) & (c ≠ 5) ⇒ (d = 1 ou d = 2)

5.3 Testes Realizados

Após o desenvolvimento do algoritmo, foram realizados vários testes para comprovar a sua funcionalidade. Para isso, utilizou-se de uma série de arquivos de

dados fictícios. Três dos arquivos utilizados são descritos a seguir com suas respectivas árvores de decisão e conjunto de regras, obtidos através do percorrimento da árvore de seu nó raiz até os seus nós folhas. Um quarto arquivo com uma base maior de dados é apresentado em Teste 4.

5.3.1 Teste 1

Seja o SRC representado pela Tabela 5.14, onde $U = \{x_1, x_2, \ldots, x_{21}\}$, e $Q = \{a, b, c, d\}$. Aplicando o algoritmo *lmurf* numa tabela de decisão $S = (U, C \cup \{\delta\}, V, \rho\}$ onde $C = \{a, b, c\}$ e $\delta = d$, tem-se:

Tabela 5.14: SRC onde $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, \ldots, x_{20}, x_{21}\}$ e $Q = \{a, b, c, d\}$.

U	a	b	c	d
x_1	6	3	3	1
x_2	6	3	3	2
x_3	6	3	3	2
x_4	4	3	4	2
x_5	6	3	3	2
x_6	6	2	5	1
x_7	6	3	5	2
x_8	4	2	4	2
x_9	4	2	3	2
x_{10}	4	2	3	1
x_{11}	4	3	4	2
x_{12}	4	3	4	2
x_{13}	4	2	3	2
x_{14}	4	3	4	2
x_{15}	4	2	4	2
x_{16}	4	3	3	2
x_{17}	6	3	3	1
x_{18}	4	3	3	1
x_{19}	4	3	4	2
x_{20}	4	3	4	2
x_{21}	4	2	3	2

- *Árvore de Decisão*

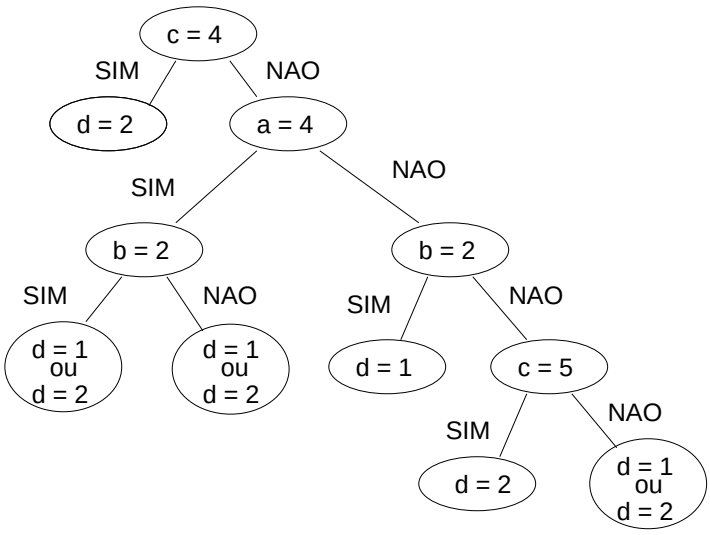


Figura 5.3: Árvore de Decisão do Teste 1.

- *Regras*
  (c = 4) $\Rightarrow$ (d = 2)
  (c $\neq$ 4) & (a = 4) & (b = 2) $\Rightarrow$ (d = 2 ou d = 1)
  (c $\neq$ 4) & (a = 4) & (b $\neq$ 2) $\Rightarrow$ (d = 2 ou d = 1)
  (c $\neq$ 4) & (a $\neq$ 4) & (b = 2) $\Rightarrow$ (d = 1)
  (c $\neq$ 4) & (a $\neq$ 4) & (b $\neq$ 2) & (c = 5) $\Rightarrow$ (d = 2)
  (c $\neq$ 4) & (a $\neq$ 4) & (b $\neq$ 2) & (c $\neq$ 5) $\Rightarrow$ (d = 1 ou d = 2)

5.3.2 Teste 2

Seja o SRC representado pela Tabela 5.15, onde $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$, e $Q = \{a, b, c, d, e\}$. Aplicando o algoritmo *lmurf* numa tabela de decisão $S = (U, C \cup \{\delta\}, V, \rho\}$ onde $C = \{a, b, c, d\}$ e $\delta = e$, tem-se:

Tabela 5.15: SRC onde $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ e $Q = \{a, b, c, d, e\}$.

U	a	b	c	d	e
x_1	1	0	0	1	1
x_2	1	0	0	0	1
x_3	0	0	0	0	0
x_4	1	1	0	1	0
x_5	1	1	0	2	2
x_6	2	2	0	2	2
x_7	2	2	2	2	2

- *Árvore de Decisão*

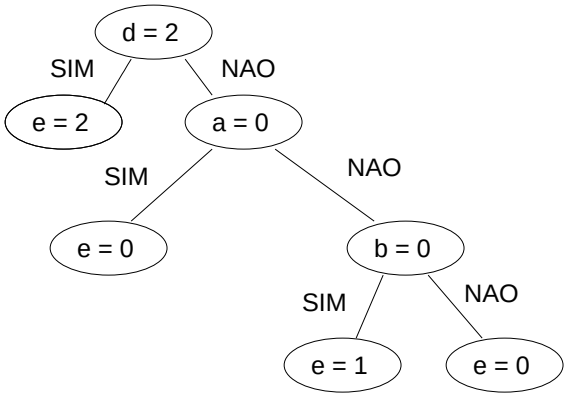


Figura 5.4: Árvore de Decisão do Teste 2.

- *Regras*
  (d = 2) $\Rightarrow$ (e = 2)
  (d $\neq$ 2) & (a = 0) $\Rightarrow$ (e = 0)
  (d $\neq$ 2) & (a $\neq$ 0) & (b = 0) $\Rightarrow$ (e = 1)
  (d $\neq$ 2) & (a $\neq$ 0) & (b $\neq$ 0) $\Rightarrow$ (e = 0)

5.3.3 Teste 3

Considere o SRC representado pela Tabela 5.16, onde $U = \{x_1, x_2, \ldots, x_{10}\}$, e $Q = \{a, b, c, d, e, f\}$. Aplicando o algoritmo *lmurf* numa tabela de decisão $S = (U, C \cup \{\delta\}, V, \rho\}$ onde $C = \{a, c, e\}$ e $\delta = f$, tem-se:

Tabela 5.16: SRC onde $U = \{x_1, x_2, \ldots, x_{10}\}$ e $Q = \{a, b, c, d, e, f\}$.

U	a	b	c	d	e	f
x_1	1	3	1	1	1	1
x_2	0	4	1	1	1	1
x_3	2	4	0	2	2	2
x_4	1	3	1	3	2	1
x_5	2	4	1	3	1	3
x_6	1	3	1	3	1	1
x_7	2	4	0	4	0	5
x_8	1	3	1	4	0	6
x_9	1	3	1	1	1	1
x_{10}	0	3	1	1	1	1

- *Árvore de Decisão*

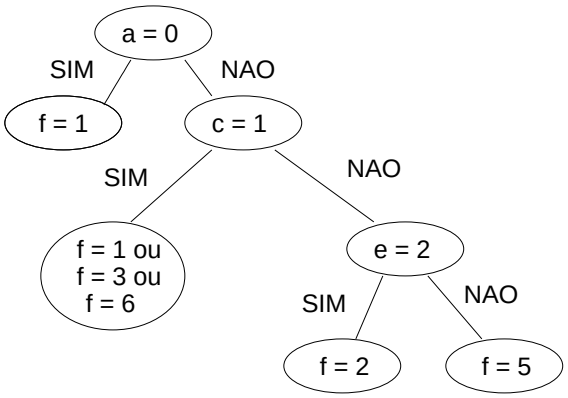


Figura 5.5: Árvore de Decisão do Teste 3.

- *Regras*
  (a = 0) $\Rightarrow$ (f = 1)
  (a $\neq$ 0) & (c = 1) $\Rightarrow$ (f = 1 ou f = 3 ou f = 6)
  (a $\neq$ 0) & (c $\neq$ 1) & (e = 2) $\Rightarrow$ (f = 2)
  (a $\neq$ 0) & (c $\neq$ 1) & (e $\neq$ 2) $\Rightarrow$ (f = 5)

Após analisar as árvores de decisão de cada um destes três arquivos de teste com suas tabelas de decisão, foi possível verificar a funcionalidade do algoritmo quanto a classificação dos elementos.

5.3.4 Teste 4

O arquivo *Car*: consiste de um conjunto de dados descrevendo a aceitação de vários modelos de carros. A descrição de cada instância é feita através de 6 atributos discretos (buying, maint, doors, persons, lug_boot, safety, acceptability), 4 com valores nominais e 2 com valores numéricos. O conjunto original é formado por 1728 elementos, sendo que não há nenhum elemento com valor ausente.

Para o domínio (arquivo) *Car*, foram realizados cinco testes e, em cada um deles, o seguinte procedimento foi seguido: o conjunto original de dados foi dividido aleatóriamente em dois, contendo respectivamente, 75% e 25% do total das instâncias, sendo o primeiro usado para indução do conceito (conjunto de treinamento) e o segundo para avaliação do conceito gerado (conjunto de teste). Esta divisão aleatória pode excluir elementos significativos a uma classificação do conjunto de teste, o que pode ser observado no Teste 1. Por grau de suporte de uma árvore de decisão está sendo considerado a porcentagem dos elementos que satisfazem a árvore como um todo. Os cinco testes são apresentados a seguir:

- **Teste 1**

  1. **Total de exemplos:** 432
  2. **Atributo(s) de Condição(ões):** maint,doors,persons,lug_boot,safety, acceptability
  3. **Atributo de Decisão:** buying
  4. **Número de regras geradas:** 10
  5. **Número de nós gerados:** 19
  6. **Número de níveis da árvore de decisão:** 7
  7. **Total de exemplos classificados pela árvore de decisão:** 0

8. **Total de exemplos não classificados pela árvore de decisão:** 432
9. **Grau de suporte:** 0.00 %

- **Teste 2**

1. **Total de exemplos:** 432
2. **Atributo(s) de Condição(ões):** buying,maint,doors,persons
3. **Atributo de Decisão:** safety
4. **Número de regras geradas:** 4
5. **Número de nós gerados:** 7
6. **Número de níveis da árvore de decisão:** 4
7. **Total de exemplos classificados pela árvore de decisão:** 177
8. **Total de exemplos não classificados pela árvore de decisão:** 255
9. **Grau de suporte:** 40.97 %

- **Teste 3**

1. **Total de exemplos:** 432
2. **Atributo(s) de Condição(ões):** buying,maint,doors,lug_boot,safety, acceptability
3. **Atributo de Decisão:** persons
4. **Número de regras geradas:** 4
5. **Número de nós gerados:** 7
6. **Número de níveis da árvore de decisão:** 4
7. **Total de exemplos classificados pela árvore de decisão:** 432
8. **Total de exemplos não classificados pela árvore de decisão:** 0
9. **Grau de suporte:** 100.00 %

- **Teste 4**

1. **Total de exemplos:** 432
2. **Atributo(s) de Condição(ões):** buying,maint,doors,persons,safety, acceptability
3. **Atributo de Decisão:** lug_boot

4. **Número de regras geradas:** 4
5. **Número de nós gerados:** 7
6. **Número de níveis da árvore de decisão:** 4
7. **Total de exemplos classificados pela árvore de decisão:** 432
8. **Total de exemplos não classificados pela árvore de decisão:** 0
9. **Grau de suporte:** 100.00 %

- **Teste 5**

1. **Total de exemplos:** 432
2. **Atributo(s) de Condição(ões):** maint, lug_boot, acceptability
3. **Atributo de Decisão:** doors
4. **Número de regras geradas:** 4
5. **Número de nós gerados:** 7
6. **Número de níveis da árvore de decisão:** 4
7. **Total de exemplos classificados pela árvore de decisão:** 427
8. **Total de exemplos não classificados pela árvore de decisão:** 5
9. **Grau de suporte:** 98.84 %

Os cinco testes realizados geraram árvores de decisões pequenas (que possuem poucos nós). Entretanto cada nó apresentou uma grande quantidade de proposições, o que possibilita a ocorrência de várias situações durante a busca de uma conclusão (decisão).

5.4 Considerações Finais

Neste capítulo demonstrou-se a possibilidade do uso dos conceitos de funções aproximadas no desenvolvimento de técnicas de aprendizado indutivo de máquina. O algoritmo desenvolvido e apresentado neste capítulo foi dividido em passos e seguido de um exemplo de sua "execução", visando tornar mais fácil o seu entendimento. Além disso, quatro outros exemplos foram apresentados buscando comprovar a funcionalidade do algoritmo.

Capítulo 6

Uma shell de um sistema especialista para o algoritmo *lmurf*

6.1 Introdução

Um sistema especialista pode ser definido como um SRC que resolve problemas bem específicos do mundo real, problemas esses que requerem consideravél habilidade, conhecimento e heurísticas para sua resolução.

Este capítulo tem como objetivo apresentar o *ESURF (Expert System using ROugh Functions)*, uma shell desenvolvida para um sistema especialista, cuja base de conhecimento é construída pelo algoritmo de aprendizado indutivo de máquina *lmurf*. A shell desenvolvida foi integrada ao protótipo *ILROS (Inductive Learning using ROugh Sets)*, um sistema apresentado em [Uchôa & Nicoletti (1998)] e em [Uchôa (1998)]. O ILROS foi aperfeiçoado, tendo suas modificações apresentadas em [Domingues & Uchôa (2001)]. O protótipo implementa os principais conceitos da TCA e alguns algoritmos de aprendizado indutivo de máquina. O ILROS e o ESURF serão apresentados nas seções seguintes.

6.2 *ILROS - Inductive Learning using ROugh Sets*

Os estudos sobre a TCA, realizados anteriormente aos apresentados nesta monografia, culmiram no desenvolvimento do ILROS (*Inductive Learning with ROugh*

Sets), um protótipo que apresenta a implementação dos principais conceitos da TCA e de alguns algoritmos de aprendizado indutivo de máquina, como: o RS1+ (apresentado em [Uchôa (1998)]), o ID3 (em [Quinlan et alii(1986)]) e o *lmurf* (apresentado no Capítulo 5). Para o protótipo optou-se por manter tanto o nome do sistema, quanto os nomes utilizados pelo sistema, em inglês, para facilitar a sua divulgação.

Nas seções seguintes serão descritos sucintamente o formato do arquivo de dados utilizado, o procedimento de execução do sistema, os procedimentos de operação de arquivo e as operações da TCA disponibilizadas no protótipo. Informações complementares sobre o funcionamento do ILROS, podem ser encontradas em [Uchôa & Nicoletti (1998)] e em [Domingues & Uchôa (2001)].

6.2.1 Descrição dos Arquivos de Dados

O arquivo de dados consiste de um arquivo texto, que tem extensão *.dat* por *default* (podendo ser redefinida pelo usuário), contendo as seguintes informações por linha (na ordem em que são apresentadas):

- nome do SRC (linha de comentário);
- número de elementos do SRC;
- número de atributos do SRC;
- uma linha com o nome dos atributos, separados por vírgula;
- um exemplo de treinamento por linha, descrito como um vetor de valores, separados por vírgulas.

Um exemplo de um arquivo de dados é o `pawlak.dat`, descrito abaixo:

```
Pawlak Example
7
5
a,b,c,d,e
1,0,0,1,1
1,0,0,0,1
0,0,0,0,0
1,1,0,1,0
1,1,0,2,2
```

2,2,0,2,2
2,2,2,2,2

6.2.2 Execução do Protótipo

O sistema protótipo ILROS é invocado via a execução do arquivo `ilros.exe`, quando compilado em ambiente *Windows* de 32 bits (95/98 ou NT) ou `./ilros`, quando compilado em ambiente *Linux*, e sua tela inicial é mostrada na Figura 6.1.

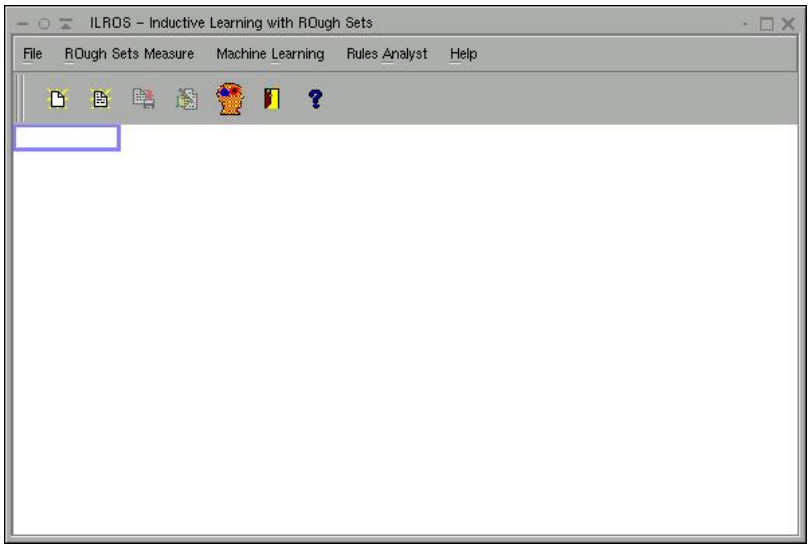


Figura 6.1: Tela inicial do protótipo.

O protótipo apresenta uma barra horizontal e uma barra vertical de ferramentas que podem ser visualizadas na Figura 6.2, e que serão descritas a seguir. Todas essas ferramentas também estão disponibilizadas pelo sistema através de *menus*.

6.2.3 Botões da Barra Horizontal de Ferramentas

A barra horizontal apresenta ferramentas que permitem o usuário manipular os arquivos de dados. Uma descrição dos botões que fazem parte da barra horizontal de ferramentas é apresentada a seguir:

- *New* - utilizado quando o usuário quer definir um novo arquivo de dados;
- *Open* - utilizado para abrir um arquivo de dados já existente;
- *Save* - utilizado para salvar um arquivo em disco;
- *Redefine* - utilizado para redefinir características do SRC (seu nome, número de atributos, número de exemplos e nome dos atributos);
- *Run Esurf* - utilizado para executar a shell do sistema especialista *ESURF*;
- *Close* - utilizado para sair do sistema;
- *About* - utilizado para apresentar alguns informações sobre o sistema;

É importante observar que os botões *Save* e *Redefine* se encontram ativos apenas quando houver um arquivo de dados disponível (carregado em memória). Quando essa situação ocorre, uma segunda barra de ferramentas (desta vez, disposta verticalmente à direita na tela) torna-se visível, disponibilizando uma série de botões que implementam operações da TCA. Essas operações serão descritas a seguir.

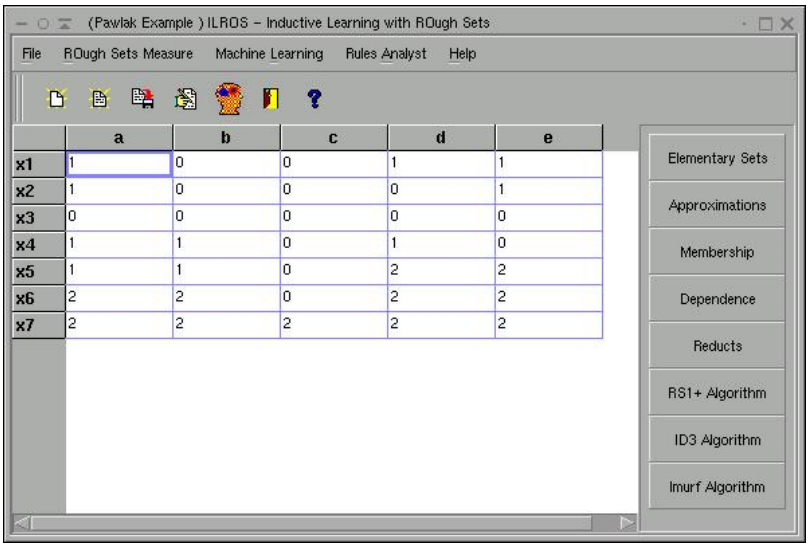


Figura 6.2: Protótipo com arquivo *pawlak.dat* carregado e visualização da barra vertical de ferramenta.

6.2.4 Botões da Barra Vertical de Ferramentas

Como mostrado na Figura 6.2, os botões disponibilizados pela barra vertical de ferramentas são: *ElementarySets, Approximations, Membership, Dependence, Reducts, RS1+ Algorithm, ID3 Algorithm* e *lmurf Algorithm*. Segue uma descrição suscinta da funcionalidade de cada um dos botões.

Botão *Elementary Sets*

O botão *Elementary Sets*, como o nome já indica, calcula os conjuntos elementares de um SRC. Antes de realizar os cálculos dos conjuntos elementares, o usuário é obrigado a escolher quais atributos devem induzir os resultados. Os resultados são mostrados na tela *Elementary Sets*, Figura 6.3.

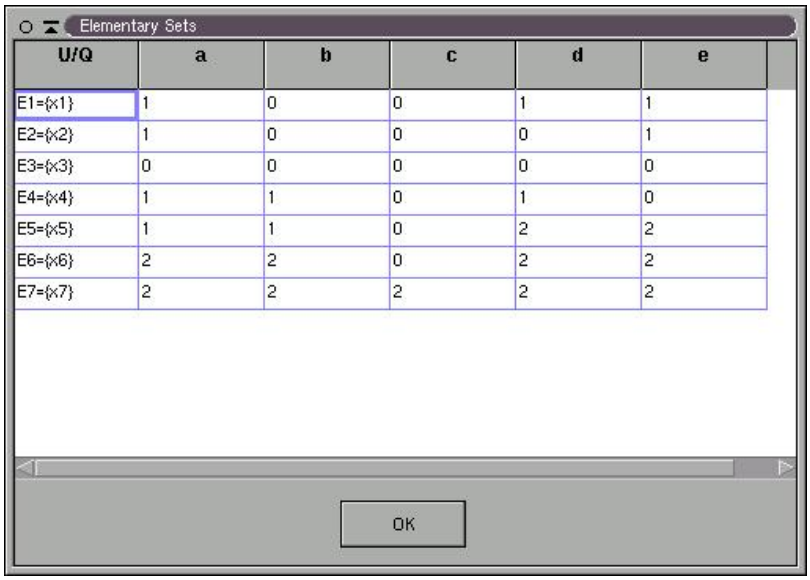


Figura 6.3: Tela com conjuntos elementares de um SRC.

Botão *Approximations*

O botão *Approximations* calcula as aproximações de um conjunto de elementos. O resultado desse procedimento é mostrado na tela *Show Results*, Figura 6.4, utilizada para apresentar resultados de outros botões. Nessa tela são retornadas a apro-

ximação inferior, a aproximação superior, a acuracidade, o indíce discriminante, e medidas de avaliação das aproximações, a saber: qualidade da aproximação inferior e qualidade da aproximação superior. Antes de realizar os cálculos, o usuário é questionado a realizar os cálculos induzido por todos os atributos ou se ele deseja indicar quais atributos deverão fazer a indução.

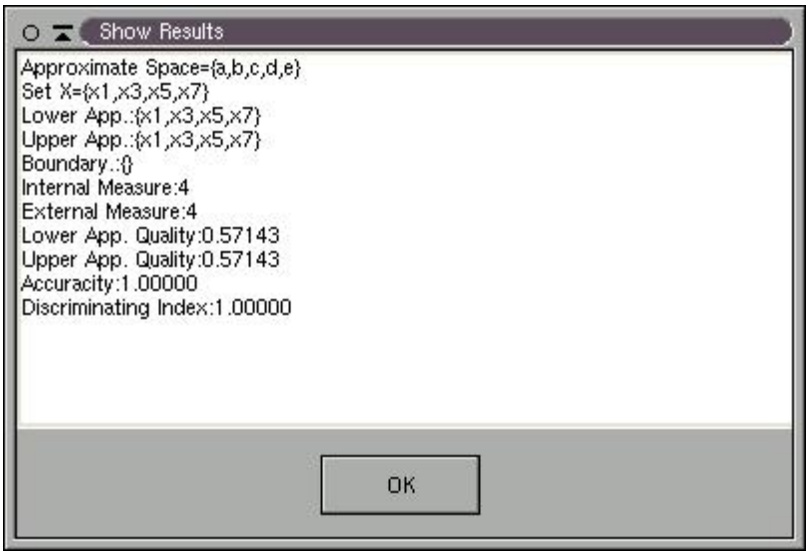


Figura 6.4: Tela que apresenta resultados de diversos procedimentos.

Botão *Membership*

O botão *Membership* calcula a pertinência aproximada de todos os elementos do SRC a um conjunto. O resultado também é mostrado na tela *Show Results*. Aqui também o usuário é questionado a indicar quais atributos farão parte do espaço aproximado.

Botão *Dependence*

O botão *Dependence* permite verificar se um conjunto de atributos é dependente ou não. Para isso, o usuário seleciona um conjunto de atributos para o cálculo. Após a escolha, o sistema notifica o usuário, através de mensagem padrão do sistema operacional, se o conjunto é dependente ou não.

Botão *Reducts*

O botão *Reducts* permite calcular todos os redutos de um dado conjunto de atributos. O resultado é apresentado na tela *Show Results*. Como exemplo, considerando-se o arquivo `pawlak.dat` carregado, tem-se os seguintes resultado apresentado na Figura 6.5 ao acionar o botão *Reducts* para o cálculo dos redutos do conjunto {a,b,c,d,e}:

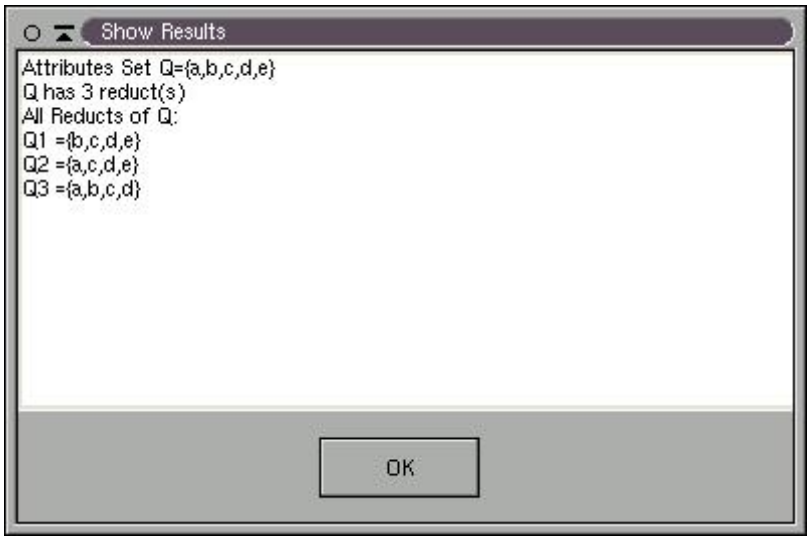


Figura 6.5: Tela apresentando o resultado do cálculo dos redutos do conjunto {a,b,c,d,e}.

Botão *RS1+ Algorithm*

O botão *RS1+ Algorithm* desencadeia o processo de geração de regras pelo algoritmo RS1+. Para isso o usuário necessita selecionar o atributo de decisão bem como o conjunto de condições, o que é feito nas telas *Select Decision Attribute* (Figura 6.6) e *Define Condiction Set* (Figura 6.7), respectivamente.

Antes, porém, que apareça a tela de seleção de atributos do conjunto de condições, o usuário é questionado se quer ou não calcular os redutos do conjunto de condições. Esse cálculo é feito levando-se em conta a relação de dependência existente entre o atributo de decisão e o conjunto de condições. É importante, em muitas situações, que o usuário possa ter uma idéia dos atributos que são efetivamente relevantes, considerando-se a relação existente entre o conjunto de condi-

ções e o atributo de decisão. A determinação dos redutos permite essa avaliação. Entretanto, o custo computacional desse cálculo (memória e tempo) é muito alto.

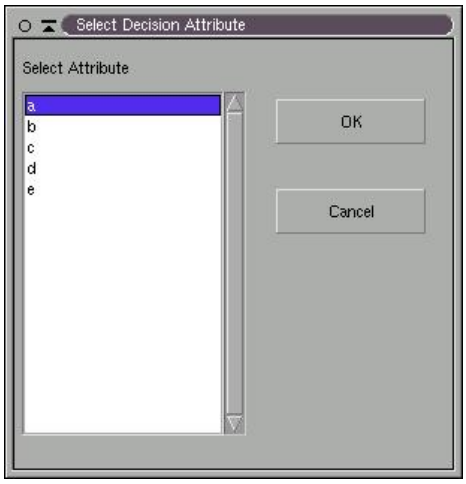


Figura 6.6: Tela de seleção do atributo de decisão.

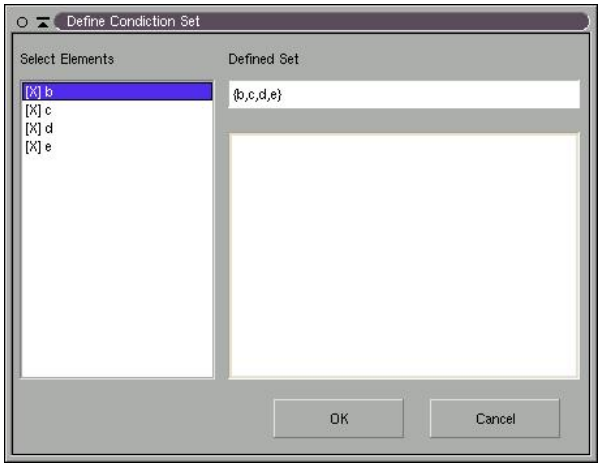


Figura 6.7: Tela de seleção do conjunto de condições.

Após a seleção do atributo de decisão e do conjunto de condições, o usuário

obtém os resultados na tela *Show Rules*, apresentada na Figura 6.8. Note que essa tela e bastante semelhante à tela *Show Result*, apresentada na Figura 6.4. A única diferença é a presença do botão *Test* em *Show Rules*. Ao acionar esse botão o usuário seleciona um SRC para teste das regras, o que implica, por parte do usuário, a seleção de um arquivo de teste. Após isso, as regras são validadas com os exemplos desse arquivo de teste, e o grau de suporte de cada uma é informado ao usuário na tela *Show Results*.

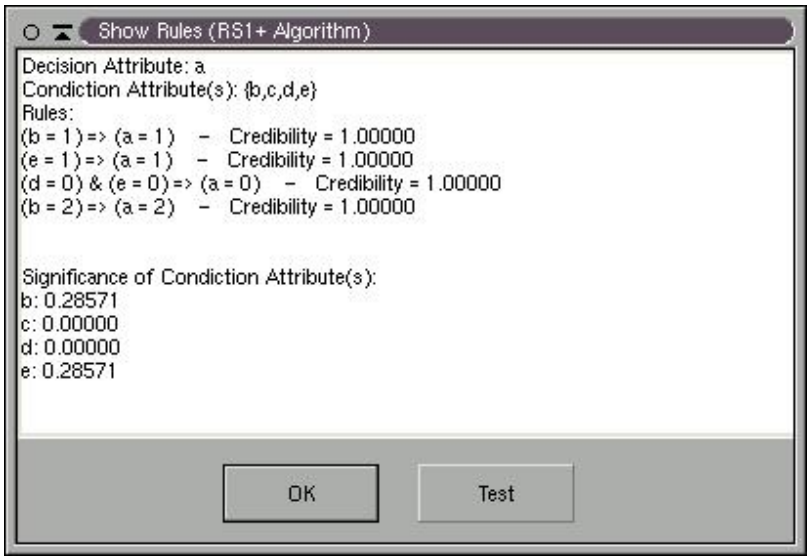


Figura 6.8: Tela que apresenta as regras geradas pelo algoritmo RS1+.

Botão *ID3 Algorithm*

O botão *ID3 Algorithm* desencadeia o processo de geração de regras pelo algoritmo ID3. Para isso o usuário necessita selecionar o atributo de decisão, o que é feito na tela *Select Decision Attribute* (Figura 6.6). As regras geradas são apresentas na tela *Show Results*.

Botão *lmurf Algorithm*

O botão *lmurf Algorithm* desencadeia o processo de geração de regras pelo algoritmo lmurf. Para isso o usuário necessita selecionar o atributo de decisão bem como o conjunto de condições, o que é feito de maneira semelhante ao demonstrado pelo botão *RS1+ Algorithm*. Após a seleção do atributo de decisão e do conjunto de condições, o usuário obtém os resultados numa tela *Show Rules* semelhante a tela *Show Rules* gerada pelo botão *RS1+ Algorithm*, com a exceção de que esta tela apresenta um botão *Save* ao invés do botão *Test*. Esse botão pode ser utilizado pelo usuário para salvar as regras geradas pelo algoritmo *lmurf*, e que serão utilizadas pelo sistema especialista *ESURF*.

6.3 *ESURF - Expert System using ROugh Functions*

O *ESURF* é uma simples *shell* de um sistema especialista que utiliza as regras geradas pelo algoritmo *lmurf*. Para a *shell* optou-se também por manter tanto o nome do sistema, quanto os nomes utilizados pelo sistema, em inglês, para seguir o padrão adotado pelo protótipo ILROS.

Como apresentado nas seções referentes ao protótipo ILROS, as seções seguintes descreverão o formato dos arquivos de dados utilizados, o procedimento de execução do sistema, os procedimentos de operação de arquivos e as operações de análise das regras.

6.3.1 Descrição dos Arquivos de Dados

Arquivo de Armazenamento de Regras

O arquivo de Armazenamento de Regras consiste de um arquivo texto, que tem extensão *.tree* por *default* (podendo ser redefinida pelo usuário), contendo as seguintes informações por linha (na ordem em que são apresentadas):

- descrição sobre qual informação (atributo) o sistema gerará uma conclusão (linha de comentário gerada automáticamente pelo sistema);
- nome do atributo da conclusão;
- número de atributos (condições) necessárias para a análise das regras e obtenção de uma conclusão;

- uma linha com o nome dos atributos (condição), separados por ponto e vírgula;
- número de sub-árvores que compõem a árvore de decisão (que gera as regras necessárias para a análise);
- um exemplo de sub-árvore por linha, formado por três nós separados por ponto e vírgula. O primeiro valor representa um nó raiz, o segundo valor representa o filho da esquerda do nó raiz e o terceiro representa o filho da direita do nó raiz. Cada nó pode apresentar mais do que um valor, sendo estes separados por vírgulas.

Um exemplo de um arquivo de dados é o `pawlak.tree`, descrito abaixo:

```
This file decide about a
a
4
b;c;d;e
3
b=1;a=1;e=1
e=1;a=1;b=2
b=2;a=2;a=0
```

Arquivo de Entrada de Dados

O arquivo de Entrada de Dados consiste de um arquivo texto, que tem extensão *.in* por *default* (podendo ser redefinida pelo usuário), contendo as seguintes informações por linha (na ordem em que são apresentadas):

- descrição do arquivo (linha de comentário gerada automáticamente pelo sistema);
- número de atributos (condições) necessárias para a análise das regras;
- um exemplo de entrada de dados por linha.

Um exemplo de um arquivo de dados é o `pawlak.in`, descrito a seguir:

```
This is an input file for analize.
4
b=2
c=3
d=4
e=1
```

6.3.2 Execução da Shell

A *Shell* é invocada via a execução do arquivo `esurf.exe`, quando compilado em ambiente *Windows* de 32 bits (95/98 ou NT); `./esurf`, quando compilado em ambiente *Linux*, ou pelo botão *Run Esurf* presente no protótipo ILROS. A tela inicial do esurf é apresentada na Figura 6.9.

A *Shell* apresenta uma barra horizontal de ferramentas que permitem o usuário manipular os arquivos de dados. Todas essas ferramentas também estão disponibilizadas pela *shell* através de *menus*.

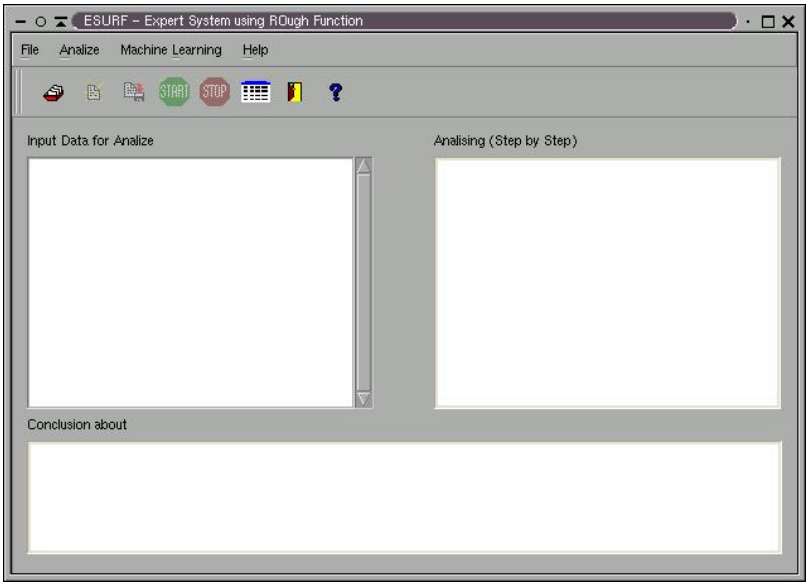


Figura 6.9: Tela inicial da *Shell ESURF*.

6.3.3 Botões da Barra Horizontal de Ferramentas

Uma descrição dos botões que fazem parte da barra horizontal de ferramentas é apresentada a seguir:

- *Load Rules* - utilizado para carregar um arquivo de regras;
- *Load Data File* - utilizado para carregar um arquivo de entrada de dados com informações pré-definidas para análise;
- *Save Data File* - utilizado para salvar um arquivo de entrada em disco;
- *Start Analize* - inicia a análise das informações. A conclusão é apresentada na caixa de texto localizada na parte inferior da *shell*;
- *Stop Analize* - termina a análise das informações;
- *Run Ilros* - utilizado para executar o protótipo *ILROS*;
- *Close* - utilizado para sair do sistema;
- *About* - utilizado para apresentar alguns informações sobre a *shell*.

É importante observar que os botões *Load Data File*, *Save Data File*, *Start Analize* e *Stop Analize* se encontram ativos apenas quando houver um arquivo de regras disponível (carregado em memória). A Figura 6.10 apresenta a *shell* com o arquivo *pawlak.tree* carregado em memória.

Com um arquivo de regras carregado em memória o usuário pode entrar com as informações necessárias à análise através do botão *Load Data File*, ou manualmente clicando em cada item da lista de informações localizada na parte superior-esquerda da *shell*. Ao clicar em um item, a tela *Data Input* (Figura 6.11) permite que o usuário atribua um valor para a informação (atributo) selecionada.

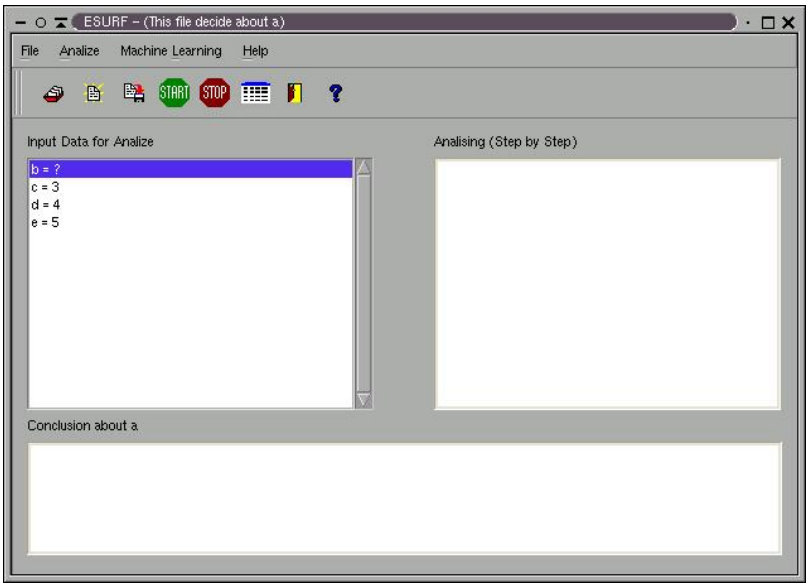


Figura 6.10: Shell com arquivo *pawlak.tree* carregado.

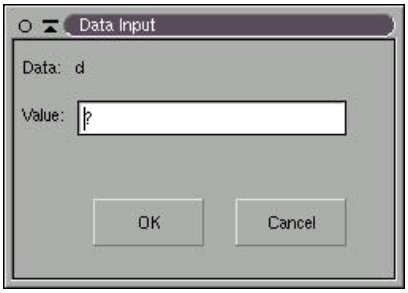


Figura 6.11: Tela para entrada de dados manualmente.

6.4 Comentários sobre a Implementação

Os algoritmos foram implementados sob a forma de bibliotecas, buscando ao máximo, isolar o ambiente gráfico dos algoritmos da TCA. Para isto foi utilizada a linguagem de programação C++, utilizando-se o compilador gcc (*Gnu C Compiler*) disponível em `http://www.gnu.org/`. Estas bibliotecas foram implementadas sob a forma de classes, utilizando-se, para isto, do paradigma de orien-

tação a objetos. Isso, simplifica o uso destas bibliotecas, bem como permite que as mesmas possam ser expandidas com facilidade. As classes implementadas são apresentadas na tabela 6.1

Tabela 6.1: Principais Classes implementadas

Classes	Descrição
krs	Classe implementando um SRC
repkrs	Representação de um SRC
appspace	Representação de um espaço aproximado
approx	Classe que agrupa as aproximações de um conjunto
arule	Classe implementando regra de decisão
binarytree	Classe implementando a árvore de decisão gerada pelo algoritmo *lmurf*
setfunction	Classe implementando operações de conjuntos
ilros	Classe implementando o protótipo ILROS
esurf	Classe implementando a shell ESURF

O protótipo ILROS e a shell ESURF foram implementadas utilizando-se da linguagem de programação C++ e da biblioteca gráfica *wxWindows* disponível em `http://www.wxwindows.org/`. Esta biblioteca permite o desenvolvimento de aplicações gráficas para os sistemas operacionais *Linux*, *Windows* e *MacOs*, permitindo o desenvolvimento de aplicações multiplataformas.

O ILROS e o ESURF são sistemas desenvolvidos sobre a licença *GPL (GNU GENERAL PUBLIC LICENSE)*. Maiores informações sobre esta licença podem obtidas em [GNU (1999)].

Capítulo 7

Conclusão

Os estudos realizados e descritos nesta monografia focalizaram a Teoria de Conjuntos Aproximados e principalmente sua extensão aos conceitos matemáticos de funções como um formalismo para o estabelecimento de técnicas de Aprendizado de Máquina (AM).

Os estudos demonstraram a possibilidade de uso do formalismo de funções aproximadas como subsídio ao desenvolvimento de algoritmos de aprendizado de máquina. Sob esta perspectiva foi desenvolvido uma algoritmo de Aprendizado Indutivo de Máquina, denominado *lmurf - Learning Machine using ROugh Functions*, como pôde ser verificado no Capítulo 5. Este algoritmo gera uma árvore binária de decisão que pode ser usada em análise especialista.

Além do algoritmo de aprendizado de máquina, estes estudos também culminaram no desenvolvimento de uma simples *shell* de um sistema especialista. A esta *shell* foi atribuída o nome de *ESURF - Expert System using ROugh Functions*. Esta *shell*, no estágio de desenvolvimento que se encontra, analisa um conjunto de regras (regras geradas pelo algoritmo *lmurf*) e retorna uma conclusão sobre uma informação (atributo) ao usuário. Como trabalhos futuros, pretende-se:

- realizar estudos que comprovem a viabilidade de se usar o formalismo de funções aproximadas para o estabelecimento de técnicas de Aprendizado de Máquina;
- realizar melhorias na *shell ESURF*, como o desenvolvimento de um motor de inferência mais elaborado e que forneça resultados mais precisos;
- realizar melhorias no algoritmo *lmurf*, como possibilitar o algoritmo a gerar uma árvore de decisão ternária, sendo este terceiro nó, um nó que representa

valores desconhecidos dos atributos;

- realizar estudos de medidas que permitam calcular a credibilidade de uma conclusão.

Além disso, encontra-se em fase de desenvolvimento um sistema de ajuda (*help*) para o ILROS e para o ESURF.

Referências Bibliográficas

[Bonissone (1991)] Bonissone, P. Plausible reasoning, In: Shapiro, S. C.; Eckroth, D. & Valassi, G. A. (Eds.) *Encyclopedia of Artificial Intelligence.* New York, John Wiley & Sons,1991. p.854-863.

[Boose (1989)] Boose, J. H. A knowledge acquisition: techniques and tools. Knowledge Aquisition (1): 3-37, 1989.

[Domingues & Uchôa (2001)] Domingues, M. A. & Uchôa, J. Q. *Implementação e Desenvolvimento de Técnicas de Descoberta de Conhecimento e Tratamento de Incerteza com Ênfase na Teoria de Conjuntos Aproximados*. Relatório de Iniciação Científica - Pibic/CNPq. Lavras, DCC-UFLA, 2001. 32p.

[Fausett (1993)] Fausett, L. V. *Fudamental of newural networks: Architectures, Algorithms And Applications.* New York, Prentice Hall, 1993.

[GNU (1999)] Free Software Foundation. *GNU's Not Unix! the GNU project and the Free Software Foundation (FSF)*. url: `http://www.gnu.org/`.

[Klir & Yuan (1995)] Klir, J. G. & Yuan, B. *Fuzzi sets and fuzzy logic: theory and application*. New Jersey, Prentice Hall, 1995.

[Langley (1996)] Langley, P. Elements of machine learning. San Francisco, Morgan Kaufmann, 1996.

[McCulloch & Pitts (1943)] McC16ullock, W. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematica Biophysics*, (5):115-137.

[Michalski & Tecuci (1993)] Michalski, R. S. & Tecuci, G. Multistrategy learning. Tutorial T15, IJCAI-1993, 1993.

[Mitchell (1997)] Mitchell, T. M. Machine learning. New York, McGraw-Hill, 1997. [Ng & Abramsom (1990)] Ng, K.C. & Abramsom, B. Uncertainty management in expert systems. IEEE Expert, (April): 29-47, 1990.

[Nicoletti (1994)] Nicoletti, M. C. *Ampliando os limites do aprendizado indutivo de máquina através das abordagens construtiva e relacional*. São Carlos, Universidade de São Paulo, 1994. (Tese de Doutorado).

[Nicoletti & Uchôa (1997)] Nicoletti, M. C. & Uchôa, J. Q. *O uso de funcoes de pertinencia na caracterização dos principais conceitos da teoria de conjuntos aproximados*. Relatório Técnico do Departamento de Computação 005/97. São Carlos, DC-UFSCar, 1997. 26p.

[Nicoletti; Uchôa & Baptistini (2001)] Nicoletti, M. C.; Uchôa, J. Q. & Baptistini M. T. Z. Rough relations properties. *Int. J. Appl. Math Comput. Sci.*, 11(3):621-635, 2001.

[Pawlak (1982)] Pawlak, Z. Rough sets. *International Journal of Computer and Information Sciences*, 11(5):341-356, 1982.

[Pawlak (1991)] Pawlak, Z. *Rough sets: theoretical aspects of reasoning about data*. London, Kluwer, 1991.

[Pawlak (1994)] Pawlak, Z. Hard and soft sets. In: Ziarko, W. P. (Ed). *Rough sets, fuzzy sets and knowledge discovery*. London, Springer-Verlag, 1994. p.130-135.

[Quinlan et alii(1986)] Quinlan, J. R; Compton, P. J; Horn, K. A. & Lazarus, L. *Inductive Knowledge Acquisition: a Case Study.* In: Proceedings of the 2nd Australian Conference on Applications of ES, Sydney, 1986, pp 183-203.

[Shafer (1976)] Shafer, G. *A mathemathical theory of evidence*. Princeton, Princeton University Press, 1976.

[Uchôa et alii(1997)] Uchôa, J. Q.;Panotim, S. M. & Nicoletti, M. C. *Elementos da Teoria de Dempster-Shafer.* Relatório do Departamento de Computação 007/97. São Carlos, DC-UFSCar, 1997. 34p.

[Uchôa (1998)] Uchôa, J. Q. *Representação e indução de conhecimento usando teoria de conjuntos aproximados*. São Carlos, UFSCar, 1998. 237p. (Dissertação de Mestrado).

[Uchôa & Nicoletti (1998)] Uchôa, J. Q. & Nicoletti, M. C. *ILROS: um sistema de aprendizado indutivo de máquina baseado em conjuntos aproximados.*(Submetido para publicação como Relatório Técnico do Departamento de Computação da UFSCar).

[Uchôa & Nicoletti (1999)] Uchôa, J. Q. & Nicoletti, M. C. ILROS: um sistema de aprendizado de máquina para domínios incompletos. In: *Anais do 4. SBAI - Simpósio Brasileiro de Automação Inteligente*. Escola Politécnica da USP, São Paulo (SP), 8 a 19 de Setembro de 1999. p. 314-319.

[Shaw & Gentry (1990)] Shaw, M. J. & Gentry, J. A. Inductive learning for risk classification. IEEE Expert, (February):47-53, 1990.

[Wong et alii (1986)] Wong, S. K. M.; Ziarko, W. & YE, R. L. Comparison of rough-set and statistical methods in inductive learning, *Internacional Journal of Man-Machine Studies*, (24):53-72, 1986.

[Zadeh (1965)] Zadeh, L. A. Fuzzy sets. *Information and Control*, (8):338-353, 1965.

Apêndice A

Pseudocódigo do algoritmo *lmurf*

A.1 Introdução

O objetivo deste apêndice é apresentar o pseudocódigo de procedimentos utilizados na implementação do algoritmo *lmurf - Learning Machine using ROugh Functions*.

A.2 Notação Utilizada

No que segue será considerado o Sistema de Representação de Conhecimento $S = (E, Q, V, \rho)$, onde

$$E = \{e_1, e_2, \ldots, e_m\}, |E| = m$$

$$Q = \{q_1, q_2, \ldots, q_n\}, |Q| = n$$

$$V_{q_j} = \{v_{j_1}, v_{j_2}, \ldots, v_{j_{i_j}}\}, |V_{q_j}| = i_j, j = 1, \ldots, n$$

Tem-se pois um SRC com m exemplos, cada um descrito por n atributos, onde cada atributo q_j pode assumir i_j valores distintos, $j = 1, \ldots, n$. Ou seja, um exemplo qualquer e_i é representado por um vetor de n posições, cada uma delas preenchida por um valor do atributo correspondente àquela posição, extraído de V_1 até V_n respectivamente. Um exemplo $e_i (1 \leq i \leq m)$ é notado pelo vetor $[v_{1e_i}, v_{2e_i}, \ldots, v_{ne_i}]$, onde cada um dos $v_{je_i} (1 \leq j \leq n)$ corresponde ao valor do atributo q_j no exemplo e_i.

Tabela A.1: Sistema de Representação de Conhecimento

	q_1	q_2	q_3	...	q_n
e_1	v_{1e_1}	v_{2e_1}	v_{3e_1}	...	v_{ne_1}
e_2	v_{1e_2}	v_{2e_2}	v_{3e_2}	...	v_{ne_2}
e_3	v_{1e_3}	v_{2e_3}	v_{3e_3}	...	v_{ne_3}
e_m	v_{1e_m}	v_{2e_m}	v_{3e_m}	...	v_{ne_m}

A.3 Pseudocódigos de alguns conceitos e medidas da TCA

Segue a descrição, em uma linguagem tipo *Pascal*, de três algoritmos relacionados à TCA e que são utilizados na implementação do algoritmo *lmurf*. Alguns procedimentos auxiliares utilizados, tais como *determina_primeiro_nao_usado* e *nro_elementos*, são deixados indefinidos, uma vez que são facilmente implementáveis.

A.3.1 Geração da Representação de um SRC

A geração da representação de um SRC consiste na geração dos conjuntos elementares do SRC. O algoritmo *rep* consiste em considerar como conjunto elementar o conjunto das instâncias que têm o mesmo valor de atributo, para todos os n atributos: $q_1, \ldots, q_n$. Este algoritmo tem como entrada: um SRC descrito pelos conjuntos de elementos $E = \{e_1, \ldots, e_m\}$ e de atributos $Q = \{q_1, \ldots, q_n\}$. Com isso, para todo exemplo e_i, $1 \leq i \leq m$, verifica se não existem exemplos $e_j, i \leq j \leq m$, cujos valores em Q (para todo $q_k, 1 \leq k \leq n$), sejam idênticos ao de e_i. Os exemplos idênticos a e_i são então reunidos em um único conjunto elementar. Após esgotados os i exemplos, o procedimento devolve em R o conjunto formado por todos os conjuntos elementares do SRC em questão.

```
procedure rep(E,Q,R);
{
  Entrada: E = {e_1, e_2, ..., e_m}, conjunto de m exemplos,
           descrito por n atributos, Q = {q_1, ..., q_n}
  Saída: R = {E_1, E_2, ..., E_s}, representação do SRC dado por (E, Q, C)
}
begin
   {inicialização}
   i := 1;
   s := 0;
   R := {};
   m := nro_elementos(E);
   n := nro_elementos(Q);
   for k := 1 to m do usado[k] := false;
   {determinação dos conjuntos elementares}
   while i <= m do
   begin
      s := s + 1;
      E_s := {e_1};
      usado[i] := true;
      j := i + 1;
      r := 1;
      while j <= m do
      begin
         while usado[j] and j <= m do j := j + 1;
         if j <> m + 1 then
            while r <= n and v_{re_i} = v_{re_j} do r := r + 1;
            if r = n + 1 then
            begin
               E_s := E_s ∪ {e_j};
               usado[j] := true;
            end;
            r := 1;
            j := j + 1;
      end;
      R := R ∪ {E_s};
      determina_primeiro_nao_usado(i);
   end;
end.
```

A.3.2 Determinação da Aproximação Inferior de um Dado Conjunto

A aproximação inferior de um dado conjunto X em um SRC consiste no conjunto formado pela união de todos os elementos dos conjuntos elementares do SRC que estão completamente contidos em X.

```
procedure aprox_inf(E,Q,X,I);
{
  Entrada: E = {e_1, e_2, ..., e_m}, conjunto de m exemplos,
           descrito por n atributos, Q = {q_1, ..., q_n}
           X = {x_1, x_2, ..., x_p}, conjunto para calcular a aproximação inferior
  Saida: I = {E_l1, E_l2, ..., E_lk}, 0 ≤ k ≤ |R|,
           conjunto das classes de equivalência do SRC, que
           estão contidos inteiramente em X
}
begin
   {inicialização}
   rep(E,Q,R);
   num := nro_elementos(R);
   p := nro_elementos(X);
   for i := 1 to num do usado[i] := false;
   i := 1;
   I := {};
   {determinação da aproximação inferior de X}
   while i <= p do
   begin
      {determinação da classe de equivalência que contém x_i}
      j := 1;
      while (j <= num) and (not( x_i ∈ E_j )) do j := j + 1;
      {verifica se E_j está contido inteiramente em X}
      if not usado[j] then
      begin
         b := 1;
         contido := true;
         k_j := nro_elementos(E_j);
         while (b <= k_j ) and (contido) do
         begin
            if e_b in X then b := b + 1
            else contido := false;
         end;
         if contido then I := I ∪ E_j;
         usado[j] := true;
         i := i + 1;
      end;
   end;
end.
```

O algoritmo tem como entrada: a) um SRC descrito pelo conjunto de elementos $E = \{e_1, \ldots, e_m\}$ e de atributos, $Q = \{q_1, \ldots, q_n\}$; b) o conjunto do qual se vai determinar a aproximação inferior, $X = \{x_1, x_2, \ldots, x_p\}$. Cada $x_i, 1 \leq i \leq p$, é fornecido como um vetor de valores de atributo $[v_{1x_i}, v_{2x_i}, \ldots, v_{nx_i}]$. O procedimento produz como saída o conjunto I, que contém a aproximação inferior de X. Este algoritmo faz uso do algoritmo *rep*, uma vez que a determinação da aproximação inferior usa a representação do SRC.

Este algoritmo inicialmente determina a representação do SRC, através da chamada *rep(E,Q,R)*, cujo argumento de saída, $R = \{E_1, E_2, \ldots, E_s\}$, é a coleção das classes de equivalência induzidas por Q. Depois, para cada um dos p elementos $x_i \in X$, determina em qual das classes de equivalência x_i se encontra. Verifica, então, se todos os elementos da classe determinada estão contidos em X. Em caso afirmativo, todos os elementos da referida classe passam a fazer parte da aproximação inferior de X, I; caso contrário, nenhum elemento dessa classe pertence à I.

A.3.3 Determinação da Aproximação Superior de um Dado Conjunto

A aproximação superior de um dado conjunto X em um SRC consiste no conjunto formado pela união de todos os elementos dos conjuntos elementares do SRC que têm intersecção não-vazia com X.

```
procedure aprox_sup(E,Q,X,S);

{
  Entrada: E = {e1, e2, ..., em}, conjunto de m exemplos,
           descrito por n atributos, Q = {q1, ..., qn}
           X = {x1, x2, ..., xp}, conjunto para calcular a aproximação superior
  Saida: S = {El1, El2, ..., Elk}, 0 ≤ k ≤ |R|, conjunto de classes de
           equivalência do SRC, que tem intersecção não vazia com X
}
begin
   {inicialização}
   rep(E,Q,R);
   num := nro_elementos(R);
   p := nro_elementos(X);
   for i := 1 to num do usado[i] := false;
   i := 1;
   S := {};
   {determinação da aproximação superior de X}
   while i <= p do
   begin
      {determinação da classe de esquivalência contendo xi}
      j := 1;
      while (j <= num) and (not( xi ∈ Ej )) do j := j + 1;
      {verifica se esta contido inteiramente em X}
      if not usado[j] then
      begin
         S := S ∪ Ej;
         usado[j] := true;
      end;
      i := i + 1
   end;
end.
```

O algoritmo aprox_sup tem como entrada: a) um SRC descrito pelo conjunto de elementos $E = \{e_1, \ldots, e_m\}$ e de atributos, $Q = \{q_1, \ldots, q_n\}$; b) o conjunto do qual se vai determinar a aproximação superior, $X = \{x_1, x_2, \ldots, x_p\}$. Cada $x_i, 1 \leq i \leq p$, é fornecido como um vetor de valores de atributo $[v_{1x_i}, \ldots, v_{nx_i}]$. O procedimento produz como saída o conjunto S, que contém a aproximação superior de X. Este algoritmo faz uso também do algoritmo *rep*, uma vez que a determinação da aproximação superior usa a representação do SRC.

Este algoritmo inicialmente determina a representação do SRC, através da chamada *rep(E,Q,R)*, cujo argumento de saída, $R = \{E_1, E_2, \ldots, E_s\}$, é a coleção das classes de equivalência induzidas por Q. Depois, para cada um dos p elementos $x_i \in X$, determina em qual das classes de equivalência x_i se encontra. Todos os elementos dessa classe de equivalência passam, então, a fazer parte da aproximação superior de X, S.

A.4 Descrição em Pseudocódigo do algoritmo lmurf

O objetivo desta seção é apresentar o pseudocódigo do algoritmo *lmurf* que é constituído por 9 procedimentos, sendo o principal denominado *lmurf*. O algortimo utiliza-se de estruturas especiais para armazenar algumas informações úteis (como nomes de atributos e conjuntos de elementos) e que são obtidas durante o cálculo das F_{inf} e F_{sup}, e também durante o cáculo do valor da pertinência aproximada. O algoritmo também utiliza uma outra estrutura de armazenamento - uma árvore binária - para armazenar as proposições que permitem decidir o percurso que se deve seguir ao percorre a árvore de decisão para buscar uma conclusão. No algoritmo, onde se fizer necessário a passagem de variáveis valores, em uma dada função, a presença de um $\uparrow$ antecedendo uma variável referencia a variável e uma variável referenciada entre parenteses ($(\uparrow variavel)$) indica uma derreferenciação da variável. Isso é feito para deixar claro que a variável não é passada por valor (podendo ser alterada dentro da função) e sim por referência (implementada em linguagens mais usuais utilizando referências ou ponteiros).

```
procedure lmurf(E,Q,C,d,Tree);
{
  Entrada: E = {e1,e2,...,em}, conjunto de m exemplos,
           descritos por n atributos, Q = {q1,...,qn}
           C = {c1,c2,...,pj} ⊆ Q, indica o conjunto de condições
           d, atributo de decisão
  Saída: Tree, árvore de decisão gerada pelo algoritmo lmurfAux
}
begin
```

```
    {inicialização}
    pert := 0;
    lmurfAux(E,Q,C,d,pert,Tree);
end.
```

O algoritmo lmurf tem como entrada: a) um SRC descrito pelo conjunto de elementos $E = \{e_1, \ldots, e_m\}$ e de atributos, $Q = \{q_1, \ldots, q_n\}$; b) um conjunto de condições $C \subseteq Q$ e c) um atributo de decisão $d \in Q$. O algoritmo inicializa a variável pert (com valor zero) e chama o procedimento *lmurfAux*. Este procedimento utiliza-se dos seguintes procedimentos:

- *calcula_melhor_Finf* e *calcula_melhor_Fsup* para obter a melhor aproximação inferior e aproximação superior de uma função aproximada;
- *constroi_sub_arvore* para construir sub-árvores que serão inseridas na árvore de decisão;
- *extrai_proposicao* para montar as proposições que serão inseridas nos nós da árvore de decisão;
- *calcula_melhor_pertinencia* para calcular o maior valor para a função de pertinência aproximada.

O procedimento *lmurfAux* produz como saída uma árvore de decisão.

```
procedure lmurfAux(E,Q,C,d,pert,Tree);
{
  Entrada: E = {e1,e2,...,em}, conjunto de m exemplos,
           descritos por n atributos, Q = {q1,...,qn}
           C = {c1,c2,...,pj} ⊆ Q, indica o conjunto de condições
           d, atributo de decisão
           pert, valor inicial da pertinência aproximada
  Saída: Tree, árvore de decisão gerada pelo algoritmo lmurfAux
}
begin
   {inicialização}
   restart := false;
   nroE := nro_elementos(E);
   while (nroE > 0) and (not(restart)) do
   begin
      restart := true;
      {gera os conjuntos elementares para o atributo de decisão}
      rep(E,d,R);
      repDecSRC := R;
      nroDecSets := nro_elementos(repDecSRC);
      if nroDecSets > 1 then
      begin
```

```
calcula_melhor_Finf(E,Q,C,d,melhorFinf);
elemFinf := melhorFinf.elementos; {elementos de melhorFinf}
nroFinf := nro_elementos(elemFinf);
{Se existe melhorFinf}
if nroFinf > 0 then
begin
   restart := false;
   {na chamada da função constroi_sub_arvore as variáveis E e
   C são passadas por referência (↑E e ↑C)
   constroi_sub_arvore(↑E,Q,↑C,d,melhorFinf,Tree);
   nroE := nro_elementos(E);
end
else
begin
   {calcula F_sup com maior número de elementos
   e menor do que U}
   calcula_melhor_Fsup(E,Q,C,d,melhorFsup);
   elemFsup := melhorFsup.elementos; {elementos de Fsup}
   nroFsup := nro_elementos(elemFsup);
   {Se não existe F_inf e F_sup ≠ U}
   if nroFsup > 0 then
   begin
      restart := false;
      {na chamada da função constroi_sub_arvore as variáveis E e
      C são passadas por referência (↑E e ↑C)
      constroi_sub_arvore(↑E,Q,↑C,d,melhorFinf,Tree);
      nroE := nro_elementos(E);
   end;
end;
{Se não existem F_inf e F_sup}
if restart = true then
begin
   calcula_melhor_pertinencia(E, C, repDecSRC, melhorPert);
   mu := melhorPert.valor;
   atributo := melhorPert.atributo;
   rep(E, atributo, R);
   nroR := nro_elementos(R);
   if (mu > pert) and (nroR > 1) then
   begin
      C_aux := C;
      {divide o SRC em dois novos SRC (com e sem
        os elementos de melhorPert)}
      elementos := melhorPert.elementos;
      E_aux := E − elementos;
      extrai_proposicao(elementos, atributo, condRule);
      inseri_noh(condRule, false, Tree);
      C := C − atributo;

      {Chamada recursiva com os elementos de melhorPert}
      lmurfAux(elementos, Q, C, d, mu, Tree);
      rep(E_aux, atributo, R);
      nroR := nro_elementos(R);
      if nroR = 1 then
```

```
                begin
                   C_aux := C_aux - atributo;
                end;
                {Chamada recursiva sem os elementos de melhorPert}
                lmurfAux(E_aux, Q, C_aux, d, mu, Tree);
             end
             else
             begin
                {calcula a proposição de decisão e inseri na árvore}
                extrai_proposicao(E, d, decRule);
                inseri_noh(decRule, true, Tree);
             end;
          end;
       end
       else
       begin
          {calcula a proposição de decisão e inseri na árvore}
          extrai_proposicao(E, d, decRule);
          inseri_noh(decRule, true, Tree);
          E := ∅;
          nroE := nro_elementos(E);
       end;
    end;
end.
```

O procedimento *lmurfAux*, tem como entrada: a) um SRC S descrito pelo conjunto de elementos $E = \{e_1, \ldots, e_m\}$ e de atributos, $Q = \{q_1, \ldots, q_n\}$; b) um conjunto de condições $C \subseteq Q$; c) um atributo de decisão $d \in Q$ e d) pert, variável que armazena o valor inicial da pertinência aproximada.

O algoritmo inicializa a variável pert com o valor zero e calcula $Class_s(d)$. Se $|Class_s(d)| = 1$, uma proposição de conclusão é gerada e inserida em um novo nó da árvore de decisão. Se $|Class_s(d)| \neq 1$, o algoritmo calcula $F_{inf}(X_r)$, (para todo $X_r \in Class_s(d)$, $1 \leq r \leq |Class_s(d)|$), com maior número de elementos e gera proposição(ões) consistentes para os nós da árvore de decisão. Se $F_{inf}(X_r) = \varnothing$, o algoritmo irá calcular $F_{sup}(X_r)$ (para todo $X_r \in Class_s(d)$, $1 \leq r \leq |Class_s(d)|$) e escolher $F_{sup}(X_r) \neq U$ com maior número de elementos para gerar proposição(ões) inconsistentes para os nós da árvore de decisão. Caso $F_{inf}(X_r) = \varnothing$ e $F_{sup}(X_r) = U$, o algoritmo calcula $\mu_{X_r}^{B_i}(x)$ para todo $X_r \in Class_s(d)$, $1 \leq r \leq |Class_s(d)|$, e todo x pertencente a cada conjunto elementar de cada um dos espaços aproximados $B_i = (E, P_i)$, tal que P_i é a relação de equivalência induzida pelo atributo $q_i \in C$, $1 \leq i \leq |C|$. Se $|\mu_{X_r}^{B_i}(x)| \leq pert$ ou $|B_i| = 1$, uma proposição de conclusão é gerada e inserida em um novo nó da árvore de decisão, senão o algoritmo divide o SRC S em dois novos SRCs, S_1 e S_2, através do maior valor obtido com o cálculo da função de pertinência aproximada $\mu_{X_r}^{B_i}(x)$. O SRC S_1 é descrito pelo conjunto de elementos

$E_1 = \{e_1, \ldots, e_k\}$ e de atributos, $Q = \{q_1, \ldots, q_n\}$; b) um conjunto de condições $C \subseteq Q$ e c) um atributo de decisão $d \in Q$ e o SRC S_2 é descrito pelo conjunto de elementos $E_2 = \{e_1, \ldots, e_j\}$ e de atributos, $Q = \{q_1, \ldots, q_n\}$; b) um conjunto de condições $C \subseteq Q$ e c) um atributo de decisão $d \in Q$. O algoritmo gera uma proposição baseada na função de pertinência aproximada e a insere em um novo nó na árvore de decisão. O procedimento *lmurf* é então reiniciado recursivamente com S_1 e depois com S_2. A cada proposição gerada é removido de E os elementos e_m,$1 \leq m \leq |E|$ que geraram as proposições. O algoritmo termina quando $E = \varnothing$, retornando a árvore de decisão.

```
procedure calcula_melhor_Finf(E,Q,C,d,melhorFinf);
{
  Entrada: E = {e1,e2,...,em}, conjunto de m exemplos,
           descritos por n atributos, Q = {q1,...,qn}
           C = {c1,c2,...,pj} ⊆ Q, indica o conjunto de condições
           d, atributo de decisão
  Saída: melhorFinf, variável que armazena os elementos e atributos de
         F_inf que contem o maior número de elementos
}
begin
   {inicialização}
   nroC := nro_elementos(C);
   {gera os conjuntos elementares para o atributo de decisão}
   rep(E,d,R);
   repDecSRC := R;
   nroDecSets := nro_elementos(repDecSRC);
   {calcula F_inf com maior número de elementos}
   w := 0;
   for i := 1 to nroDecSets do
   begin
      for j := 1 to nroC do
      begin
         aprox_inf(E,cj,repDecSRC[i],I);
         nroI := nro_elementos(I);
         if nroI > w then
         begin
            w := nroI;
            melhorFinf.elementos := I;
            melhorFinf.atributo := cj;
         end;
      end;
   end;
end.
```

O procedimento calcula_melhor_Finf tem como entrada: um SRC descrito pelos conjuntos de elementos $E = \{e_1, \ldots, e_m\}$ e de atributos $Q = \{q_1, \ldots, q_n\}$; b) um conjunto de condições $C \subseteq Q$ e c) um atributo de decisão d. O algoritmo produz como saída a variável melhorFinf que armazena os elementos e atributos de F_{inf}

que contem o maior número de elementos.

```
procedure calcula_melhor_Fsup(E,Q,C,d,melhorFsup);
{
  Entrada: E = {e1,e2,...,em}, conjunto de m exemplos,
           descritos por n atributos, Q = {q1,...,qn}
           C = {c1,c2,...,pj} ⊆ Q, indica o conjunto de condições
           d, atributo de decisão
  Saída: melhorFsup, variável que armazena os elementos e atributos de
         Fsup que contem o maior número de elementos
}
begin
   {inicialização}
   nroE := nro_elementos(E);
   nroC := nro_elementos(C);
   {gera os conjuntos elementares para o atributo de decisão}
   rep(E,d,R);
   repDecSRC := R;
   nroDecSets := nro_elementos(repDecSRC);
   {calcula Fsup com maior número de elementos}
   w := 0;
   for i := 1 to nroDecSets do
   begin
      for j := 1 to nroC do
      begin
         aprox_sup(E,cj,repDecSRC[i],S);
         nroS := nro_elementos(S);
         if (nroS < nroE) and (nroS > w) then
         begin
            w := nroS;
            melhorFsup.elementos := S;
            melhorFsup.atributo := cj;
         end;
      end;
   end;
end.
```

O procedimento calcula_melhor_Fsup tem como entrada: um SRC descrito pelos conjuntos de elementos $E = \{e_1, \ldots, e_m\}$ e de atributos $Q = \{q_1, \ldots, q_n\}$; b) um conjunto de condições $C \subseteq Q$ e c) um atributo de decisão d. O algoritmo produz como saída a variável melhorFsup que armazena os elementos e atributos de F_{sup} que contem o maior número de elementos.

```
procedure constroi_sub_arvore(↑E,Q,↑C,d,melhorF,subTree);
{
  Entrada: ↑E, onde (↑E) = {e1,e2,...,em}, referência para um conjunto
           de m exemplos, descritos por n atributos, Q = {q1,...,qn}
           ↑C, onde (↑C) = {c1,c2,...,pj} ⊆ Q, indica uma referência
           para o conjunto de condições d, atributo de decisão melhorF,
           valor da variável obtida com o cálculo da melhor Finf ou
```

```
            melhor F_sup
    Saída: subTree, sub–árvore gerada pelo algoritmo
  }
  begin
     {inicialização}
     elemF := melhorF.elementos; {elementos de melhorF}
     attrF := melhorF.atributo; {attributo de melhorF}
     {calcula a proposição de condição e inseri na árvore}
     extrai_proposicao(elemF, attrF, condRule);
     inseri_noh(condRule, false, subTree);
     {calcula a proposição de decisão e inseri na árvore}
     extrai_proposicao(elemF, d, decRule);
     inseri_noh(decRule, true, subTree);
     {atualização dos elementos de E}
     (↑ E) := (↑ E) − elemF;
     {calcula U/R para o atributo de melhorF}
     rep(E, attrF,R);
     nroR := nro_elementos(R);
     if nroR = 1 then
     begin
        {atualização dos atributos de C}
        (↑ C) := (↑ C) − attrF;
     end;
  end.
```

O procedimento constroi_sub_arvore tem como entrada: uma referência para um SRC descrito pelos conjuntos de elementos $\uparrow E$, onde $(\uparrow E) = \{e_1, \ldots, e_m\}$ e de atributos $Q = \{q_1, \ldots, q_n\}$; b) uma referência para um conjunto de condições $\uparrow C$, onde $(\uparrow C) \subseteq Q$; c) um atributo de decisão d e d) melhorF, que armazena a variável obtida com o cálculo da melhor F_{inf} ou melhor F_{sup}. O algoritmo utiliza-se do procedimento *extrai_proposicao* para montar as proposições que serão inseridas nos nós da sub-árvore produzida como saída do algoritmo e que irá fazer parte da árvore de decisão gerada pelo algoritmo *lmurfAux*.

```
  procedure extrai_proposicao(E, q, proposicoes);
  {
    Entrada: E = {e_1, e_2, ..., e_m}, conjunto de m exemplos,
    descritos por 1 atributo q
    Saida: proposicoes, conjunto com k proposições
  }
  begin
     {inicialização}
     usado := {};
     numElementos = nro_elementos(E);

     {Com um conjunto de elementos e um atributo é calculado um conjunto
       de proposições do tipo (atributo := valor)}
     proposicoes[1].atributo := Q;
     proposicoes[1].valor := v_{qe_1};
     usado[1] :=  v_{qe_1};
     w := 2;
```

```
    for i := 2 to numElementos do
    begin
       if not(v_qe_i in usado) then
       begin
          proposicoes[w].atributo := Q;
          proposicoes[w].valor := v_qe_i;
          usado[w] :=  v_qe_i;
          w := w + 1;
       end;
    end;
end.
```

O procedimento extrai_proposicao tem como entrada: a) um SRC descrito pelo conjunto de elementos $E = \{e_1, \ldots, e_m\}$ e pelo atributo q; O procedimento utiliza-se do atributo q e dos elementos de E para construir proposições (de condição ou decisão) no formato:

- $(a = v_{ae_1}) ou \ldots ou (a = v_{ae_j})$, onde a recebe o valor do atributo q e v_{ae_j} recebe o valor do elemento e_j, $1 \leq j \leq |E|$

```
procedure calcula_melhor_pertinencia(E, Q, setX, Member);

{
  Entrada: E = {e1, e2, ..., em}, conjunto de m exemplos,
           descrito por n atributos, Q = {q1, ..., qn}
           setX, conjunto de conjuntos elementares do atributo de
           decisão
  Saida:   Member, atributo e valor com maior pertinência
}
begin
   {inicialização}
   old_pert = 0;
   pert := 0;
   nroQ := nro_elementos(Q);
   {gera os conjuntos elementares para cada atributo do SRC}
   for i := 1 to nroQ do
   begin
      rep(E,q_i,R);
      repCSRC[i] := R;
   end;
   m := nro_elementos(setX);
   n := nro_elementos(repCSRC);
   {Separa as representações e seus conjuntos elementares}
   for i := 1 to n do
   begin
      condElemSets := repCSRC[i];
      w := nro_elementos(condElemSets);
      {Separa um conjunto dos conjuntos Elementares de uma representação}
      for j := 1 to w do
      begin
         elementarySet := condElemSets[j];
```

```
            {calcula a pertinência de um conjunto com relação a todos
              os conjuntos de setX}
            for k := 1 to m do
            begin
               Intersec := (elementarySet ∩ setX[k]);
               num1 := nro_elementos(Intersec);
               num2 := nro_elementos(elementarySet);
               pert := num1 / num2;
               if pert > old_pert then
               begin
                  old_pert := pert;
                  Member.valor := pert;
                  Member.elementos := elementarySet;
                  Member.atributo := q_i;
               end;
            end;
         end;
      end;
   end.
```

O procedimento calcula_melhor_pertinencia tem como entrada: a) um SRC descrito pelo conjunto de elementos $E = \{e_1, \ldots, e_m\}$ e de atributos $Q = \{q_1, \ldots, q_n\}$; b) $setX = \{X_1, X_2, \ldots, X_r\}$, onde $X_r = \{x_1, x_2, \ldots, x_k\}$ é o conjunto usado para calcular a pertinência aproximada. O procedimento gera representações (conjuntos elementares) para cada um dos atributos q_i, $1 \leq i \leq |Q|$, e calcula a pertinência aproximada de cada representação com cada conjunto de $setX$, retornando um objeto com atributo, elementos e valor da pertinência aproximada que apresenta maior valor.