Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

MLC^{++} BIBLIOTECA DE APRENDIZADO DE MÁQUINA EM C^{++}

Versão 1.0

LUIS CARLOS MOLINA FÉLIX
SOLANGE OLIVEIRA REZENDE
CRISTINA YOSHIMI DOI
MARCOS FERREIRA DE PAULA
MARIA JOSÉ ROMANATO

N$^{\underline{0}}$ 72

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos
Mar./1998

MLC++
Biblioteca de Aprendizado de Máquina em C++ [1]

Luis Carlos Molina Félix [2]
Solange Oliveira Rezende [2]
Cristina Yoshimi Doi [2]
Marcos Ferreira de Paula [2]
Maria Jose Romanato [3]

[2] Universidade de São Paulo
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística
Caixa Postal 668, 13560-970 - São Carlos - SP
{lmolina,solange,cydoi,mfpaula}@icmsc.sc.usp.br

[3] Universidade Estadual Paulista "Júlio de Mesquita Filho" -UNESP
Faculdade de Ciências e Letras
Departamento de Ciências de Educação
Rodovia Araraquara-Jaú km. 1, 14800-901 - Araraquara - SP
maze@fclar.unesp.br

Versão 1.0
Março de 1998

---

[1] Trabalho realizado com o auxílio do CNPq, IMP e da FAPESP.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Nesta última década, uma imensa quantidade de dados tem sido gerada pelas organizações, as quais estão perdendo competitividade por se acharem impossibilitadas de entender, interpretar e extrapolar esses dados. Métodos de aprendizado de máquina, métodos estatísticos e métodos de reconhecimento de padrões provêem algoritmos para extrair conhecimento de bases de dados ajudando a analisar as informações, encontrando padrões e melhorando a precisão de uma predição. Um problema que os usuários e analistas enfrentam quando tentam descobrir padrões é que existem muitos algoritmos disponíveis e é difícil determinar qual utilizar.

A MLC^{++}, uma Biblioteca de Algoritmos de Aprendizado de Máquina em C^{++}, foi projetada para ajudar na seleção de algoritmos para realizar tarefas específicas e/ou no desenvolvimento de novos algoritmos mais convenientes para outras tarefas. O seu projeto começou em 1993, na Universidade de Stanford, e trata-se de um software de domínio público (inclusive os códigos fontes) e a partir de 1995, encontra-se sob a responsabilidade da Silicon Graphics®. Atualmente, dezenas de pessoas estão usando a biblioteca e mais de 300 estão em uma lista de discussão. Uma breve descrição da biblioteca encontra-se em [Kohavi 94a].

A biblioteca MLC^{++} tem um papel dual: serve tanto como um sistema para os usuários finais quanto para projetistas de algoritmos. Essa biblioteca trabalha com mais de 30 algoritmos e algumas bases de dados para testes. A MLC^{++}, além de ser uma biblioteca de classes em C^{++}, provê utilitários e ferramentas que podem ser usadas isoladamente, sendo o benefício principal do seu uso a habilidade para juntar classes e as instâncias para explorar novos algoritmos ou variantes de algoritmos existentes.

A biblioteca MLC^{++} é assim chamada, pelo fato de não existir um único bom algoritmo de aprendizado de máquina para todas as tarefas. Isto foi teoricamente provado e demonstrado experimentalmente [Kohavi 97b]. Recomenda-se testar os diferentes algoritmos com a mesma base de dados, para comparar e determinar o melhor desempenho para aquele caso.

Cada algoritmo de aprendizado de máquina possui um formato próprio para os dados de entrada, sendo necessário, quase sempre, uma transformação na estrutura e formato desses dados ao usar um outro algoritmo. Uma das vantagens da biblioteca MLC^{++} é a transformação automática dos formatos dos dados de entrada para os diferentes algoritmos. A MLC^{++} se encontra disponível em http://www.sgi.com/Technology/mlc/index.html.

Quanto à organização do trabalho, a seção 2 trata das especificações gerais do ambiente IRIX® versão 6.2, da Silicon Graphics®, referentes a forma como são exibidas as saídas e o caminho de busca dos algoritmos. A seção 3 descreve a formatação necessária para os arquivos de entrada da MLC^{++} que são formados por um arquivo de treinamento, um arquivo de teste e um arquivo descritor, e que juntos compõem o *dataset*. A seção 4 tem como objetivo apresentar os utilitários da biblioteca, que são ferramentas que permitem cálculos estatísticos, medidas do desempenho dos algoritmos, obtenção de informações do *dataset*, entre outros. A seção 5 apresenta os diferentes indutores utilizados pela biblioteca MLC^{++}, assim como uma classificação deles. A seção 6 explica os diferentes tipos de *wrappers*. A seção 7 refere-se à forma como podem ser mostradas graficamente as saídas dos algoritmos, usando produtos de domínio público ou próprios da Silicon Graphics®. A seção 8 apresenta várias listas de discussão onde são tratados problemas referentes a essa biblioteca. A seção 9 apresenta as considerações finais deste trabalho.

2. CONFIGURAÇÃO DO AMBIENTE

Esta seção trata da estrutura física e da decomposição lógica da biblioteca MLC^{++}. Dependendo de seu uso, para projetistas ou para usuários finais, são criados diferentes diretórios com informações apropriadas para cada caso. As variáveis de ambiente não afetam o comportamento dos algoritmos, só estabelecem a forma de exibição das saídas e os caminhos de busca dos arquivos.

2.1. Estrutura dos Diretórios da MLC^{++}

Os arquivos com informações referentes a biblioteca estão organizados em alguns subdiretórios:

- **PATH :** diretório principal que contém todos os arquivos da biblioteca e o arquivo Descrip.txt que dá informação aproximada de como adquirir a biblioteca, permissão para uso, etc. Também contém os arquivos todo e longdo do que se deve fazer a curto e longo prazo, além dos melhoramentos descritos no próprio código.
- **db / :** diretório que contém as bases de dados para testes.
- **doc / :** diretório que contém a documentação pertinente à biblioteca.
- **inc** / **:** diretório que contém os arquivos .include. Todos os arquivos .h incluídos pelos usuários devem estar nesse diretório. Por razões técnicas os arquivos .c para *templates* devem ser acessados nesse mesmo diretório, assim eles serão ligados simbolicamente aos arquivos atuais dentro do diretório src /.
- **src** / **:** diretório de arquivos fontes com extensão .c.
  - **src/tests** / **:** rotinas de teste (arquivos de teste). Estes são arquivos que testam arquivos no diretório src /. Cada arquivo de teste tem um arquivo .c, um arquivo .exp (saídas esperadas) e opcionalmente um arquivo .cin para entradas.
- **bin** / **:** arquivos binários e shell scripts.

O administrador da MLC^{++} (*mlcadmin*) possui os diretórios da biblioteca no seu diretório local que constitui o **MLCPATH** da biblioteca. Alguns softwares são instalados abaixo do **MLCPATH** por conveniência (por exemplo, *dot*, *dotty*, C4.5). Todos os arquivos são do grupo mlc e os membros da biblioteca MLC^{++} devem pertencer a esse grupo: /etc/group. Os usuários autorizados a apagar e/ou adicionar arquivos devem estar no arquivo .rhosts.

2.2. Configuração das Variáveis de Ambiente

Nesta seção são descritas as variáveis de ambiente que são aplicáveis a qualquer programa da MLC^{++}. Elas não são fixadas através de opções e não afetam o comportamento inerente dos algoritmos, somente a exibição e o caminho de busca dos arquivos. Tais variáveis são definidas a seguir na Tabela 1:

Nome da opção	Domínio	Default	Explicação
MLCPATH	Pathnames	Diretório atual	Refere-se ao caminho que a MLC^{++} usa para abrir arquivos de dados no caso de não encontrá-los no diretório atual. src/tests/ é o *default*.
LD_LIBRARY_ PATH	Pathnames	Diretório atual	Deve-se incluir este diretório quando se têm arquivos executáveis .so (dinamicamente compartilhados) no diretório atual, a menos que estejam em /usr/lib, /usr/mlclib ou /lib.
MLCDIR	Pathnames	Diretório atual	Refere-se ao diretório raiz onde a MLC^{++} reside.

Nome da opção	Domínio	Default	Explicação
GENTAG	Pathnames	Diretório atual	Ativa o programa gerador de etiquetas. Todos os makefiles chamam este programa.
CCFLAGS	Pathnames	Diretório atual	Considera-se os flags do compilador como padrões. Todos os makefiles chamam o compilador CC com estes flags.
DUMPCORE	Pathnames	Arquivo de nomes	Se for definido, a MLC^{++} faz um *dumping*[1] do código em um arquivo quando acontece um erro fatal.
PROMPTLEVEL	required, basic, all	required	Determina quando entrar com uma opção.
LOGLEVEL	int ≥ 0	0	Refere-se a quantidade de informação assim impressa. Quanto maior o número, mais informação será mostrada. Os níveis 1 e 2 são normalmente usados; os níveis maiores ajudam a entender os algoritmos internamente e depurá-los.
OPTION_ DUMP	Nome do arquivo	-	Arquivo para fazer um *dumping* dos valores das opções.
DEBUGLEVEL	int ≥ 0	0	Indica o nível do depurador interno. Números mais altos executam mais verificações internas. Esta opção é usualmente relevante para os projetistas. Quando se obtém um erro, elevando o nível para 1 ou 2 obtém-se uma mensagem de erro melhor. A execução dos programas é mais lenta, aproximadamente 50% no nível 1 e aproximadamente 100% mais lento, no nível 2. Nota: a biblioteca é compilada em modo **FAST**, para quem ignora **DEBUGLEVEL**.
LINE_WIDTH	int ≥ 1	79 linhas ou 32Kb.	Indica o comprimento da linha de saída da MLC^{++}. O empacotamento automático acontecerá para quebrar palavras antes desta largura. Linhas empacotadas começarão com o string **WRAP_PREFIX**. A largura da linha *default* de 79 é fixada somente se a saída é um terminal tty (*isatty ()*). Se a saída é redirecionada a um arquivo, a largura *default* é de 32K.
WRAP_PREFIX	string	três espaços	Determina o prefixo assim inserido no começo de uma linha empacotada. Normalmente, alguns poucos espaços.
SHOW_LOG_ ORIGIN	yes, no	no	Mostra as mensagens referentes ao código fonte para o *log*. As mensagens do *log* aparecem quando a opção **LOGLEVEL** é fixada como 1 ou maior.
KEEPTEMP	yes, no	No	Mantém temporariamente os arquivos que são gerados. Isto é útil quando se quer analisar os arquivos usados que fazem interface com indutores externos.
LEFTYPATH	Pathnames	-	Indica o caminho dos arquivos visualizadores.
TMPDIR	Diretório	/var/tmp	Indica o diretório onde são gerados os arquivos temporários.

Tabela 1: Variáveis de Ambiente.

[1] Quando ocorre um erro na execução, é criado um arquivo dumpcore contendo o código gerado até aquele momento.

No Exemplo 1 pode ser observada a configuração das variáveis de ambiente para o sistema IRIX® versão 6.2, da Silicon Graphics®, que podem ser inseridas no arquivo .login.

Exemplo 1:

```
setenv MLCDIR "/usr/mlclib/mlc"
setenv LD_LIBRARY_PATH "/usr/mlclib/mlc:/lib:$MLCDIR"
setenv MLCPATH ".:$MLCDIR/db"
setenv LOGLEVEL 7
setenv LEFTYPATH "/usr/dot/sgi.mips2/lib/lefty"
setenv PATH ":/usr/sbin:/usr/bsd:/usr/dot/sgi.mips2/bin:/sbin:/usr/bin:/bin:
/usr/bin/X11:/usr/mlclib/mlc"
```

2.3. Opções de Configuração

Os utilitários da MLC^{++} possuem as seguintes opções para uma de linha de comando:

<utilitário> {[-s] [-o <opção do arquivo>] [-O <opção> = <valor>]}

A opção "-s" suprime as opções de ambiente. A opção "-o" permite ler opções de um arquivo contendo <opção> =<valor> em cada linha e a opção "-O" (maiúsculo) permite fixar uma opção específica. Os flags "-o" e "-O" podem ser repetidos várias vezes.

Cada opção usada por um programa deve ter um nome único. Por convenção, nomes de variáveis aparecem em letras maiúsculas e usam o caracter *underscore* (_) entre as palavras, embora algumas opções comuns não o utilizem (por exemplo, **LOGLEVEL**). As opções são fixadas de acordo com a seguinte hierarquia:

- **MLC^{++} default:** é o conjunto de valores *default* dentro da biblioteca MLC^{++}. Se a biblioteca não possui um valor *default* para uma variável, o valor da opção deve ser fornecido pelo usuário e deve ser considerada **required** (exigida).
- **Opção de ambiente:** uma variável de ambiente contém o valor *default*. Os usuários podem fixar as variáveis de ambiente usando o nome da própria variável. Por exemplo**, setenv DATAFILE foo** fixa a opção **DATAFILE** com o valor **foo**. Uma variável de ambiente tem precedência sobre os valores *default* da biblioteca. O flag da linha de comando "-s" suprimirá o valor de todas as variáveis de ambiente.
- **Opção para uma linha de comando**: esta opção foi descrita no começo desta seção.
- **Entrada do usuário:** os usuários podem alterar o valor das opções sob certos aspectos de **PROMPTLEVEL** (descritos abaixo). O valor sugerido ao usuário será obtido a partir do ambiente e dos padrões da MLC^{++}, descritos anteriormente. Se o valor for digitado, ele será tomado como o valor da opção. Se for apertada a tecla <return>, então o valor *default* será aceito. Se for digitado "?", então o sistema fornecerá informação de ajuda sobre a opção específica. A variável de ambiente **PROMPTLEVEL** determina quando o usuário deve fornecer o valor de uma opção, e pode ser:
  - **required-only:** o usuário deve fornecer o valor das opções exigidas. As opções que têm valores definidos através da variável de ambiente não serão requeridas. Este é o nível mais baixo em modo *prompt*.
  - **basic:** o usuário deve fornecer somente o valor das opções básicas, independente se elas têm um valor *default* ou não. Algumas opções são definidas como opções **nuisance** na biblioteca MLC^{++} e não serão requeridas nesse modo. O objetivo deste modo é solicitar o valor das opções mais usadas. Uma opção com um valor *default* da MLC^{++} pode ser mudada para uma opção **nuisance**, fixando o valor da opção com um ponto de exclamação "!". Uma opção **nuisance** pode ser mudada para uma opção **no-nuisance** fixando seu valor com um ponto de interrogação "?".
  - **all:** o usuário deve fornecer o valor de todas as opções.

Quando uma opção requer um valor entre um conjunto de valores, a opção digitada casará com o valor mais próximo da lista de opções. Por exemplo, digitando **setenv INDUTOR naive** casará com o naive-bayes (prefixo de casamento) e **setenv INDUTOR c4.5** casará com o C4.5.

Para facilitar a repetição de execuções de um programa com as mesmas opções, pode-se colocar as opções em um arquivo. O *dump* está em um formato que pode ser alimentado por *csh* ou *tcsh*. A opção *default* é não *dump*. Para fixar um arquivo *dump*, coloque a variável de ambiente **OPTION_DUMP** com o nome do novo arquivo. É recomendável fazer o seguinte no arquivo .login:

setenv OPTION_DUMP ~/.mlcoptions

Para repetir uma execução que fez um *dumping*, simplesmente reinicialize-a, por exemplo:

source ~ /.mlcoptions

O arquivo *dump* é gerado com as opções de entrada e serão mantidas, embora a execução seja interrompida.

As opções mostradas na Tabela 2 são aplicáveis a muitos utilitários MLC^{++}. Um hífen (-) denota que a opção deve ser fixada e não existe *default*.

Nome da opção	Domínio	Default	Explicação
INDUCER	Nome do indutor	-	O nome de um indutor deve acompanhar esta opção.
DATAFILE	Nome do arquivo de dados	-	Especifica o arquivo de dados para trabalhar. Esta opção geralmente é o arquivo base (sem nenhum sufixo) que implica usar .names para o arquivo de nomes, .data para o arquivo de dados e .test para o arquivo de teste.
NAMESFILE	Nome do arquivo de nomes	-	Indica o arquivo de nomes para ajudar na análise gramatical do arquivo de dados; deve usar o mesmo nome do arquivo base adicionando a extensão .names.
TESTFILE	Nome do arquivo de teste	-	Os utilitários que usam um arquivo de teste opcional usarão esta opção. Ele deve usar o mesmo nome do arquivo base de **DATAFILE** agregando a extensão .test, exceto se **DATAFILE** termina com .all, nos casos que o *default* for vazio (nenhum arquivo de teste).
DUMPSTEM	Nome do arquivo	-	O arquivo base para escrever saídas. Esta opção é usada em utilitários que geram arquivos. Ver o utilitário **conv**, por exemplo.
SEED	int	7258789	Indica o *default* do ponto de partida para o gerador de números aleatórios. O valor *default* é o número de telefone da sala onde a MLC^{++} foi desenvolvida.
REMOVE_ UNKNOWN_ INST	yes, no	no	Remove as instâncias desconhecidas dos arquivos de dados Quando eles são lidos.
CORRUPT_ UNKNOWN_ RATE	real r 0 <r< 1	0	Especifica a taxa (probabilidade) para que um atributo seja corrompido.
DISPGRAPH	yes, no	no	Exibe a árvore induzida ou gráfico usando *dotty*.

Tabela 2: Opções de configuração para os utilitários MLC^{++}.

2.4. Decomposição da Biblioteca MLC++

Nesta seção é descrita a decomposição principal da MLC++ dependendo dos diferentes tipos de funcionalidades (classes). A biblioteca MLC++ está dividida em quatro tipos de funções:

- **Classes de Suporte Geral:** estas classes provêem apoio para operações gerais, não específicas aos algoritmos de aprendizado de máquina, suportando classes para matrizes, listas encadeadas, cadeias de caracteres, números aleatórios e gráficos. Tenta-se usar o software de domínio público e educacional tanto quanto possível para esta parte da MLC++. Por exemplo, as manipulações gráficas são feitas usando *LEDA* (Library of Efficient Data Structures) escrito por Stefan Näeher [Näeher 92] e *dot* da AT&T [Gansner 93].
- **Classes Essenciais:** estas são as ferramentas básicas que são compartilhadas por muitos algoritmos em aprendizado de máquina supervisionado. Elas estão divididas em três tipos de funcionalidades:
  - **Input/Output:** são as classes para ler e escrever arquivos de dados.
  - **Geração e Conversões:** classes para gerar *datasets* artificiais, convertendo tipos de atributos entre formatos (por exemplo, codificação local {a,b,c,d} → {0001,0010,0100,1000}, codificação binária) e normalizando valores.
  - **Wrappers:** "envolvem" um algoritmo de indução para gerar precisões de conjuntos de testes, precisão da estimativa usando *cross-validation* para gerar curvas de aprendizado.

  A maioria dos trabalhos em MLC++ se concentram nessas classes.
- **Categorizadores:** os categorizadores são funções que representam instâncias para categorias, ou classes. Eles são as estruturas básicas geradas pelos algoritmos de indução. A biblioteca MLC++ provê os categorizadores mais comuns, como: *categorizador constante* (que retorna a mesma categoria independente da instância), *categorizador de atributo* (que usa somente um atributo para predizer a categoria), *categorizador de disparo*, *categorizador do nearest-neighbor* (vizinho mais próximo), *categorizadores de árvores e gráficos de decisão*, *categorizador de perceptron* e *categorizador de redes neurais*. Também são construídos *categorizadores recursivos*, por exemplo, em um *categorizador de árvore de decisão*, os nós ramificados são categorizados entre eles mesmos (mapeando um conjunto de instâncias no conjunto de filhos daquele nó), e o algoritmo de indução pode usar qualquer categorizador, inclusive a possibilidade de árvores de decisão recursivas. O ID3 [Quinlan 86] sempre usa *categorizadores de atributos* para atributos nominais e categorizadores de disparo para atributos reais. Para gerar árvores multivariantes [Brodley 92] [Ileath 93] com perceptrons como nós, o algoritmo de indução pode colocar *categorizadores de perceptron* nos nós. No futuro, os criadores da MLC++ proverão de **mapeadores**, uma generalização de categorizadores onde a saída será um número real em vez de um rótulo de categoria discreta.
- **Algoritmos de Indução:** são os algoritmos de indução (indutores) capazes de gerar regras de produção ou árvores de decisão. A MLC++ atualmente provê e suporta a maioria dos algoritmos de indução.

Na Figura 1 é ilustrada a interação que existe entre indutores, categorizadores, verificadores e *wrappers*. A parte superior ilustra a indução e subsequentes testes de um categorizador. A parte inferior denota um *wrapper cross-validation* estimando a aproximação de um indutor sobre um *dataset*. A separação do indutor e do categorizador é extremamente importante no aprendizado supervisionado e representa um papel importante na decomposição global da biblioteca MLC++.

Na próxima seção será apresentado o formato que deve ter qualquer arquivo de dados utilizado pela MLC++, baseado no padrão C4.5 de Quinlan [Quinlan 93].

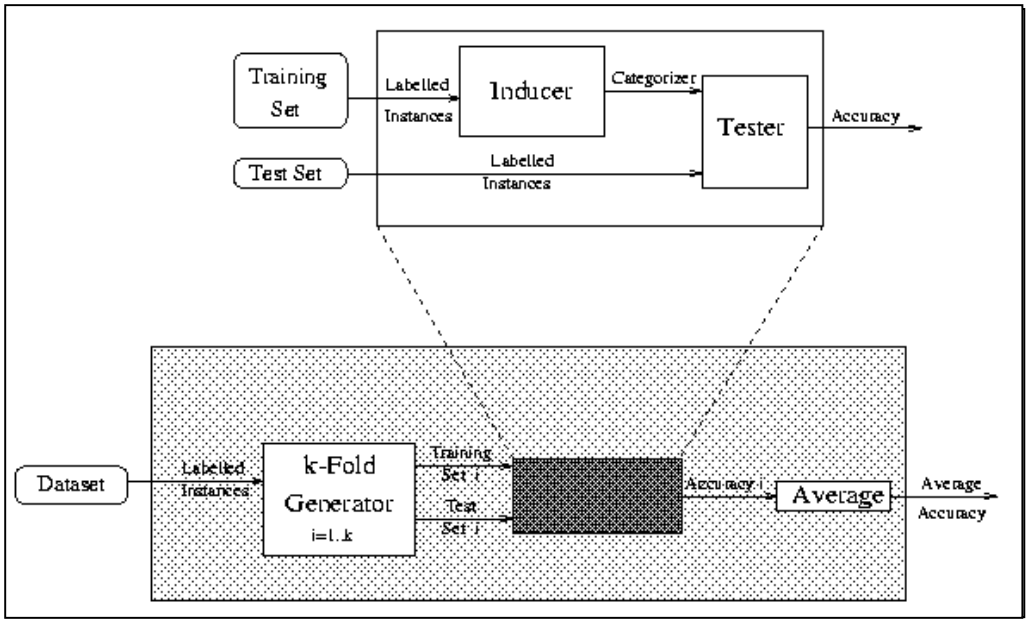

Figura 1: Interações entre indutores, categorizadores, verificadores e *wrappers*.

3. FORMATO DOS ARQUIVOS DE DADOS

Nesta seção são descritos os formatos dos arquivos *dataset* que a MLC^{++} reconhece por *default*. Os formatos dos arquivos são baseados no formato C4.5 de Quinlan [Quinlan 93], que compartilha o mesmo formato de arquivos de dados com a base de dados de Irvine [Murphy 95]. Nota-se que a maioria dos *datasets* de Irvine não incluem o arquivo de nomes que define o *schema* do *dataset* (por exemplo, o nome dos atributos de domínios).

O arquivo de dados e o arquivo de teste contêm instâncias rotuladas, uma instância por linha. O arquivo de nomes define o *schema* que permite analisar gramaticalmente o arquivo de dados. As funções da MLC^{++} que lêem *datasets* requerem só um *filestem* (arquivo base) e levam o sufixo apropriado de .data, .test ou .names correspondendo aos arquivos de dados, teste e nomes respectivamente.

Para este trabalho, foi utilizada na maioria dos testes uma base de dados referente à aprovação de cartão de crédito. Essa base foi escolhida devido a suas propriedades: 690 registros, 15 atributos, apenas 5% de dados faltantes, 2 classes, entre outras. Foram escolhidos 490 registros para criar o conjunto de treinamento e 200 registros para gerar o conjunto de teste. No exemplo 2 é apresentada a descrição do *dataset* crx localizado no arquivo /usr/mlclib/mlc/db/crx.doc.

Exemplo 2:

> This file concerns credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.
>
> *Title:*
> Credit Approval
> *Sources:*
> (confidential)
> Submitted by quinlan@cs.su.oz.au

Past Usage:

See Quinlan,

*"Simplifying decision trees", Int J Man-Machine Studies 27,
Dec 1987, pp. 221-234.

"C4.5: Programs for Machine Learning", Morgan Kaufmann, Oct 1992.

Relevant Information:

This dataset is interesting because there is a good mix of attributes -- continuous, discrete with small numbers of values, and discrete with larger numbers of values. There are also a few missing values.

Instances:	Attributes:
690	15

Missing Attribute Values:

37 cases (5%) have one or more missing values. The missing values from particular attributes are:

A1: 12
A2: 12
A4: 6
A5: 6
A6: 9
A7: 9
A14: 13

Class Distribution:

+: 307 (44.5%)
-: 383 (55.5%)

yes,no. -classes

A1: b, a. type: A
A2: continuous. type: B
A3: continuous. type: B
A4: u, y, l, t. type: C
A5: g, p, gg. type: D
A6: c, d, cc, i, j, k, m, r, q, W, x, e, aa, ff. type: E
A7: v, h, bb, j, n, z, dd, ff, o. type: E
A8: continuous. type: B
A9: t, f. type: F
A10: t, f. type: F
A11: continuous. type: B
A12: t, f. type: F
A13: g, p, s. type: G
A14: continuous. type: H
A15: continuous. type: H

3.1. Arquivo de Nomes

O arquivo de nomes descreve o *schema* de um *dataset* através do nome e o domínio para cada atributo e rótulo. A extensão *default* do arquivo de nomes é .names.

A barra vertical (|) indica um comentário e o que está a sua direita é ignorado. Podem ser usadas linhas em branco, espaços e tabulações livremente para tornar a leitura mais fácil. Quando eles

aparecem como parte de um nome, múltiplos espaços brancos (espaços, tabulações) são compactados em um único espaço.

O arquivo de nomes começa com uma definição de classes seguida por definições de atributos e domínios. A definição de classes mostra as diferentes classes existentes e terminam com um ponto, a definição de atributos consiste do nome de atributo seguido por dois pontos. Uma definição de domínio usa a palavra reservada "continuous" ou uma lista de valores de domínio separados por vírgula.

O valor do domínio e do nome do atributo são cadeias que não devem conter os caracteres: vírgula (,), dois pontos (:), barra (/), barra vertical (|), ou ponto de interrogação (?) (esse caracter é permitido somente sob condições especiais). Vale ressaltar que espaços são caracteres válidos em cadeia de caracteres, mas vários caracteres em brancos são compactados em um único espaço.

No Exemplo 3 é mostrado o arquivo de nomes /usr/mlclib/mlc/db/crx.names para o dataset crx.

Exemplo 3:

```
| This file concerns credit card applications.  All attribute names and values have been
| changed
| to meaningless symbols to protect confidentiality of the data.
| 7/16/95: Changed w to W and +,- in classes to y,n in order allow for CN2's limited
| language
| (ingest.l)  It turns out that w is a special token, and all nominal vals must start with
| letter.

yes, no.  | classes

A1:     b,a.
A2:     continuous.
A3:     continuous.
A4:     u, y, l, t.
A5:     g, p, gg.
A6:     c, d, cc, i, j, k, m, r, q, W, x, e, aa, ff.
A7:     v, h, bb, j, n, z, dd, ff, o.
A8:     continuous.
A9:     t,f.
A10:    t,f.
A11:    continuous.
A12:    t,f.
A13:    g, p, s.
A14:    continuous.
A15:    continuous.
```

3.2. Arquivo de Dados

O arquivo de dados descreve as instâncias do *dataset* que são usadas para treinar o algoritmo. Esses dados são geralmente chamados de conjunto de treinamento. A extensão *default* do arquivo de dados é .data.

A barra vertical (|) indica um comentário e o que está a sua direita é ignorado. Podem ser usadas linhas em branco, espaços e tabulações livremente para tornar a leitura mais fácil.

Cada linha do arquivo de dados descreve uma instância. Cada instância é composta de valores de atributo separados por vírgula e termina com um ponto. Os valores dos atributos devem pertencer aos domínios correspondentes dos atributos dentro do arquivo de nomes. No Exemplo 4 é mostrado um subconjunto do arquivo de dados /usr/mlclib/mlc/db/crx.data que contém 490 registros.

Exemplo 4:

```
b,30.83,0,u,g,W,v,1.25,t,t,01,f,g,00202,0,yes
a,58.67,4.46,u,g,q,h,3.04,t,t,06,f,g,00043,560,yes
a,24.50,0.5,u,g,q,h,1.5,t,f,0,f,g,00280,824,yes
b,27.83,1.54,u,g,W,v,3.75,t,t,05,t,g,00100,3,yes
b,20.17,5.625,u,g,W,v,1.71,t,f,0,f,s,00120,0,yes
b,32.08,4,u,g,m,v,2.5,t,f,0,t,g,00360,0,yes
b,33.17,1.04,u,g,r,h,6.5,t,f,0,t,g,00164,31285,yes
a,22.92,11.585,u,g,cc,v,0.04,t,f,0,f,g,00080,1349,yes
b,54.42,0.5,y,p,k,h,3.96,t,f,0,f,g,00180,314,yes
b,42.50,4.915,y,p,W,v,3.165,t,f,0,t,g,00052,1442,yes
b,22.08,0.83,u,g,c,h,2.165,f,f,0,t,g,00128,0,yes
b,29.92,1.835,u,g,c,h,4.335,t,f,0,f,g,00260,200,yes
a,38.25,6,u,g,k,v,1,t,f,0,t,g,00000,0,yes
b,48.08,6.04,u,g,k,v,0.04,f,f,0,f,g,00000,2690,yes
a,45.83,10.5,u,g,q,v,5,t,t,07,t,g,00000,0,yes
```

3.3. Arquivo de Teste

O arquivo de teste descreve as instâncias do *dataset* que são usadas para testar o desempenho do algoritmo. Esses dados são normalmente chamados de conjunto teste. A extensão *default* do arquivo de teste é .test. O formato do arquivo de teste é o mesmo que o do arquivo de dados. No exemplo 5 é mostrado um subconjunto do arquivo de teste /usr/mlclib/mlc/db/crx.test que contém 200 registros.

Exemplo 5:

```
a,47.25,0.75,u,g,q,h,2.75,t,t,01,f,g,00333,892,yes
b,24.17,0.875,u,g,q,v,4.625,t,t,02,t,g,00520,2000,yes
b,39.25,9.5,u,g,m,v,6.5,t,t,14,f,g,00240,4607,yes
a,20.50,11.835,u,g,c,h,6,t,f,0,f,g,00340,0,yes
a,18.83,4.415,y,p,c,h,3,t,f,0,f,g,00240,0,yes
b,19.17,9.5,u,g,W,v,1.5,t,f,0,f,g,00120,2206,yes
a,25.00,0.875,u,g,x,h,1.04,t,f,0,t,g,00160,5860,yes
b,20.17,9.25,u,g,c,v,1.665,t,t,03,t,g,00040,28,yes
b,25.75,0.5,u,g,c,v,1.46,t,t,05,t,g,00312,0,yes
b,20.42,7,u,g,c,v,1.625,t,t,03,f,g,00200,1391,yes
b,?,4,u,g,x,v,5,t,t,03,t,g,00290,2279,yes
b,39.00,5,u,g,cc,v,3.5,t,t,10,t,g,00000,0,yes
a,64.08,0.165,u,g,ff,ff,0,t,t,01,f,g,00232,100,yes
b,28.25,5.125,u,g,x,v,4.75,t,t,02,f,g,00420,7,yes
a,28.75,3.75,u,g,c,v,1.085,t,t,01,t,g,00371,0,yes
```

A seguir são apresentados os utilitários da biblioteca MLC^{++}, que permitem aos usuários fazer cálculos estatísticos, medir o desempenho dos algoritmos, obter informações do *dataset*, entre outros.

4. UTILITÁRIOS DISPONÍVEIS

A biblioteca MLC^{++} provê ferramentas que podem auxiliar os usuários testarem algoritmos de aprendizado de máquina. Atualmente, a biblioteca inclui as seguintes ferramentas: estimativa de precisão, Inducer, info, biasVar, Categorize, LearnCurve, C45Tree, project, discretize, conv e Diagramas de Lógica Geral, que serão detalhadas nas próximas seções.

4.1. Estimativa de Precisão

A estimativa de precisão refere-se ao processo de calcular o desempenho futuro de um indutor em um determinado *dataset* [Kohavi 95a] [Weiss 91]. O utilitário **AccEst** calcula a estimativa de precisão sobre futuras instâncias através de diferentes métodos para calcular essas estimativas.

Em muitos casos onde a MLC^{++} apresenta uma estimativa de precisão, existe também a confiança do resultado. O número depois de ± indica o desvio padrão da estimativa de precisão. Se um único conjunto de teste está disponível, o desvio padrão é um cálculo teórico que é razoável para grandes conjuntos de teste e para precisões não tão próximas de 0 ou 1. Se métodos de *resamplings* são usados (*cross-validation* e *bootstrap*), então a média do desvio padrão de todas as amostras é determinada. A média do desvio padrão não é o desvio padrão da própria população. Isto mostra como a média é variável, que é menor que a variação da própria população devido ao número de amostras. Uma estimativa de precisão entre colchetes tem um limite de confiança de 95%, que é calculado por uma fórmula dada em [Kohavi 95a] e [Devijver 82]. Um estimativa de precisão entre parênteses é um intervalo percentual de 95% [Efron 93]; o limite percentual é pessimista, no sentido que inclui um amplo limite devido ao número integral de amostras. Menos de 40 amostras, as estimativas serão menores ou maiores, de forma que a pessoa possa ver a variação das estimativas.

Além de *holdout* (treinar/testar) que é o método mais comum de estimativa de precisão, a MLC^{++} suporta *cross-validation*, *cross-validation estratificada*, *bootstrap* .632 e geração de curvas de aprendizado, as opções para selecionar o método de estimativa de precisão estão definidas na Tabela 3.

Nome da opção	Domínio	Default	Explicação
ACC_ ESTIMATOR	cv, strat-cv, bootstrap, hold-out, test-set	cv	Define o método de estimativa de precisão a ser usado: *cross-validation, cross-validation estratificado, bootstrap .632, holdout* e *testset*. O *testset* "cheats" observa o conjunto de testes e pode ser usado para encontrar limites superiores no desempenho.
ACC_TRIM	real r $0 \le r < 0.5$	0.0	Define a relação de *cross-validation* para ajustar os valores extremos. Este pode estabilizar o procedimento porque pode haver *folds* (dobras) não representativos. Um valor razoável é 0.1. Um exemplo pode ser visto em [Rice 88], onde pode-se ver o método exato do cálculo da média e variância para dados ajustados.

Tabela 3: Opções da estimativa de precisão.

4.1.1. Holdout

Neste método o *dataset* é dividido em dois conjuntos disjuntos de instâncias. O indutor é treinado sob um conjunto, o conjunto de treinamento, e testado com um outro conjunto disjunto,

o conjunto de teste. O cálculo da estimativa de precisão é feita no conjunto teste. Na Tabela 4 são apresentadas as diferentes opções para este tipo de estimativa de precisão.

Nome da opção	Domínio	Default	Explicação
HO_TIMES	int ≥ 1	1	Especifica o número de vezes que o *holdout* será repetido. Se o *holdout* é repetido, normalmente é chamado de subprovas aleatórias.
HO_NUMBER	int	0	Indica o número de instâncias para o treinamento do indutor, deixando o resto para o conjunto de testes. Se o valor é zero, a opção **HO_PORCENT** é usada. Se o número é negativo, o valor absoluto especifica o número de instâncias deixadas no conjunto de teste, com o restante usado para treinamento.
HO_PORCENT	real r 0 ≤ r <1	2/3	Determina a porcentagem das instâncias fixada para o treinamento, deixando o resto para o conjunto de teste.

Tabela 4: Opções do **Holdout**.

4.1.2. Cross-validation

Cross-validation é uma classe usada para calcular a precisão de um indutor sobre conjuntos de teste não vistos. *Cross-validation* executa *k-folds* sobre um indutor em um determinado conjunto de treinamento. Divide o conjunto de treinamento em *k* subconjuntos de teste mutuamente exclusivos (os *folds*) de tamanho aproximadamente igual, então o indutor é treinado repetidamente em *k* -1 partes e a outra parte restante será usada para testes. A estimativa de precisão do *cross-validation* é a média das precisões calculadas dos *k-folds*. Na Tabela 5 são apresentadas as opções para este método estimador de precisão.

Nome da opção	Domínio	Default	Explicação
CV_FOLDS	int ≠ 0 e 1	10	Indica o número de *folds* a ser usado *no cross-validation.* Um número negativo *k* (-k) faz deixar fora ao *cross-validation.*
CV_TIMES	int ≥ 0	1	Especifica o número de vezes para repetir o procedimento de *cross-validation.* Números muito grandes reduzem a variância da estimativa. Zero (0) usa uma heurística que repete o *cross-validation* até a estimativa da variância cair a um desvio padrão limite (**DES_STD_DEV**) ou um número máximo de vezes ser alcançado (**MAX_TIMES**). Note que a estimativa de variância assume independência dos *folds* que é inexata quando *cross-validation* é executado muitas vezes, esta suposição de independência fica irreal quando aumenta o número de execuções. Dado tudo aquilo, isto trabalha bem na prática.

Tabela 5: Opções do **Cross-validation**.

Cross-validation usa a estimativa de precisão de cada teste para calcular a média da estimativa de precisão e a variância dos testes. Existe a opção de enviar todos os conjuntos de treinamento gerados durante o *cross-validation* a arquivos para utilizá-los posteriormente.

No exemplo 6 é mostrada uma execução de *cross-validation* em dois indutores utilizando o mesmo *dataset.*

Exemplo 6:

```
setenv DATAFILE crx.all          #".all" contém todos os dados
setenv INDUCER ID3               #algoritmo ID3
setenv ACC_ESTIMATOR cv
AccEst
```

A saída será:

```
10 folds: 1 2 3 4 5 6 7 8 9 10   #folds
Method: cv                       #método cross-validation
Trim: 0                          #ajuste dos valores extremos
Seed: 7258789                    #ponto de partida
Folds: 10, Times: 1
Accuracy: 72.50% +- 3.165% (78.22% -  65.932%)      #estimativa da precisão
```

Para fazer *cross-validation* com um outro indutor é neccesário apenas modificar o valor da variável de ambiente **INDUCER**.

```
setenv INDUCER IB                #algoritmo Instance-Based.
AccEst
```

A saída será:

```
10 folds: 1 2 3 4 5 6 7 8 9 10
Method: cv
Trim: 0
Seed: 7258789
Folds: 10, Times: 1
Accuracy: 77.50% +- 2.960%  (82.738% -  71.226%)
```

Fixando o **LOGLEVEL** para **1** é possível ver todas as opções disponíveis. Fixando o **PROMPTLEVEL** para **basic** ou **all** (o *default* é **requires-only**), a MLC^{++} solicitará o preenchimento de todas as opções. Digita-se **?** para qualquer tipo de ajuda.

4.1.3. Stratified Cross-validation

O *stratified cross-validation* é um método semelhante ao *cross-validation*, exceto que os *folds* são estratificados de tal forma que eles contenham aproximadamente as mesmas proporções de instâncias que o *dataset* original.

4.1.4. Bootstrap .632

No *bootstrap* .632 [Efron 93], a estimativa de precisão é calcula da seguinte forma: dado um *dataset* de tamanho *n*, uma amostra de *bootstrap* é criada para testar *n* instâncias uniformes a partir dos dados (com substituição). Considerando que o *dataset* é testado com substituição, a probabilidade de qualquer instância não ser escolhida depois de *n* amostras é $(1 - 1/n)^n \approx e^{-1} \approx 0.368$; o número esperado de instâncias distintas de um *dataset* original, aparecendo no conjunto de teste, é 0.632. A estimativa de precisão ε0 é derivada usando a amostra *bootstrap* para treinar e o resto das instâncias para testar. Dado um número *b*, o número de amostras *bootstrap* permite a $\varepsilon 0_i$ ser a estimativa de precisão para a amostra *bootstrap i*. A estimativa de *bootstrap* .632 está definida como:

$$acc_{boot} = \frac{1}{b}\sum_{i=1}^{b}\left(0.632\varepsilon 0_i + 0.368 acc_{\xi}\right)$$

onde acc_ξ é a estimativa de erro de substituição do *dataset* completo (por exemplo, o erro dado no treinamento). Na Tabela 6 são apresentadas as diferentes opções para este tipo de estimativa de precisão.

Nome da opção	Domínio	Default	Explicação
BS_TIMES	int ≥ 1	10	Especifica o número de amostras *bootstrap*.
BS_FRACTION	real r 0 ≤ r <1	0.632	Determina a fração *bootstrap* multiplicando as instâncias não vistas.

Tabela 6: Opções do **Bootstrap**.

4.2. Inducer

O utilitário **Inducer** executa um dos indutores escolhidos (explicados em detalhes na seção 5) em um determinado arquivo de dados e mostra as seguintes estatísticas:

- **Instance counts**: define o número de instâncias de treinamento, de instâncias de teste, de instâncias de teste não vistas e de instâncias vistas.
- **Classification counts**: mostra o número de classificações corretas e incorretas.
- **Generalization accurancy:** mostra a precisão das instâncias não vistas.
- **Memorization accurancy:** define a precisão sob as instâncias vistas.
- **Accurancy:** calcula a precisão global no conjunto de teste.

Uma grande discrepância entre a **Generalization accurancy** e a **Memorization accurancy** normalmente indica *overfitting*. Algumas opções são mostradas na Tabela 7:

Nome da opção	Domínio	Default	Explicação
DISP_ CONFUSION_ MAT	no, ASCII, scatterviz, both	no	Exibe uma matriz de classificação das classes corretas vs. as incorretas.
DISP_ MISCLASS	yes, no	no	Mostra as instâncias não classificadas.
DISPLAY_ STRUCT	none, ASCII, dot, dotty, treeviz	none	Exibe o classificador de indução. As saídas ASCII são exibidas na tela e as saídas *dot* vão para o arquivo Inducer.dot para indutores que suportam saídas dot (somente árvore de decisão e classificadores gráficos de decisão). O treeviz gera arquivos compatíveis com o *Tree visualizer*® do *MineSet*® da *Silicon Graphics*®. O *Tree visualizer*® permite caminhar pela árvore. A Figura 2 mostra uma fotografia do *dataset* iris.
INDUCER_ DOT	Nome do arquivo	Inducer. dot	Somente aparece si o **DISPLAY_STRUCT** for **dot**.
DIST_DISP	yes, no	no	Exibe a distribuição de nós. Esta opção só estará disponível quando for pertinente.
CAT_NAME	Nome do arquivo	-	Define o nome do categorizador persistente. A extensão .cat será adicionada. Use-se com o utilitário **Categorize**. Poucos indutores suportam persistência.

Tabela 7: Opções para o utilitário **Inducer**.

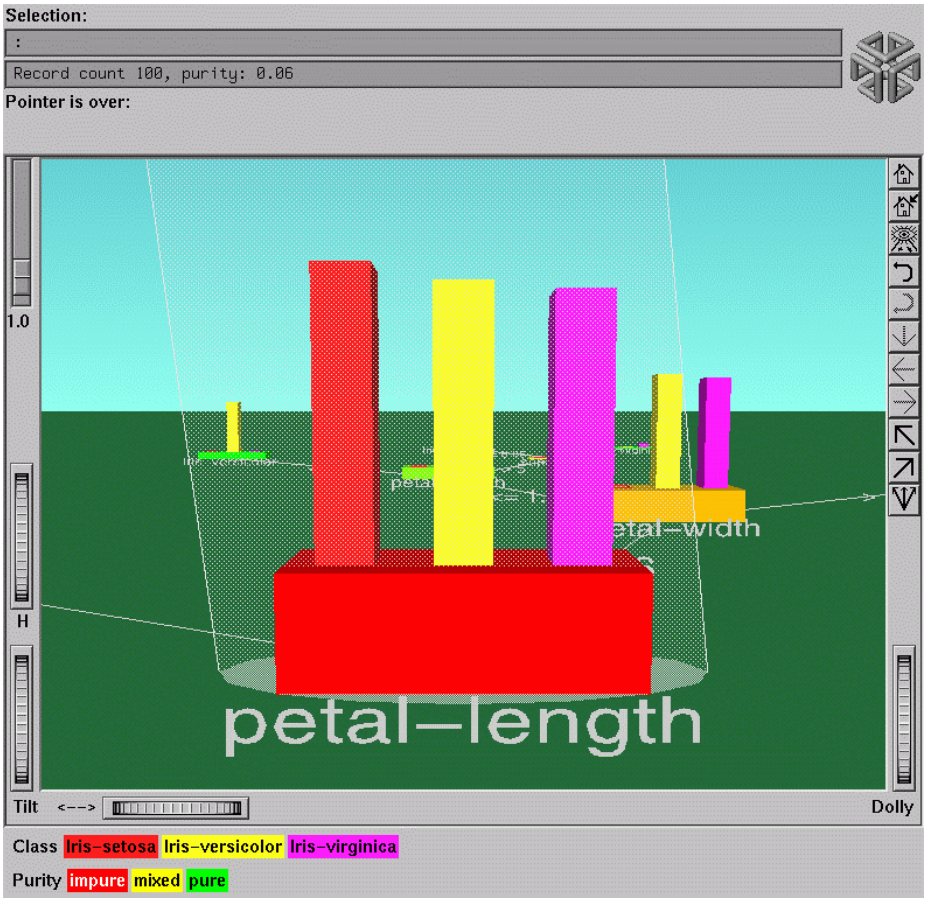

Figura 2: Uma árvore gerada através do Treeviz visualizer® do MineSet®.

4.3. Info

O utilitário **info** provê informação estatística básica sobre um *dataset*. Ele informa o número de instâncias nos arquivos .data, .test e .all (o .all é opcional e deverá conter os arquivos .data e .test para usar-se em AccEst). Informa também as probabilidades das classes, o número de atributos e seu tipo (contínuo ou nominal). Se a opção **SHOW_ATTR_INFO** for **yes**, então o número de valores para cada atributo será dado. Isto pode ajudar a localizar com precisão as declarações impróprias de atributos ou até mesmo atributos contínuos que têm poucos valores.

Converter atributos para o tipo nominal geralmente é sugerido para ganhar velocidade. Por exemplo, o tempo de execução para o C4.5 (excluindo a cargo da MLC^{++}) no *dataset* StatLog DNA [Taylor 94] é de 14 segundos em uma SGI Indy, se os atributos são declarados como contínuos e 4.7 segundos se eles são declarados como nominais. Diferenças de precisão secundárias podem aparecer devido a diferentes modos de manipulação de tais atributos.

A seguir, no Exemplo 7, é apresentada uma execução com o utilitário **info**, a fim de se obter os atributos do *dataset* crx:

Exemplo 7:

```
setenv DATAFILE crx
setenv SHOW_ATTR_INFO yes
info
```

A saída será:

```
OPTION DATAFILE = crx
OPTION NAMESFILE = crx.names
DataFile is crx.data. LOGLEVEL is 7
Hidden option MAX_ATTR_VALS = 100
Hidden option MAX_LABEL_VALS = 25
OPTION WEIGHT_IS_ATTRIBUTE = true
OPTION REMOVE_UNKNOWN_INST = false
OPTION CORRUPT_UNKNOWN_RATE = 0
Reading crx.data....Checking schema against names-file schema
Schema: attr 0 (A1).  Original: attr 0 (A1).
Schema: attr 1 (A2).  Original: attr 1 (A2).
Schema: attr 2 (A3).  Original: attr 2 (A3).
Schema: attr 3 (A4).  Original: attr 3 (A4).
Schema: attr 4 (A5).  Original: attr 4 (A5).
Schema: attr 5 (A6).  Original: attr 5 (A6).
Schema: attr 6 (A7).  Original: attr 6 (A7).
Schema: attr 7 (A8).  Original: attr 7 (A8).
Schema: attr 8 (A9).  Original: attr 8 (A9).
Schema: attr 9 (A10).  Original: attr 9 (A10).
Schema: attr 10 (A11).  Original: attr 10 (A11).
Schema: attr 11 (A12).  Original: attr 11 (A12).
Schema: attr 12 (A13).  Original: attr 12 (A13).
Schema: attr 13 (A14).  Original: attr 13 (A14).
Schema: attr 14 (A15).  Original: attr 14 (A15).
Schema: no more attrs.  Original: attr 15 (Label).
done.
Reading crx.test.. done.
Reading crx.all.....Checking schema against names-file schema
Schema: attr 0 (A1).  Original: attr 0 (A1).
Schema: attr 1 (A2).  Original: attr 1 (A2).
Schema: attr 2 (A3).  Original: attr 2 (A3).
Schema: attr 3 (A4).  Original: attr 3 (A4).
Schema: attr 4 (A5).  Original: attr 4 (A5).
Schema: attr 5 (A6).  Original: attr 5 (A6).
Schema: attr 6 (A7).  Original: attr 6 (A7).
Schema: attr 7 (A8).  Original: attr 7 (A8).
Schema: attr 8 (A9).  Original: attr 8 (A9).
Schema: attr 9 (A10).  Original: attr 9 (A10).
Schema: attr 10 (A11).  Original: attr 10 (A11).
Schema: attr 11 (A12).  Original: attr 11 (A12).
Schema: attr 12 (A13).  Original: attr 12 (A13).
Schema: attr 13 (A14).  Original: attr 13 (A14).
Schema: attr 14 (A15).  Original: attr 14 (A15).
Schema: no more attrs.  Original: attr 15 (Label).
Data + Test == All
Number of instances in crx.all = 690
Duplicate or conflicting instances : 0
Number of instances in crx.data = 490
Duplicate or conflicting instances : 0
Number of instances in crx.test = 200
Duplicate or conflicting instances : 0
Class probabilities for crx.all file
Probability for the label 'yes' : 44.49%
```

Probability for the label 'no' : 55.51%
Majority error: 44.49% on value no
Number of attributes = 15 (continuous : 6 nominal : 9)
OPTION SHOW_ATTR_INFO = yes

Information about .all file :
3 distinct values for attribute #0 (A1) nominal
349 distinct values for attribute #1 (A2) continuous
215 distinct values for attribute #2 (A3) continuous
4 distinct values for attribute #3 (A4) nominal
4 distinct values for attribute #4 (A5) nominal
15 distinct values for attribute #5 (A6) nominal
10 distinct values for attribute #6 (A7) nominal
132 distinct values for attribute #7 (A8) continuous
2 distinct values for attribute #8 (A9) nominal
2 distinct values for attribute #9 (A10) nominal
23 distinct values for attribute #10 (A11) continuous
2 distinct values for attribute #11 (A12) nominal
3 distinct values for attribute #12 (A13) nominal
170 distinct values for attribute #13 (A14) continuous
240 distinct values for attribute #14 (A15) continuous

4.4. Bias-Variance

O **biasVar** fornece uma decomposição de Bias-Variance descrito em [Kohavi 96a].

As diferentes opções do **biasVar** são mostradas na Tabela 8.

Nome da opção	Domínio	Default	Explicação
NUM_TEST_ INST	int	0	Determina o número de instâncias de teste ou seleciona zero para uma fração de teste.
TESTE_FRACT	real	0.6667	Determina a fração de instâncias para reservar como conjunto de teste.
TREM_ INTERNO_ FRACT	real	0.5	Especifica a divisão interna do conjunto de treinamento.
TRAIN_TIMES	int	10	Determina o número de réplicas a serem formadas.
REPEAT_TIMES	int	1	Determina o número de vezes para repetir os conjuntos de réplicas.

Tabela 8: Opções para o utilitário **biasVar.**

4.5. Categorize

O utilitário **Categorize** provê um modo para categorizar novos registros sem etiquetas usando um categorizador persistente.

Na Tabela 9 são apresentadas as diferentes opções para o utilitário **Categorize**.

Nome da opção	Domínio	Default	Explicação
CAT_NAME	Nome do arquivo	-	Define o nome do arquivo que contém o categorizador persistente. A extensão .cat será adicionada. Gerado pelo utilitário **Inducer**.

Nome da opção	Domínio	Default	Explicação
DUMPSTEM	Nome do arquivo	-	Define o nome do arquivo que vai conter as etiquetas previsíveis, uma por linha.

Tabela 9: Opções para o utilitário **Categorize.**

4.6. LearnCurve

Learning Curve é uma classe que recebe um indutor e um conjunto de treinamento para gerar uma curva de aprendizado que mostre como a quantidade de dados de treinamento afeta a precisão do categorizador induzido. O indutor é treinado usando um certo número de instâncias escolhidas randomicamente de um dado conjunto, e treinado várias vezes naquele nível. A média da estimativa de precisão que prediz o conjunto total naquele nível é calculada usando todas as classes resultantes de cada teste. Os dados podem ser exibidos graficamente usando-se qualquer pacote gráfico, como *gnuplot* ou *Mathematica*. Este cálculo é feito através deste utilitário.

O utilitário **LearnCurve** gera uma curva de aprendizado para um determinado algoritmo de indução e um *dataset*. Dado um *dataset*, o eixo X representa o número de instâncias de treinamento e o eixo Y representa a precisão quando está sendo treinado sobre um determinado número de instâncias e testado nas instâncias não vistas.

Nome da opção	Domínio	Default	Explicação
NUM_ INTERVALS	int > 0	10	Indica o número de intervalos no eixo X. Os pontos das amostras são distribuídos quase uniformemente.
NUM_REPEATS	int > 0	5	Determina o número de vezes a ser executado o algoritmo de indução para cada ponto da amostra.
MIN_TEST_SIZE	int ≥ 0	0	Especifica o número mínimo de instâncias do *dataset* a ser armazenado no teste. Se for maior que zero, então este número será fixado como sendo o número de instâncias dividido pelo número de intervalos.
SEED	int	7258789	Determina o ponto de partida para criar amostras aleatórias.
DUMPSTEM	Nome do arquivo	-	Determina o arquivo para fazer um *dumping* no conjunto de treinamento criado. O conjunto de teste sempre é o arquivo completo. Isto permite comparar resultados com outros algoritmos de indução fora da MLC^{++}.
LC_OUT_TYPE	none, mathematica, gnuplot	none	Manda as saídas de dados para o aplicativo *mathematica* ou *gnuplot*. Para o mathematica, o arquivo LearnCurve.m pode ser usado com a saída gerada.

Tabela 10: Opções para o utilitário **LearnCurve.**

No exemplo 8 é mostrada uma execução de Learning Curve no indutor C4.5 sobre o *dataset* crx.

Exemplo 8:

```
setenv INDUCER c4.5
setenv DATAFILE crx.all                 #contém o dataset completo
setenv NUM_INTERVALS 20                 #número de intervalos no eixo X
setenv NUM_REPEATS 10                   #número de execuções para cada ponto
setenv MIN_TEST_SIZE 200                #deixar 200, pelo menos para testar
setenv DUMPSTEM                         #nenhum arquivo para fazer dumping
setenv LC_OUTPUT_TYPE gnuplot
```

LearnCurve
gnuplot crx.gnuplot

A saída será:

```
Inducer: c4.5.  Intervals: 20, Repeats: 10, Min test size: 200.  Seed: 7258789
DATAFILE: crx.all (size=690)
Inducer is c4.5
Train Size: 26; training 10 timesOPTION C45_RETRIES = 0
OPTION C45_STATS = false

C45Inducer::train_and_test: Error: 14.61% +- 1.37% [12.13% - 17.50%]
.[14.61]C45Inducer::train_and_test: Error: 15.81% +- 1.42% [13.24% - 18.78%]
.[15.81]C45Inducer::train_and_test: Error: 18.52% +- 1.51% [15.75% - 21.66%]
.[18.52]C45Inducer::train_and_test: Error: 16.27% +- 1.43% [13.65% - 19.26%]
.[16.27]C45Inducer::train_and_test: Error: 25.75% +- 1.70% [22.57% - 29.21%]
.[25.75]C45Inducer::train_and_test: Error: 18.07% +- 1.49% [15.33% - 21.18%]
.[18.07]C45Inducer::train_and_test: Error: 17.77% +- 1.48% [15.05% - 20.86%]
.[17.77]C45Inducer::train_and_test: Error: 26.66% +- 1.72% [23.43% - 30.15%]
.[26.66]C45Inducer::train_and_test: Error: 28.31% +- 1.75% [25.02% - 31.86%]
.[28.31]C45Inducer::train_and_test: Error: 26.96% +- 1.72% [23.72% - 30.46%]
.[26.96] Mean Error: 20.87% +- 1.70%
                                        ..........................................
Train Size: 464; training 10 timesC45Inducer::train_and_test: Error: 11.50% +-
  2.13% [7.97% - 16.32%]
.[11.50]C45Inducer::train_and_test: Error: 13.72% +- 2.29% [9.83% - 18.81%]
.[13.72]C45Inducer::train_and_test: Error: 17.70% +- 2.54% [13.28% - 23.20%]
.[17.70]C45Inducer::train_and_test: Error: 12.39% +- 2.20% [8.71% - 17.32%]
.[12.39]C45Inducer::train_and_test: Error: 13.72% +- 2.29% [9.83% - 18.81%]
.[13.72]C45Inducer::train_and_test: Error: 15.49% +- 2.41% [11.35% - 20.78%]
.[15.49]C45Inducer::train_and_test: Error: 13.27% +- 2.26% [9.46% - 18.32%]
.[13.27]C45Inducer::train_and_test: Error: 18.14% +- 2.57% [13.66% - 23.68%]
.[18.14]C45Inducer::train_and_test: Error: 15.49% +- 2.41% [11.35% - 20.78%]
.[15.49]C45Inducer::train_and_test: Error: 15.04% +- 2.38% [10.97% - 20.29%]
.[15.04] Mean Error: 14.65% +- 0.68%
Train Size: 490; training 10 timesC45Inducer::train_and_test: Error: 19.00% +-
  2.78% [14.17% - 25.00%]
.[19.00]C45Inducer::train_and_test: Error: 14.50% +- 2.50% [10.29% - 20.05%]
.[14.50]C45Inducer::train_and_test: Error: 12.50% +- 2.34% [8.61% - 17.80%]
.[12.50]C45Inducer::train_and_test: Error: 17.50% +- 2.69% [12.86% - 23.36%]
.[17.50]C45Inducer::train_and_test: Error: 15.50% +- 2.57% [11.14% - 21.16%]
.[15.50]C45Inducer::train_and_test: Error: 15.50% +- 2.57% [11.14% - 21.16%]
.[15.50]C45Inducer::train_and_test: Error: 17.50% +- 2.69% [12.86% - 23.36%]
.[17.50]C45Inducer::train_and_test: Error: 15.00% +- 2.53% [10.71% - 20.61%]
.[15.00]C45Inducer::train_and_test: Error: 12.00% +- 2.30% [8.20% - 17.23%]
.[12.00]C45Inducer::train_and_test: Error: 19.00% +- 2.78% [14.17% - 25.00%]
.[19.00] Mean Error: 15.80% +- 0.78%
Done.
Size,  Mean Error, std-dev of mean
26, 20.8734939759% +- 1.69544991388%
52, 17.2884012539% +- 0.889742945324%
77, 17.145187602% +- 1.35452075527%
103, 15.8773424191% +- 0.683698668633%
129, 15.8645276292% +- 0.861044369946%
155, 16.1308411215% +- 0.804703711556%
181, 16.4833005894% +- 0.480745308506%
```

206, 14.8966942149% +- 0.272800307443%
232, 15.3056768559% +- 0.299108043494%
258, 15.162037037% +- 0.329629774125%
284, 14.8522167488% +- 0.39039387284%
309, 15.3805774278% +- 0.443009235387%
335, 14.7323943662% +- 0.272300469484%
361, 15.3495440729% +- 0.613366195245%
387, 15.8415841584% +- 0.746130911235%
413, 15.4512635379% +- 0.896974495961%
438, 14.1666666667% +- 0.383597883598%
464, 14.6460176991% +- 0.680547020357%
490, 15.8% +- 0.775313556641%
Average nodes for all C4.5 trees:17.6105263158.
Gnuplot output in crx.gnuplot

A saída gerada é mostrada na Figura 3.

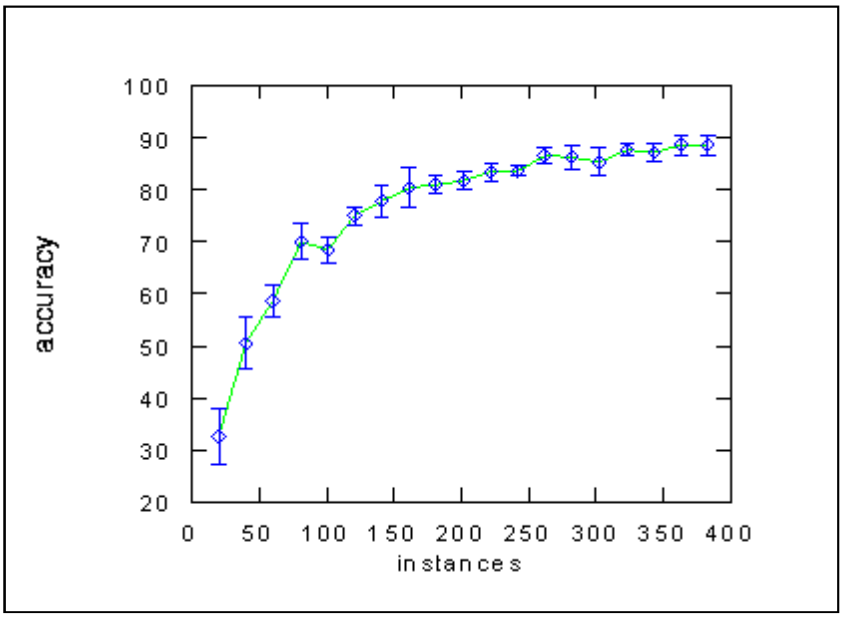

Figura 3: Gráfico gnuplot gerado por **LearnCurve**.

4.7. C45Tree

O utilitário **C45Tree** executa o algoritmo C4.5 e gera arquivos baseados no **DATAFILE** com sufixos unpruned.dot e pruned.dot. Estes podem ser vistos usando os visualizadores *treeviz®*, *dot* ou *dotty*. Se a opção **DISPGRAPH** for **yes**, os gráficos aparecerão usando *dotty*.

Limitações: **DIST_DISP** pode não ser fixada como **yes** e os subconjuntos divididos ("-s" em C4.5) não estão implementados.

4.8. Project

O utilitário **project** permite projetar um *dataset* em um espaço contendo somente um subconjunto de atributos (o resto dos atributos são descartados). Este conjunto de atributos para projetar-se é consultado quando o utilitário é executado. Os arquivos de nomes, de dados e de teste são convertidos para o espaço projetado.

O exemplo 9, explica uma execução usando **Feature Subset Selection**.

Exemplo 9:

Usando o Feature Subset Selection no indutor Table-majority descobriu-se que das 180 características utilizadas no *dataset DNA splice-juntion* usado no projeto StatLog [Taylor 94], um subconjunto pequeno de 11 características eram relevantes para Table-majority [Kohavi 97a]. Para gerar este subconjunto, projeto-se sobre as características numeradas 81, 83, 84, 89, 92, 93, 94, 95, 96, 101, 104. O desempenho de Table-majority é de 94.6% (sobre um conjunto de teste independente), contra 92.7% para o C4.5, quando são dados todos os atributos.

4.9. Discretize

O utilitário **discretize** provê habilidades de discretização. Note que a discretização deve ser feita usando somente o conjunto de treinamento.

Este utilitário só considera o conjunto de treinamento para formar os intervalos, e estes intervalos são usados para discretizar tanto o conjunto treinado quanto o conjunto de teste. É um erro discretizar todos os dados e depois executar *cross-validation*, porque os intervalos de discretização serão escolhidos baseados nos *folds* internos que servem como conjuntos de teste. O disc-filter da MLC^{++} fará a coisa certa se usado no *cross-validation*; por exemplo, para cada um dos *folds* de *cross-validation*, intervalos diferentes serão formados como se estes fossem conjuntos de treinamento e de teste.

4.10. Conv

O utilitário **conv** provê conversões simples dos dados para algoritmos que não trabalham bem com atributos categóricos ou que requeiram um formato de entrada ligeiramente diferente. São fornecidas duas codificações para os atributos nominais:

- **Codificação local:** cada valor de um atributo categórico é feito dentro de um atributo indicador. Para um determinado valor no arquivo de dados, o atributo indicador apropriado é fixado como 1, e todos os outros atributos indicadores que compartilham a representação são fixados como 0. Um valor desconhecido torna todos os atributos indicadores iguais a zero.
- **Codificação binária:** uma variável categórica com k possíveis valores é fixada dentro de $[\log_2 (k + 1)]$ bits. O valor i é mapeado dentro da representação binária de $i + 1$, e o zero binário é designado para valores desconhecidos.

Na Tabela 11 são apresentadas as diferentes opções do utilitário **conv**.

Nome da opção	Domínio	Default	Explicação
CONVERSION	local, binary, none, aha	local	A opção **aha** converte para o formato do programa IBL de David Aha [Aha 92].
ATTR_DELIM	space, comma, period, semicolon, colon	comma	Define os atributos delimitadores.
LAST_ATTR_ DELIM	Qualquer caracter	comma	Especifica o último delimitador antes da etiqueta.
END_OFLINE_ DELIM	Qualquer caracter	period	Delimita o fim do marcador de linha.

Nome da opção	Domínio	Default	Explicação
NORMALIZATION	none, NormalDist	none	NormalDist normaliza os atributos contínuos para ter média 0 e variância 1. Se existir somente um elemento, é assumido uma variância igual a 1. A variância 0 é modificada para 0.01.

Tabela 11: Opções para o utilitário **conv**.

4.11. Diagramas de Lógica Geral

Os Diagramas de Lógica Geral (**GLDs**) são projeções gráficas de espaços discretos multi-dimensionais sobre duas dimensões. Eles são semelhantes aos mapas de Karnaugh, mas são generalizados para entradas e saídas não booleanas. Um **GLD** pode ser gerado para um conjunto de teste mostrando as predições do categorizador induzido para o espaço completo de instâncias e uma revisão mostrando os erros. Os **GLDs** podem ser exibidos em um terminal X, Postscript, Xfig e terminal X de edição gráfica.

Um **GLD** provê um modo de exibir até aproximadamente dez dimensões em uma representação gráfica que pode ser entendida por esses humanos. Foram descritos **GLDs** em [Michalski 78] e depois usados em [Wnek 90]. Eles foram usados também em [Thrun 91] e em [Wnek 94] para comparar algoritmos. Os **GLDs** têm uma história longa e têm sido redescobertos várias vezes. Eles, às vezes, são chamados de *Dimensional Stacking* (Empilhado Dimensional) [LeBlank 90]. Os **GLDs** não trabalham com os indutores base (ver apêndice A).

Cada possível instância no espaço define exatamente uma caixa no **GLD**, que tem as seguintes opções de exibição (**GLD_SET**):

- **Test:** mostra as instâncias do conjunto de teste com suas classes.
- **Overlay:** mostra as instâncias do conjunto de teste. A exibição mostra previsões corretas ou incorretas do categorizador.
- **Predicted Train:** mostra o espaço previsível completo e reveste as classes do conjunto treinado. É necessário um monitor que suporte cores e escala de cinzas.
- **Predicted Test:** mostra o espaço previsível completo e reveste as instâncias do conjunto de teste assinalando os erros com um *X*. É necessário um monitor que suporte cores e escala de cinzas.

A saída do utilitário **GLD** pode ser tanto em um terminal X quanto em um arquivo no formato Xfig. A principal vantagem da saída Xfig é que podem ser adicionados nomes de atributo e outros comentários, e então inseridos em um documento. Existem atualmente oito cores que se repetem após um ciclo, caso haja mais classes. Na Tabela 12 são mostradas as possíveis opções.

Nome da opção	Domínio	Default	Explicação
GLD_MANAGER	Motif, Xfig	Motif	Exibe a saída para um terminal X ou gera saídas Xfig em **GLD**. Ver Figura 4.
GLD_SET	test, overlay, train, predicted-test	predicted-test	Define a possível instância no espaço que será mostrado no **GLD**.
GLD_MANUAL_ ORDER	yes, no	no	Restringe o indutor a um determinado conjunto de atributos com especificações manuais ordenadas sobre os eixos.
GLD_FONT	font	9x15	Define a fonte a ser usada se o gerenciador for Motif. Com menor fonte, o maior valor do atributo será ajustado dentro das caixas designadas.

Nome da opção	Domínio	Default	Explicação
GLD_HORIZONTAL_ MARGIN	int	300	Define quanto espaço (pixels) é deixado do lado esquerdo do diagrama para as etiquetas.
GLD_VERTICAL_ MARGIN	int	100	Define quanto espaço (pixels) é deixado abaixo do diagrama para as etiquetas.
GLD_HORIZONTAL_ PIXELS	int	400	Define o tamanho horizontal do diagrama em pixels, excluindo as margens.
GLD_ VERTICAL_ PIXELS	int	400	Define o tamanho vertical do diagrama em pixels, excluindo as margens.
GLD_COLOR_ DISPLAY	yes, no	yes	Modelos diferentes têm cores diferentes. Para combinações Predicted-Train/Test, é necessário um monitor colorido.
GLD_COLOR_ FILL	yes, no	yes	Usa modelos ou preenche a caixa com uma cor. Somente pertinente se **COLOR_DISPLAY** é **yes**.
GLD_OUTFILE	Nome do arquivo	.fig	Especifica o nome de arquivo para saída se o gerenciador de exibição for Xfig.
GLD_MAX_LINE_ WIDHT	int	4	A espessura das linhas no **GLD** está determinada pela posição da linha. A linha 2^i tem uma largura *i*, a menos que *i* seja maior que este parâmetro.
GLD_ALT_COLOR_ SCHEME	yes, no	no	Ativa um esquema de coloração alternativo que, às vezes, trabalha melhor.

Tabela 12: Opções para o utilitário **GLD**.

No exemplo 10, são mostrados quatro conjuntos diferentes de Diagramas de Lógica Geral.

Exemplo 10:

```
setenv INDUCER ID3
setenv DATAFILE monk1
setenv GLD_MANAGER xfig
setenv GLD_OUTFILE monk1-GLD-test.fig
setenv GLD_SET test
GLD
setenv GLD_OUTFILE monk1-GLD-overlay.fig
setenv GLD_SET overlay
GLD
setenv GLD_OUTFILE monk1-GLD-predTrain.fig
setenv GLD_SET predictedTrain
GLD
setenv GLD_OUTFILE monk1-GLD-predTest.fig
setenv GLD_SET predictedTest
GLD
```

A saída será:

```
Generating GLD for monk1.data
Classifying (% done): 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% done.
Generating GLD for monk1.data
Classifying (% done): 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% done.
Generating GLD for monk1.data
Classifying (% done): 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% done.
Generating GLD for monk1.data
Classifying (% done): 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% done.
```

Na Figura 4 e na Figura 5 são mostradas as saídas para os quatro conjuntos diferentes de **GLDs**. As possíveis instâncias no espaço estão definidas como: Set=test, Set=overlay, Set=predictedTrain e Set=predictedTest.

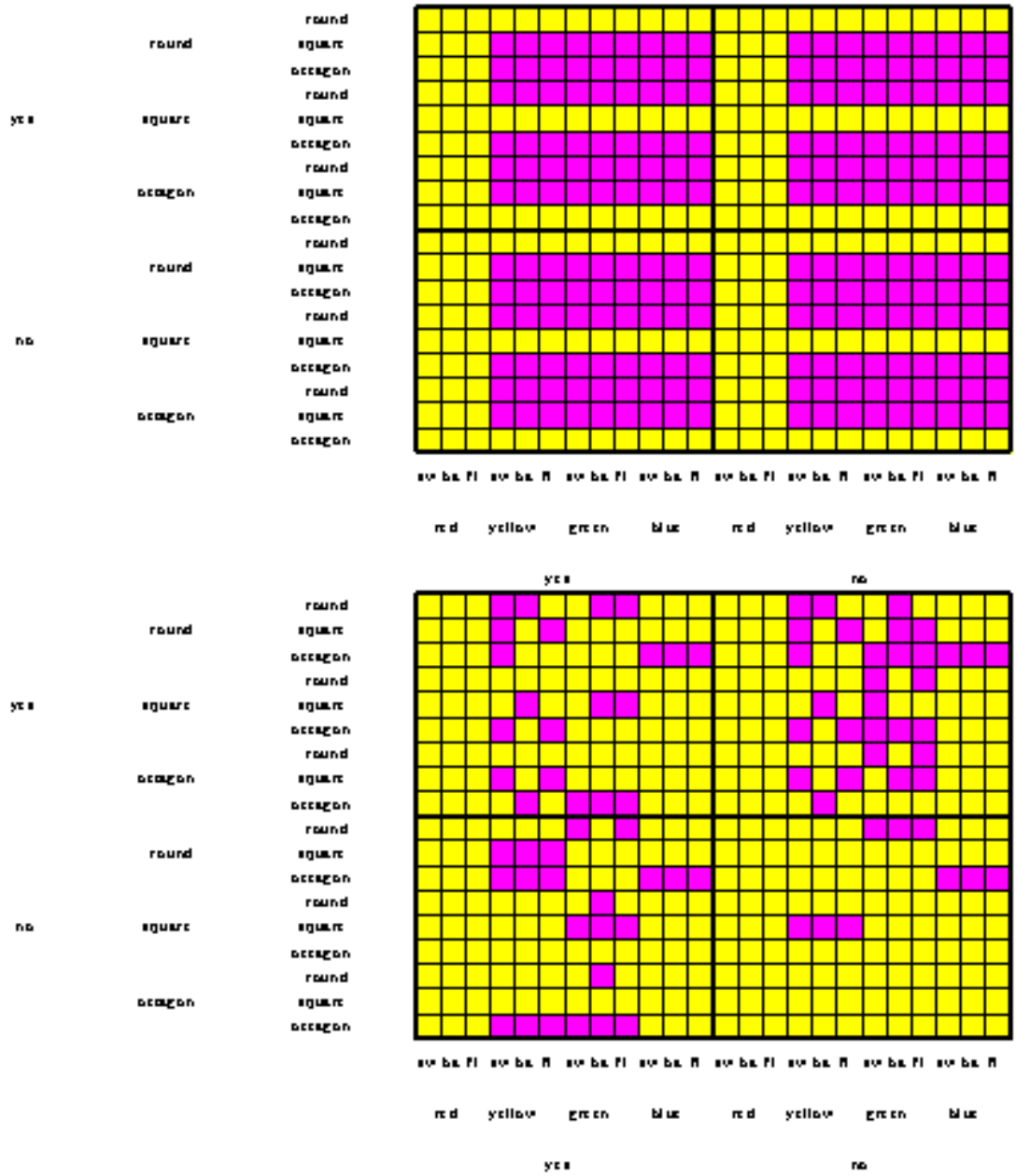

Figura 4: **GLD** para o *dataset* monk/ID3. Set=test acima e Set=overlay embaixo.

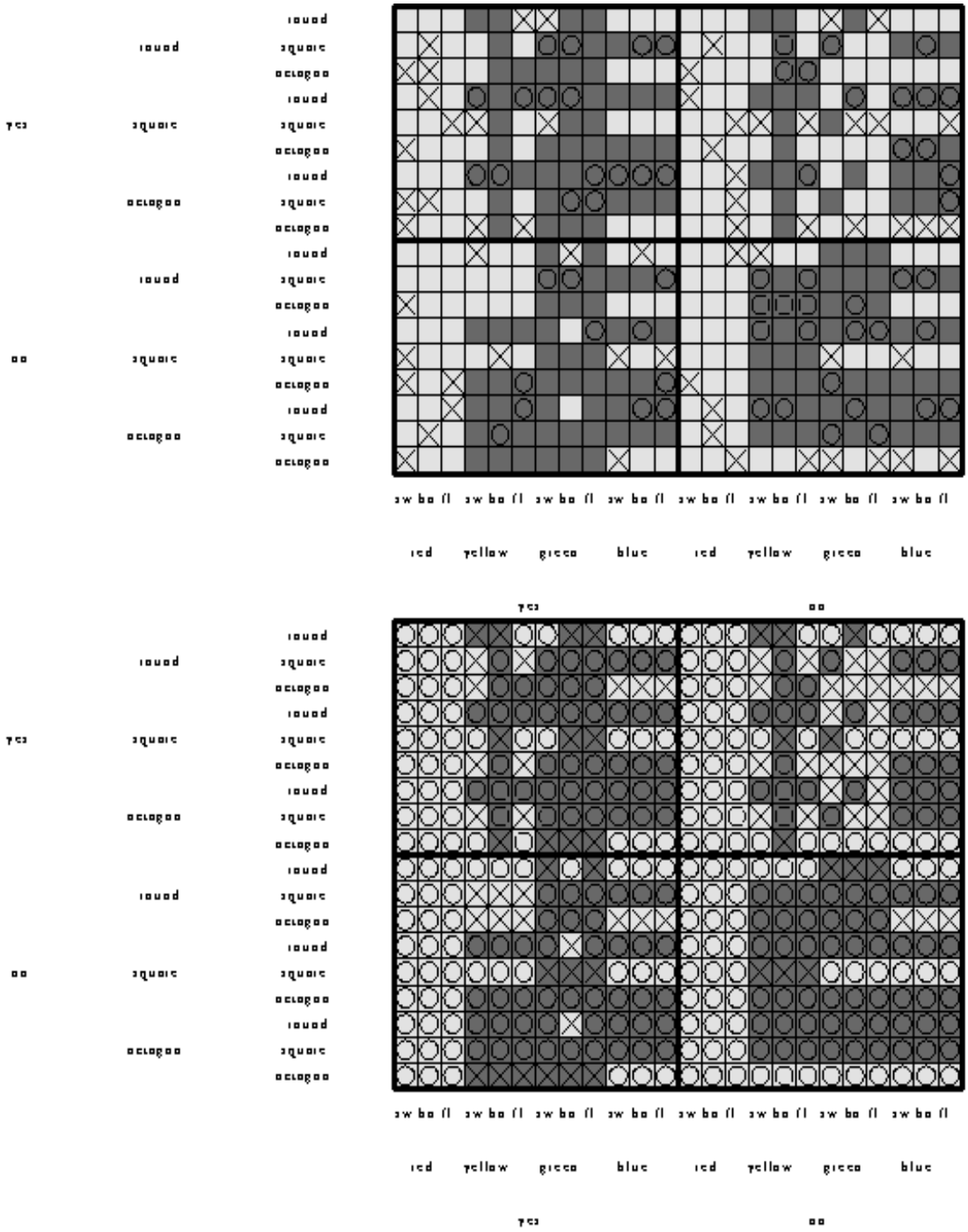

Figura 5: **GLD** para o *dataset* monk/ID3. Set=predictedTrain acima e Set=predictedTest embaixo.

Na próxima seção são discutidas as classificações de alguns dos indutores da biblioteca MLC^{++}, para determinar algumas opções do utilitário a ser usado.

5. INDUTORES

Os usuários da biblioteca MLC^{++} devem conhecer o funcionamento básico dos indutores, uma vez que não existe um indutor ótimo para todas as bases de dados. Existem indutores que têm um ótimo desempenho para algumas bases de dados, mas em outras, apresentam um pobre desempenho. Isto depende de vários fatores, como tamanho do arquivo de dados, número de registros, número de variáveis, quantidade de valores contínuos e discretos, etc. [Kohavi 97b]. Uma das tarefas dos usuários da MLC^{++} é encontrar o melhor indutor para uma determinada base de dados, observando os resultados provenientes da execução de alguns deles.

Alguns dos indutores implementados dentro da MLC++ são: Const, Decision Tables, ID3, Lazy Desicion Trees, Nearest-neighbor, Naive-bayes, IR, OODG, Option Decision Trees, Perceptron, Winnow. Existem ainda outros indutores, os externos, que fazem interface com a MLC++, tais como: C4.5, C4.5-rules, C4.5-unpruned, CART, CN2, IB, Neural network: Aspirin Migraines, OC1, PEBLS, Ripper, T2, etc. Uma breve descrição destes indutores é mostrada no Apêndice A.

Classificar todos estes indutores não é uma tarefa fácil, pois existe uma infinidade de métodos para agrupá-los. Neste trabalho foi feita uma classificação desses indutores, para ter uma melhor noção do seu funcionamento. O método de classificação escolhido parte da forma como os indutores aprendem, para depois ser dividido de acordo com seu paradigma de aprendizado de máquina, e então partir para uma classificação considerando a linguagem utilizada para descrever o aprendizado e finalmente classificá-los pelo modo em que podem ser integrados novos exemplos. A seguir é explicada esta classificação, para posteriormente mostrá-la graficamente na Figura 6.

O aprendizado de máquina é um conjunto de técnicas que executam duas tarefas fundamentais: generalizam a partir de um conjunto de exemplos conhecidos, e detalham estruturas destas generalizações. Existe uma forma básica de classificar os algoritmos de aprendizado de máquina, a partir da forma como eles aprendem, utilizando um aprendizado supervisionado, ou um aprendizado não supervisionado:

- **Aprendizado supervisionado:** a noção básica é de classificador. Os classificadores são sistemas que agrupam os conjuntos de dados de entrada em um número de possíveis categorias. Os padrões são encontrados explorando o número de casos conhecidos que mostram ou implicam padrões bem definidos e baseados nesses casos, as generalizações são formadas. Pode-se representar aprendizado supervisionado como um processo sintetizado, ou seja, generalizando a partir de casos conhecidos, de tal forma que, dado um conjunto de observações no qual as classes são conhecidas, o objetivo está em encontrar regras capazes de classificar novas observações entre as classes já existentes.
- **Aprendizado não supervisionado:** os padrões dos exemplos não são conhecidos em sua estrutura. Os padrões dos dados são encontrados começando de alguma caracterização lógica de regularidade. O ponto de partida do aprendizado não supervisionado é alguma descrição do tipo de padrão requerido. Pode-se representar aprendizado não supervisionado como um processo analítico, ou seja, expandindo alguma descrição implícita, de tal forma que, dado um conjunto de observações, o objetivo é tentar estabelecer a existência de classes ou *clusters* nos dados.

Os paradigmas de aprendizado de máquina podem ser classificados em:

- **Paradigma Simbólico**: busca aprender construindo representações simbólicas de um conceito através da análise de exemplos e contra-exemplos desse conceito. As representações simbólicas estão tipicamente na forma de alguma expressão lógica, árvore de decisão, regras de produção, rede semântica, etc. Os métodos de aprendizado simbólico podem ser tratados como:

- **Proposicionais**: é um dos mais simples formalismos lógicos existentes. Estes métodos lidam apenas com enunciados declarativos, chamados *proposições* (ou sentenças). Os enunciados exclamativos, imperativos e interrogativos são excluídos. Só usam sentenças que têm valores verdadeiro ou falso. É possível construir proposições compostas através do uso de conectivos do tipo *e* ($\cap$), *ou* ($\cup$), *não* ($\neg$), *condicional* ($\rightarrow$), etc. É importante ressaltar que em função do potencial de representatividade das linguagens, os métodos proposicionais utilizam estratégias na realização de busca para a expressão dos exemplos e das hipóteses aprendidas, que serão explicadas nesta mesma seção.
- **Relacionais**: estão representados pela Programação Lógica Indutiva (PLI) onde se constróem programas lógicos – restritos às cláusulas de Horn – a partir de exemplos positivos e negativos da relação a ser aprendida usando no aprendizado, informação fornecida pela teoria de domínio. Os indutores adotam uma estratégia bottom-up ou top-down de busca no espaço de hipóteses. De maneira geral, a PLI pode ser definida como o uso da lógica simbólica para a representação explícita de problemas e de suas bases de conhecimento associadas, bem como o uso de inferência lógica controlada para a solução efetiva desses problemas. É importante ressaltar que em função do potencial de representatividade das linguagens, os métodos relacionais utilizam estratégias na realização de busca para a expressão dos exemplos, das hipóteses aprendidas e da teoria de domínio, que serão explicadas nesta mesma seção.

♦ **Paradigma Estatístico:** as técnicas estatísticas, em geral, tendem a focar tarefas em que todos os atributos têm valores contínuos ou ordinais. Muitos deles também são paramétricos, assumindo alguma forma de modelo, e então encontrando valores apropriados para os parâmetros do modelo a partir de dados. Os classificadores estatísticos freqüentemente assumem que valores de atributos estão normalmente distribuídos, e então usam os dados fornecidos para determinar média, variância e co-variância da distribuição.

♦ **Paradigma Instance-based:** uma forma de classificar um caso é lembrar de um caso similar cuja classe é conhecida e assumir que o novo caso terá a mesma classe. Esta filosofia exemplifica os sistemas instance-based, que classificam casos nunca vistos através de casos similares conhecidos [Quinlan 86].

♦ **Paradigma Conexionista:** as redes neurais são construções matemáticas relativamente simples que utilizam o mecanismo de paralelismo, onde são conectados um grande número de pequenas unidades de processamento ligadas em rede. Seus elementos são modelados como os neurônios e foram inspirados no modelo biológico do sistema nervoso. Sua representação envolve unidades altamente interconectadas, no qual o nome conexionismo é utilizado para descrever a área de estudo.

♦ **Paradigma Genético:** este formalismo de classificação é derivado do modelo evolucionário de aprendizado [Holland 86]. Um classificador genético consiste de uma população de elementos de classificação que competem para fazer uma predição. Elementos que possuem uma performance fraca são descartados, enquanto os elementos mais fortes proliferam, produzindo variações de si mesmos. Este paradigma possui uma analogia direta com a teoria de Darwin, onde sobrevivem os mais bem adaptados ao ambiente.

Em um sistema de aprendizado é necessário utilizar linguagens para representar exemplos ou instâncias, conceitos induzidos e dependendo do caso, a teoria de domínio previamente conhecida. Estas linguagens são comumente referenciadas como *linguagens de descrição*, e estão divididas por:

♦ **Linguagem de descrição de exemplos ou instâncias:** descreve os exemplos utilizados pelo programa para aprender conceitos, estabelecendo limites sobre os tipos de padrões que o sistema pode aprender.

♦ **Linguagem de descrição de hipóteses ou conceitos aprendidos:** descreve o estado interno de um programa de aprendizado, correspondente a teoria dos conceitos ou padrões que existem nos dados, estabelecendo limites sobre o que pode ou não pode ser aprendido.

- **Linguagem de descrição da teoria de domínio ou conhecimento de fundo**: descreve todo o conhecimento prévio que o programa possui a respeito do domínio.

Alguns dos formalismos para descrever objetos, relações entre eles, conceitos e a teoria de domínio utilizada podem ser:

- **Atributos para representar exemplos de treinamento:** um atributo é uma possível característica relevante do conceito a ser aprendido. A partir dos exemplos de treinamento expressos como vetores de pares atributo-valor, pertencendo cada um deles a uma classe, entre várias mutuamente exclusivas, tenta-se encontrar uma regra que prediga a classe de um novo exemplo em função de seus atributos e valores.
- **Árvores de decisão para representar conceitos**: é um método para aprender conceitos quando objetos são descritos através de vetores *atributo-valor* e uma classe associada, representadas por nós e ramos. Sob o título de família *TDIDT (Top Down Induction of Decision Trees*), as árvores de decisão são construídas de uma maneira *top-down*, ou seja, a partir da raiz, descendo até as folhas. Os exemplos classificados, a partir dos quais é gerada a árvore de decisão, são conhecidos somente através de um conjunto de atributos e de seus respectivos valores. Costuma-se, geralmente, considerar a classe do exemplo como um atributo. As árvores de decisão, por sua vez, são construídas usando estes mesmos atributos.
- **Regras de produção para representar conceitos:** tentam resolver um problema encontrando regras que cubram várias partes do problema que posteriormente, serão agrupadas na forma normal disjuntiva, ou seja, "unidas" com o operador $\cup$. Basicamente, uma regra de produção primeiro aprende uma regra que cobre uma parte dos exemplos dados, remove os exemplos cobertos por aquela regra do conjunto de treinamento e recursivamente aprende outra regra que cobre os exemplos restantes até que todos os exemplos de treinamento sejam processados.
- **Redes semânticas para representar conceitos:** é uma tentativa de se formalizar como o conhecimento é organizado na memória mostrando o relacionamento entre objetos e propriedades. São compostas de nós e conectores rotulados, onde cada nó representa um objeto ou a propriedade de um objeto e cada conector representa o relacionamento entre dois nós. Uma rede semântica pode ser usada como uma ferramenta explicativa, para explorar exaustivamente um tópico ou para encontrar o relacionamento entre dois objetos.
- **Frames para representar objetos complexos**: descrevem objetos complexos, onde os objetos são definidos como pertencendo a classes – que por sua vez são objetos – das quais herdam propriedades e, às vezes, procedimentos. Um frame é identificado por um nome e consiste de um conjunto de slots (conjunto de atributos denominados facetas de valores particulares), tal que cada *slot* possui um nome único ao frame no qual está definido.
- **Scripts para representar situações e objetos**: são uma especialização de frames projetados para manipular situações além de objetos. Em scripts, os nós são eventos e os conectores entre eles são simplesmente causais, isto é, um evento provoca o próximo. Um script é como um script cinematográfico e é usado para contar histórias sobre uma seqüência de eventos, responder questões em determinado ponto do nó e identificar eventos que levem a alguma decisão, normalmente são considerados como uma subclasse dos frames.

Os algoritmos de aprendizado indutivo podem ser classificados de duas formas segundo o modo em que podem ser integrados novos exemplos:

- **Não Incremental:** necessita de que todos os exemplos de treinamento, simultaneamente, estejam disponíveis para que seja induzido um conceito. É vantajoso usar esses algoritmos para problemas de aprendizado onde todos os exemplos estão disponíveis e, provavelmente, não irão ocorrer mudanças.
- **Incremental:** revê a definição do conceito corrente, se necessário, em resposta a cada nova instância de treinamento observada. Os exemplos observados são considerados um a um

pelo sistema, isto é, o sistema considera o primeiro exemplo e, de acordo com esse exemplo, constrói uma determinada hipótese; a seguir considera um segundo exemplo, que pode ou não modificar a primeira hipótese, baseando-se em como esta classifica o segundo exemplo. A medida que mais exemplos são apresentados, o sistema continua modificando o conceito.

Na Figura 6 é apresentado o resumo desta classificação.

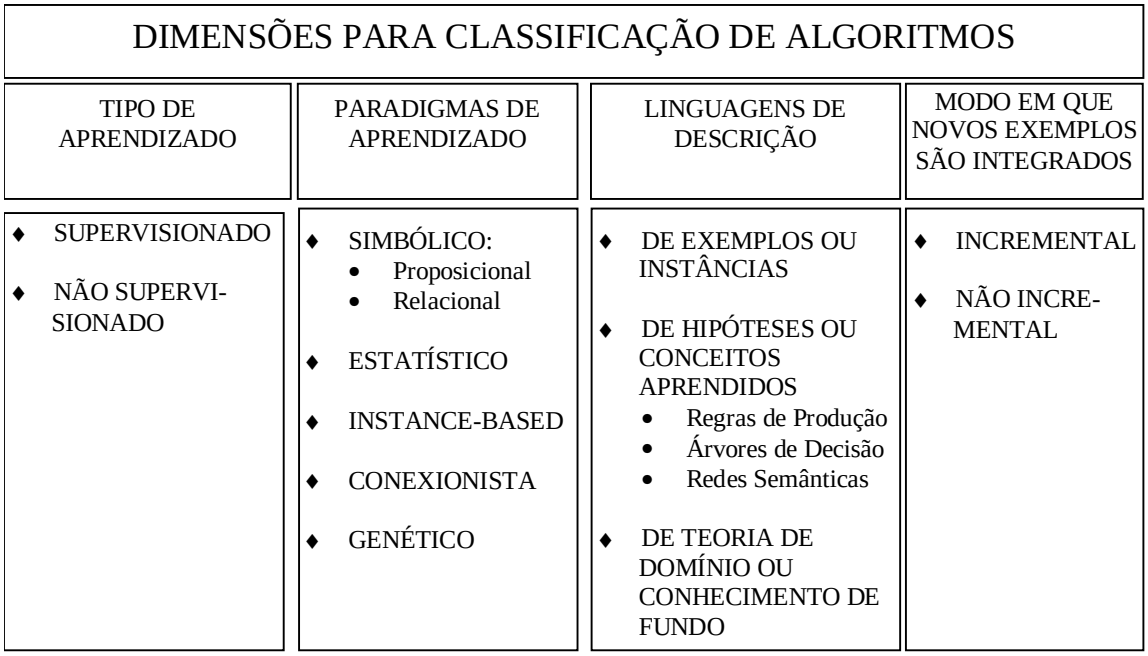

Figura 6: Uma forma de classificar os algoritmos de aprendizado de máquina.

A Tabela 13 apresenta a classificação de alguns dos algoritmos utilizados pela biblioteca MLC^{++}, de acordo com a classificação mostrada anteriormente:

Indutor	**Tipo**	**Paradigma**	**Linguagem**	**Incrementa -bilidade**	**Observações**
aha-IB	Supervisionado	Instance-based	Exemplos	Incremental	
C4.5	Supervisionado	Proposicional	Árvore de decisão	Não incremental	
C4.5rules	Supervisionado	Proposicional	De árvore de decisão para regras de produção	Não incremental	
C4.5unpruned	Supervisionado	Proposicional	Árvore de decisão	Não incremental	
CART	Supervisionado	Estatístico	Árvore de decisão	Não incremental	
CN2	Supervisionado	Proposicional	Regras de produção	Não incremental	
Decision Tables	Supervisionado	Instance-based	Exemplos	Incremental	
HOODG	Supervisionado		Redes semânticas		
ID3	Supervisionado	Proposicional	Árvore de decisão	Não incremental	
MC4	Supervisionado	Proposicional	Árvore de decisão	Não incremental	

Indutor	Tipo	Paradigma	Linguagem	Incrementa-bilidade	Observações
Naive Bayes	Não supervisionado	Estatístico			
Nearest-neighbor	Não supervisionado	Estatístico		Incremental	Pode trabalhar como supervisionado
Neural Network: Aspirine Migraine	Supervisionado	Conexionista	-	Não incremental	Não tem linguagem
OC1	Supervisionado	Proposicional	Árvore de decisão	Não incremental	
OneR	Supervisionado	Proposicional	Regras de produção	Não incremental	
PEBLS	Supervisionado		Exemplos		
Perceptron	Supervisionado	Conexionista	-	Não incremental	Não tem linguagem
T2	Supervisionado	Proposicional	Árvore de decisão	Não incremental	
Winnow	Supervisionado	Conexionista	-	Não incremental	Não tem linguagem

Tabela 13: Classificação dos Indutores usados pela MLC^{++}.

A próxima seção trata sobre o conceito *wrapper*, que é um indutor que envolve outro indutor tentando modificar seu próprio comportamento para melhorar o seu desempenho.

6. WRAPPERS

Sendo os algoritmos da MLC^{++} tratados como objetos C^{++}, isso possibilita a construção de *wrappers* (invólucros). Um *wrapper* é um algoritmo que trata outro algoritmo como uma caixa preta para modificar seu comportamento, esperando melhorar seu desempenho ou permitindo realizar operações que não poderiam ser executadas de outra maneira.

Os dois mais importantes *wrappers* na MLC^{++} são os estimadores de precisão e os seletores de características. Os estimadores de precisão usam métodos como *holdout*, *cross-validation*, ou *bootstrap* para calcular o desempenho de um indutor [Kohavi 95a]. Os métodos de seleção de características executam uma busca usando seu próprio indutor para determinar quais atributos na base de dados são interessantes. O *wrapper* aproveita a seleção automática de características para determinar a característica mais importante e fornecê-la para o indutor que está sendo executado [Kohavi 95f].

Um *wrapper de votação* executa um algoritmo em partes diferentes do *dataset* e permite *votar* sobre a classe previsível [Wolpert 92] [Breiman 96] [Perrone 93] [Ali 96]. Um *wrapper de discretização* pré-discretiza os dados, permitindo aos algoritmos que não suportam características contínuas (ou aqueles que não se comportam bem) trabalhar corretamente. Um *wrapper de otimização de parâmetro* permite refinar os parâmetros de um algoritmo automaticamente, baseado em uma busca no espaço de parâmetros que otimiza a estimativa de precisão com parâmetros diferentes.

A habilidade para criar algoritmos híbridos e *wrappers* é muito importante para o aproveitamento do aprendizado com multiestratégias.

O indutores **Discretization-filter**, **Bagging**, **Feature Subset Selection** e **C4.5-auto-parm** são criados como combinações de outros e são descritos a seguir.

6.1. Discretization filter

O **Disc-filter** é um indutor *wrapper* que pega o indutor da opção **DISCF_INDUCER**. As opções para o indutor *wrapper* serão antepostas pelo prefixo **DISCF_**.

A opção mais importante é o tipo de discretização: **entropy**, **1r**, **bin**, **c4.5-disc**, **t2-disc**. A discretização por entropia parece ser o melhor método de discretização das opções permitidas para a maioria dos *datastes* reais [Dougherty 95]. Algumas opções são:

- **DISC_NUM_INTR:** determina o número de intervalos. As possíveis opções são:
  - **Algo-heuristic:** é a heurística dependente do algoritmo.
  - **Fixed-value**: número para ser especificado.
  - **MDL** (*Multi-interval Discretization of continuos-valued attributes for classification Learning*) [Fayyad 93].
- **MIN_SPLIT:** determina o número mínimo de instâncias no método de entropia que se requer em um intervalo e a heurística do algoritmo *default* ao **MDL**. O método **1r** é o método de Holte de discretização usado na regra OneR [Holte 93].
- **MIN_INST:** determina o número mínimo de instâncias (0 será alterado para 6, o padrão sugerido por Holte). O método de instâncias usa binning uniforme (intervalos iguais) e a heurística do algoritmo para o número de instâncias é usar duas vezes o logaritmo (base 2) do número de valores distintos. A heurística do algoritmo T2 [Maass 94] [Auer 95] forma um número de classes com uma instância.

6.2. Bagging

O **Bagging** é um indutor *wrapper* especificado na opção **BAG_INDUCER**, e executado várias vezes em subconjuntos do conjunto de treinamento. Durante a classificação, os classificadores induzidos votam, e então a classe majoritária é escolhida [Breiman 94] e [Wolpert 92]. O indutor *wrapper* deve ser um indutor regular (não um indutor base)(ver Apêndice A).

O **Bagging** parece trabalhar melhor com indutores instáveis, ou seja, indutores que sofrem alta variação devido a pequenas perturbações nos dados [Geman 92]. Os indutores instáveis incluem árvores de decisão (por exemplo, ID3) e perceptrons; um exemplo de um indutor estável é o nearest-neighbor que tem uma alta influência em espaços dimensionais elevados, mas com pequena variação. O que se perde com o bagging é a habilidade para entender os dados: tem-se 20 especialistas que votam no rótulo e são alcançados melhores desempenhos se eles são todos bons, mas discordam entre eles muitas vezes.

As opções para **Bagging** são:

- **BAG_REPLICATIONS**: determina o número de classificadores para criar, onde "o mais", "a melhor", no sentido que o resultado será mais estável.
- **BAG_PROPORTION**: determina a proporção do conjunto de treinamento que será passado para cada cópia do indutor. Uma proporção mais alta vai ter um conjunto maior de treinamento, porém, se o dados não forem bastante perturbados, os classificadores não serão diferentes e bagging não trabalhará bem [Krogh 95].
- **BAG_UNIF_WEIGHTS**: é uma opção booleana que determina se os votos são iguais ou estimados. Se os votos são estimados, a parte do conjunto de treinamento que não foi usada para treinamento interno é usada como um conjunto de teste e a precisão estimada é o peso associado ao categorizador induzido. Devido a alta variação na estimação, esta opção parece não trabalhar bem na prática.

6.3. Feature Subset Selection

A **Feature Subset Selection** (FSS Seleção de Subconjunto de Característica) é um indutor *wrapper* que seleciona um bom subconjunto de características para melhorar o desempenho da estimativa de precisão de um algoritmo [Kohavi 94a] [Kohavi 95c] [Kohavi 95f].

Todas as opções de estimativa de precisão podem ser usadas com as opções extras, listadas na Tabela 14.

Nome da opção	Domínio	Default	Explicação
FSS_INDUCER	Inducer	-	Deve estar acompanhada de um Indutor.
FSS_DOT_FILE	Filename	FSS.dot	Especifica o nome do arquivo dot que conterá o espaço de busca.
FSS_SEARCH_ METHOD	hill-climbing, best-first	best-first	Seleciona o método de busca no espaço de subconjuntos de características.
FSS_ DIRECTION	forward, backward	forward	Começa com um conjunto vazio de características (forward) ou todas com as características (backward).
FSS_EVAL_ LIMIT	int ≥ 0	0	Define o número máximo de avaliações do conjunto de teste. Este é um método de parada na busca (o outro é **MAX_STALE**). O valor zero (0) implica em nenhum limite.
FSS_SHOW_ REAL_ACC	always, best-only, final-only, never	best-only	Mostra a precisão no conjunto de teste. Note que esta precisão não é usada pelo mecanismo de busca para dirigir a busca. Se a precisão não for **never** então este indutor se comporta como um indutor base.
FSS_MAX_ STALE	int > 0	5	Uma busca é considerada passada se este número de nós não melhorados for ampliado. Isto determina uma condição de término.
FSS_EPSILON	real ≥ 0	0.001	Considera um nó não melhorado se a precisão estimada for melhor que a melhor precisão dada por este número.
FSS_USE_ COMPOUND	yes, no	yes	Gera nós que combinam as características dos melhores filhos gerados [Kohavi 95c].
FSS_CMPLX_ PENALTY	real	0.001	Quanto deve ser penalizada a estimativa para cada característica.

Tabela 14: Opções para o *wrapper* **Feature Subset Selection**.

No exemplo 11, pode-se observar a execução do indutor IB no ***wrapper*** **Feature Subset Selection** usando o *dataset* crx.

Exemplo 11:

```
setenv LOGLEVEL 1
setenv INDUCER FSS
setenv FSS_INDUCER IB
setenv DATAFILE crx.data
setenv TESTFILE crx.test
setenv FSS_DOT_FILE IBFSS.dot
Inducer
```

A saída será:

```
MLC++ Debug level is 0, log level is 1
OPTION PROMPTLEVEL = required-only
OPTION INDUCER = FSS
```

```
OPTION INDUCER_NAME = FSS
OPTION FSS_INDUCER = IB
OPTION FSS_INDUCER_NAME = IB
OPTION FSS_NUM_NEIGHBORS = 1
OPTION FSS_EDITING = false
OPTION FSS_NNKVALUE = num-distances
OPTION FSS_NORMALIZATION = extreme
OPTION FSS_NEIGHBOR_VOTE = inverse-distance
OPTION FSS_MANUAL_WEIGHTS = false
OPTION FSS_DOT_FILE = IBFSS.dot
OPTION FSS_SEARCH_METHOD = best-first
OPTION FSS_SHOW_TEST_SET_PERF = best-only
OPTION FSS_EVAL_LIMIT = 0
OPTION FSS_MAX_STALE = 5
OPTION FSS_EPSILON = 0.001
OPTION FSS_PERF_ESTIMATOR = automatic
OPTION FSS_CV_WEIGHT_THRESHOLD = 5000
OPTION FSS_ERROR_TYPE = error
OPTION FSS_PERF_EST_SEED = 7258789
OPTION FSS_ERROR_TRIM = 0
OPTION FSS_CV_FOLDS = 10
OPTION FSS_CV_TIMES = 1
OPTION FSS_CV_FRACT = 1
OPTION FSS_HO_TIMES = 1
OPTION FSS_HO_NUMBER = 0
OPTION FSS_HO_PERCENT = 0.666666666667
OPTION FSS_USE_COMPOUND = true
OPTION FSS_CMPLX_PENALTY = 0
OPTION FSS_USE_COMPOUND = true
OPTION FSS_CMPLX_PENALTY = 0
OPTION FSS_DIRECTION = forward
OPTION DATAFILE = crx.data
OPTION NAMESFILE = crx.names
OPTION WEIGHT_IS_ATTRIBUTE = true
OPTION REMOVE_UNKNOWN_INST = false
OPTION CORRUPT_UNKNOWN_RATE = 0
Reading crx.data.... done.
OPTION TRAIN_WITHOUT_LOSS = false
OPTION TESTFILE = crx.test
Reading crx.test.. done.
OPTION COMPUTE_LOG_LOSS = false

New best node (1 evals) #0[]: error: 44.29% +- 2.43% (32.65% - 55.10%).
  Test Set: 45.00% +- 3.53% [38.26% - 51.92%].  Bias: -0.71% cost: 10
   complexity: 0
.................
New best node (18 evals) #9[8]: error: 13.47% +- 2.00% (6.12% - 24.49%).
  Test Set: 17.00% +- 2.66% [12.43% - 22.82%].  Bias: -3.53% cost: 10
   complexity: 1
..............
New best node (35 evals) #33[3, 4, 8, 9, 11]: error: 12.65% +- 1.69% (6.12% -
   22.45%).
  Test Set: 15.50% +- 2.57% [11.14% - 21.16%].  Bias: -2.85% cost: 10
   complexity: 5
..............
```

```
New best node (50 evals) #48[3, 4, 8, 9, 12]: error: 11.02% +- 1.98% (2.04% -
  20.41%).
 Test Set: 16.50% +- 2.63% [12.00% - 22.27%].  Bias: -5.48% cost: 10
  complexity: 5
.....................................................................
Final best node #48[3, 4, 8, 9, 12]: error: 11.02% +- 1.98% (2.04% - 20.41%).
 Test Set: 16.50% +- 2.63% [12.00% - 22.27%].  Bias: -5.48% cost: 10
  complexity: 5
Expanded 9 nodes
Classifying (% done): 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% done.
OPTION DISP_LIFT_CURVE = false
OPTION DISP_MISCLASS = false
Number of training instances: 490
Number of test instances: 200.  Unseen: 200,  seen 0.
Number correct: 167.  Number incorrect: 33
Generalization error: 16.500%.  Memorization error: unknown
Error: 16.500% +- 2.631% [11.997% - 22.266%]
Average Normalized Mean Squared Error: 16.500%
Average Normalized Mean Absolute Error: 16.500%
OPTION DISP_CONFUSION_MAT = no
OPTION DISP_CONF_SCATTERGRAM = false
```

Este exemplo mostra que pode-se melhorar a precisão de 15.5% para 16.5% observando somente cinco características. Neste caso sabe-se que estas são as únicas cinco características relevantes, mas é importante notar que elas foram encontradas automaticamente. A Figura 7 mostra os nós visitados e sua informação. O gráfico é armazenado automaticamente no arquivo FSS.dot. As extremidades mostram a diferença na precisão estimada entre os dois nós. A informação em cada nó do gráfico é a seguinte:

1. A primeira linha indica o número do nó mostrado pela ordem em que foram calculados. Os números entre parênteses mostram o conjunto de características usadas a partir da característica 0.
2. Na segunda linha aparece a estimativa de precisão (por exemplo, *cross-validation*, *bootstrap*, *holdout*) com a média do desvio padrão.
3. A terceira linha só aparecerá para nós onde a precisão real foi calculada (a precisão no conjunto de teste). O cálculo depende do valor mostrado em **FSS_REAL_ACC**. Por *default*, somente os nós que foram "os melhores" em alguma fase terão este número. Note que esta precisão nunca é usada pelo algoritmo de busca.

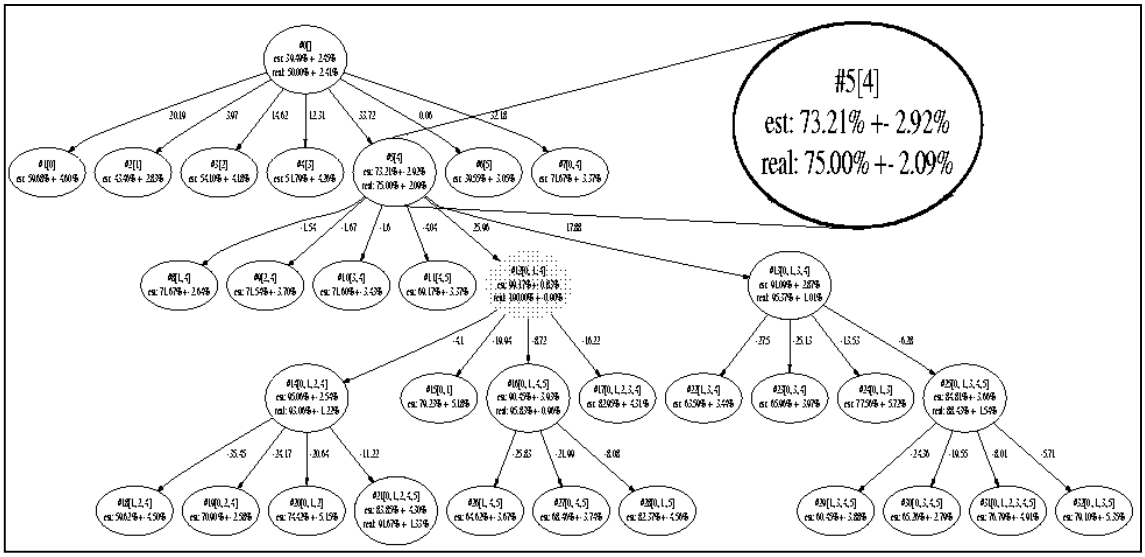

Figura 7: Espaço de busca para IB no *dataset* crx.

6.4. C4.5-auto-parm

O **C4.5-auto-parm** é um algoritmo *wrapper* que executa uma busca sobre possíveis valores de parâmetros para C4.5 e tenta escolher o melhor para um determinado *dataset*. Pode-se decidir quais parâmetros variar, fixando as opções **AP_VARY_X**, onde X é M,C,G ou S. O significado destas opções pode ser encontrado em [Quinlan 93].

Quase todas as opções aplicáveis à busca FSS são aplicáveis aqui com o prefixo **AP_** no lugar do prefixo **FSS_**. O espaço de busca explorado é colocado no arquivo AP.dot e pode ser visto usando *dot* ou *dotty* (ver seção 7).

Uma descrição do algoritmo pode ser encontrada em [Kohavi 95d], embora algumas mudanças desde então, significa que os resultados não casam exatamente.

A próxima seção trata das ferramentas gráficas que existem para visualizar as saídas dos indutores, sempre que for permitido.

7. VISUALIZAÇÃO

Alguns algoritmos de indução suportam saída visual dos classificadores. Todo suporte gráfico dos algoritmos pode ser exibido em representações bidimensionais usando os utilitários *dot* e *dotty* [Koutsofios 94]. O *dotty* também é capaz de exibir informação extra de cada nó, como as distribuições de classe. A Figura 8 mostra a visualização de um gráfico usando o *dot*.

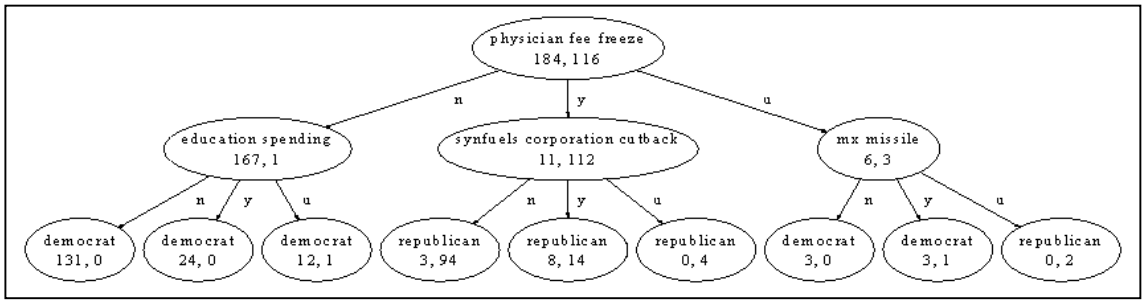

Figura 8: Árvore de decisão usando *dot* no *dataset* votes.

Os algoritmos que geram árvore de decisão, como ID3, podem gerar saídas para o *MineSet*® da Silicon Graphics® (ver apêndice B). O *Tree Visualizer*® dá uma visão interativa tridimensional de uma árvore de decisão com capacidade de navegação. A Figura 9 mostra um instante da navegação.

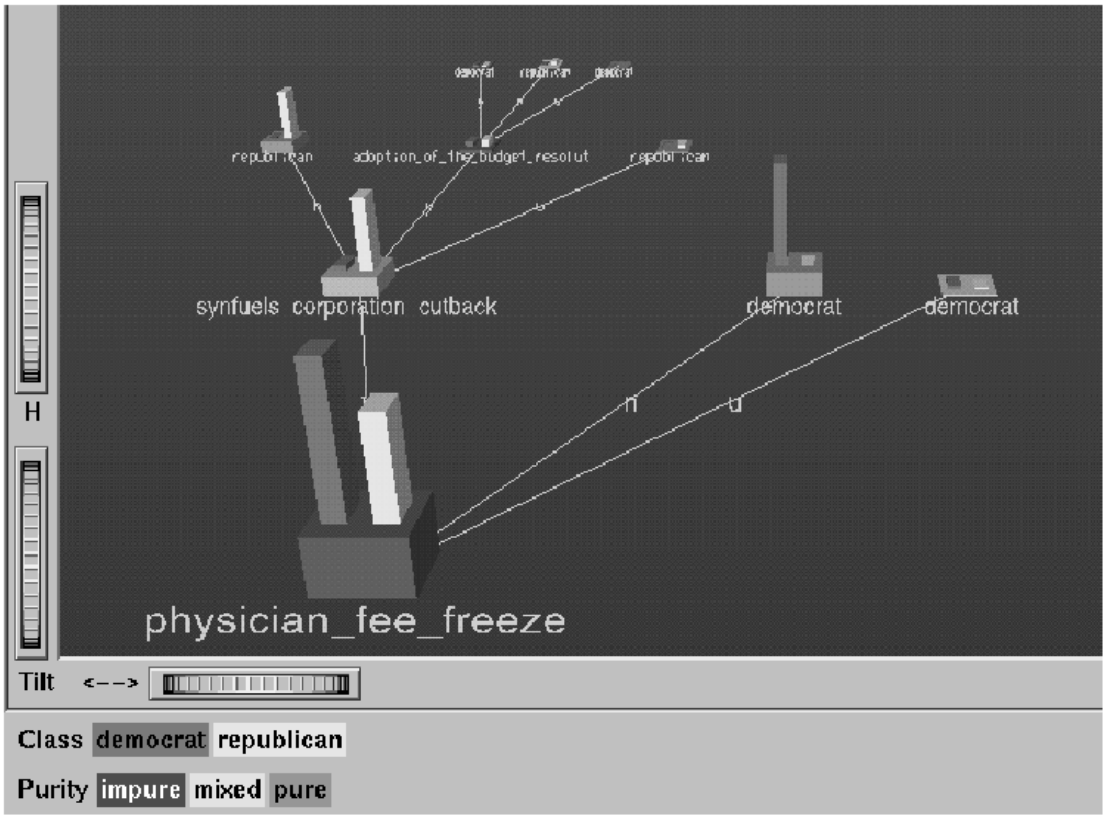

Figura 9: Árvore de decisão usando *Tree Visualizer*® no *dataset* votes.

A biblioteca também provê um utilitário para exibir Diagramas de Lógica Geral de classificadores implementados em MLC^{++} (ver Diagramas de Lógica Geral, seção 4.11).

Finalmente, o algoritmo *Naive-Bayes* pode ser visualizado de duas formas distintas, ambas contanto com o utilitário *Evidence Visualizer*® do *MineSet*®. A primeira visão, na Figura 10, mostra as probabilidades diretas com gráficos de torta tridimensionais, onde a altura da torta representa o número de instâncias para aquele valor ou intervalo.

Cada gráfico representa a distribuição de classes dado que um único atributo é fixado para um valor específico. As tortas são uma visualização útil, mas os valores de probabilidade devem ser multiplicados para formar as probabilidades posteriores usadas para predição. A segunda visão, na Figura 11, está baseada na seleção de uma única classe. Os gráficos de torta são substituídos por gráficos de barras tridimensionais. A altura de cada barra representa a probabilidade logarítmica (evidência) a favor da classe especificada, dado que um único atributo é fixado a um valor específico. As probabilidades logarítmicas são úteis porque elas podem ser somadas para formar o seguinte, tornando o processo de classificação *Naive-Bayes* mais intuitivo. As barras se tornam menos saturadas quando o número de instâncias decresce, significando um intervalo de confiança mais amplo.

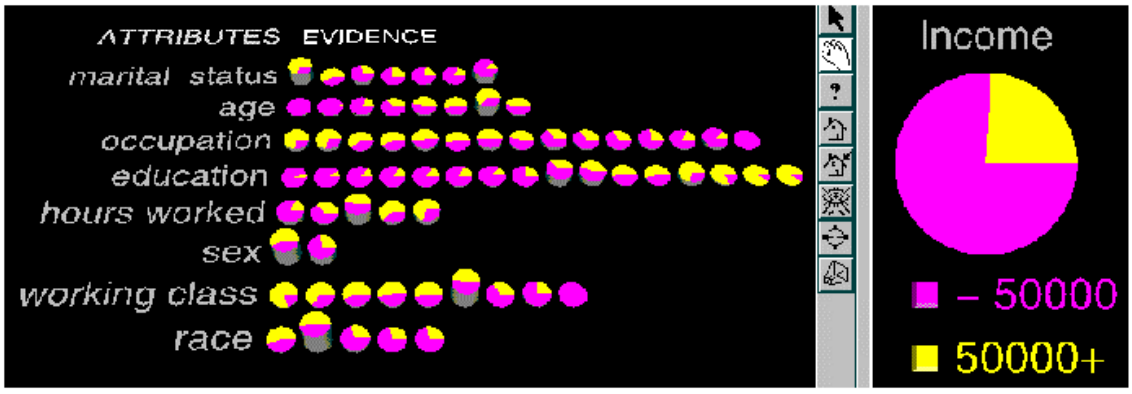

Figura 10: Gráfico de torta no *Evidence*® do modelo Naive-Bayes.

Figura 11: Gráfico de barras no *Evidence*® do modelo Naive-Bayes.

Ambas visões suportam classificação interativa de dados. Um único gráfico de torta inicialmente à esquerda mostra a distribuição da classe anterior. Os usuários podem interagir com a visualização selecionando valores para os atributos e observando como a probabilidade posterior (gráfico de torta à direita) muda. Por exemplo, selecionando a torta para *sepal-lenght* < 5.45 polegadas e a torta para *sepal-windht* > 3:05 polegadas, mostra que uma íris com estas características provavelmente é uma íris-setosa (Figura 12). Usuários podem fazer perguntas da forma: "como seria se eu tivesse uma instância com certos valores" e ver as predições do classificador e as probabilidades posteriores de acordo com o modelo *Naive-Bayes*.

Figura 12: Gráfico de torta do *dataset* iris selecionando uma sépala.

8. LISTAS DE DISCUSSÃO

A lista de discussão *smallmlcpp@cs* envia mensagens para pessoas que trabalham com a MLC^{++} e é direcionada para discussões de problemas específicos com pormenores. Um outra lista *mlcpp@cs* envia mensagens a *smallmlcpp@cs* e também para outras pessoas interessadas em MLC^{++}, trabalhando assim de um modo mais geral.

Existe ainda mais uma lista de discussão criada pela Silicon Graphics®, onde se recebe informações técnicas da MLC^{++}, e o endereço é http://www.sgi.com/Technology/mlc/mail.html.

9. CONSIDERAÇÕES FINAIS

Um indutor pode ser visto como um algoritmo de aprendizado de máquina capaz de criar uma classificação a partir de um conjunto de exemplos. O principal objetivo está em extrair conceitos expressos em alguma linguagem, por exemplo, regras de produção ou árvores de decisão, capazes de serem aplicadas a novos casos.

A MLC^{++} tenta juntar alguns indutores dentro de uma biblioteca desenvolvida em C^{++}. O problema não é fácil, já que cada indutor tem seu próprio formato de dados de entrada. A biblioteca MLC^{++} converte de forma automática todas as entradas próprias para cada indutor a partir do formato C4.5 [Quinlan 86].

Existe um conceito interessante dentro da biblioteca chamado *wrapper*, onde um indutor envolve um outro indutor tentando modificar seu próprio comportamento para melhorar o seu desempenho. Usando o *wrapper* estimador de precisão, além de obter os dados fornecidos pelo indutor, este *wrapper* fornece estimativa de precisão usando os métodos *holdout*, *cross-validation*, ou *bootstrap* para calcular o desempenho do indutor. O *wrapper* **Feature Subset Selection** (FSS) executa uma busca usando seu próprio indutor para determinar quais atributos na base de dados são interessantes e os dar para o indutor que está sendo executado.

As declarações de que um classificador é melhor do que outro, como aquela que diz que “as regras são muito melhores do que as árvores de decisão” feita por [Parsaye 96], são muitas vezes irrelevantes e freqüentemente mal interpretadas. Existem dois problemas com tais afirmações: o que é "melhor" e como a semântica "melhor" explica o aprendizado.

Vários critérios podem ser usados para comparar estruturas e que por serem mais gerais ou mais poderosas não implica que é mais fácil de aprender; de fato, o oposto é verdadeiro: estruturas mais poderosas são provavelmente mais difíceis de aprender, como os Frameworks Provavelmente e Aproximadamente Corretos proposto por Valiant [Valiant 84] [Kearns 94]. Modelos de Programação com Lógica Indutiva [Lavrac 94] [Muggleton 92] são mais poderosos do que os métodos de aprendizado proposicional, contudo poucos sistemas práticos estão disponíveis e são tão precisos quanto os sistemas proposicionais.

Resultados mostram que não há um único algoritmo que pode ser uniformemente mais preciso do que outros em todos os domínios [Kohavi 97b]. Embora tais teoremas sejam de aplicabilidade limitada na prática, sabe-se muito pouco sobre qual dos algoritmos deverá ser escolhido para problemas específicos. Alguns dos critérios que podem ser utilizados para medir a qualidade das regras geradas por um sistema de aprendizado descritos em [Henery 94] são:

- **Precisão:** é o grau de confiabilidade das regras. Geralmente é representado pela taxa de erro, embora em alguns casos existam erros mais sérios que outros, e em certos casos pode ser importante controlar a taxa de erro em algumas classes chaves.
- **Velocidade:** em algumas circunstâncias a velocidade de um classificados é o aspecto mais importante. Um classificador que é 90% preciso pode ser preferido à outro que é 95% preciso se ele for 100 vezes mais rápido nos testes. Tais considerações podem ser importantes para aplicações que possuem sérias restrições de tempo nas suas tarefas.
- **Compreensibilidade:** se é um operador humano que deve aplicar o procedimento de classificação, o procedimento deve ser facilmente compreensível ou senão enganos podem ser feitos na aplicação de regras.
- **Tempo de Aprendizado:** especialmente em um ambiente que muda rapidamente, pode ser necessário aprender uma regra de classificação rapidamente, ou fazer ajustes a uma regra existente em tempo real. “Rapidamente” pode implicar também que seja necessário somente um pequeno número de observações para estabelecer a regra.

Este Relatório Técnico apresenta no Apêndice A uma breve explicação dos principais indutores. No apêndice B é apresentada uma forma como a biblioteca MLC^{++} pode ser usada através do *MIndUtil®*, um programa desenvolvido pela *Silicon Graphics®* que junto com uma interface gráfica e softwares para visualizar gráficos (*Tree Visualizer®*, *Evidence Visualizer®*, etc.) compõem o *Mineset®* que foi desenvolvido para ajudar na extração de conhecimento em bases de dados (Data Mining).

É importante ressaltar que este trabalho é a primeira versão de um Relatório Técnico que trata da biblioteca MLC^{++}. Espera-se qualquer contribuição das pessoas que usem este trabalho, no sentido de corrigí-lo e melhorá-lo.

Os autores agradecem a ajuda a todas as pessoas que colaboraram com suas críticas e sugestões na realização deste trabalho, em especial à Profa. Dra. Maria Carolina Monard e a Jaqueline Brigladori Pugliesi.

APÊNDICE A

TIPOS DE INDUTORES

A MLC^{++} suporta vários indutores (algoritmos de indução), que estão classificados em: *indutor regular*, que deve ser implementado na própria MLC^{++}, e *indutor base*, que pode ser implementado na MLC^{++} ou pode ser um indutor externo. Um indutor base não pode categorizar instâncias isoladas, somente um conjunto de instâncias. Todos os indutores externos que fazem interface com a MLC^{++} (por exemplo, C4.5, PEBLS, aha-IB, OC1 e T2) são indutores base. Os indutores base são determinados pelo conjunto de treinamento e conjunto de testes a fim de retornar a estimativa de precisão. Alguns algoritmos que fazem parte da MLC^{++} também são indutores base ou podem se comportar de maneira análoga sob certas condições. Por exemplo, se a opção do indutor **Feature Subset Selection** (FSS) **SHOW_REAL_ACC** não for **never**, então o FSS se comportará como um indutor base porque deve ter acesso ao teste fixado para exibir a precisão real e o seu progresso (esta precisão não é usada no processo de indução; é somente usada para fins de exibição). Além dos detalhes técnicos, alguns *wrappers* (por exemplo, *bagging*) só suportam operações sobre indutores regulares. A matriz de confusão no utilitário **Inducer** é uma opção somente fornecida para indutores regulares. A seguir será brevemente explicados alguns indutores utilizados pela biblioteca MLC^{++}:

Aha Instance-based series (IBL)
O indutor aha-IB é uma forma de classificar um caso lembrando um caso similar cuja classe é conhecida e assumir que o novo caso terá a mesma classe. O Aha-IB é um indutor externo que faz interface com as séries IB1-4 a partir de 3/9/94 [Aha 92]. A opção **IB_CLASS** deverá ser fixada em um dos seguintes valores: **ib1**, **ib2**, **ib3**, ou **ib4**. Podem ser fixados o **SEED** e os flags específicos nas opções **IBL_SEED** e **IBL_FLAGS**. O executável **ibl** deve estar no path de trabalho.

O IBL é um sistema de pesquisa e não é muito robusto [Aha 92]. Não verifica quando seus limites são excedidos e às vezes ultrapassa os limites sobre os arrays, corrompendo a memória e normalmente fazendo um *dumping* no arquivo core. Se isso acontecer, a causa mais provável é que alguma constante em datastructures.h é muito pequena. Na versão distribuída na MLC^{++} foram aumentados os limites para 200 atributos e 10.000 instâncias.

Existem alguns problemas que foram descobertos ao se trabalhar com o IBL em muitos arquivos de dados:

1. Se os arquivos são muito pequenos (poucas instâncias), a precisão do conjunto de teste não é reportada (por exemplo, *dataset* soybean-small).
2. O programa estoura a memória certas vezes. Requer mais de 150MB para o *dataset* mushroom.
3. Não manipula espaços em branco nos valores de atributos, os quais podem causar problemas em alguns arquivos (isto poderia ser observado na conversão ao código MLC^{++}, mas é muito raro).

Contacte David Aha (aha@aic.nrl.navy.mil) para perguntas, problemas, e pedidos de código-fonte.

C4.5 e variantes
O algoritmo C4.5 foi introduzido por J. Ross Quinlan [Quinlan 93] para induzir modelos de classificação a partir de um conjunto de dados. C4.5 é uma extensão do algoritmo original ID3 e permite atributos não avaliados, atributos contínuos, atributos discretos, derivação de regras, entre outras.

O sistema C4.5 consiste de quatro programas principais:

- gerador de Árvore de Decisão (**c4.5**);
- gerador de Regras de Produção (**c4.5rules**);
- interpretador de Árvore de Decisão (**consult**);
- interpretador de Regras de Produção (**consultr**).

Na construção de uma árvore de decisão é necessário um conjunto de dados de treinamento, onde cada registro deve ter a mesma estrutura, consistindo de vários pares de atributo/valor. Um desses atributos representa a meta do registro, por exemplo: o atributo que seja mais relevante. O problema é determinar uma árvore de decisão com base nas respostas das questões sobre os atributos não-metas. O atributo meta usualmente trabalha com valores {true, false} ou {good, bad}, ou algo equivalente.

O programa utiliza dois métodos (árvore de decisão ou conjunto de regras de produção) para permitir o uso da informação imprecisa sobre os valores dos atributos, e caracterizar o grau de certeza associado ao predicado mais semelhante entre a classe e suas alternativas.

O **c4.5**, **c4.5unpruned**, e **c4.5rules** são interfaces para o C4.5 e precisam ser instalados por conta própria. Eles requerem que o **c4.5** e **c4.5rules** estejam no path, e usem o arquivo $MLCDIR/c45test.awk. O C4.5 pode ser comprado com o livro C4.5 de Ross Quinlan (ISBN: 1-55860-240). Os *patches* podem ser obtidos através de um ftp anônimo a ftp.cs.su.oz.au, no diretório pub/ml, arquivo patch.tar.Z.

É possível modificar o comportamento *default* (opções) do C4.5 substituído a opção **FLAGS_C45** (o *default* é **-u -f %s**) **%s** pelo nome do arquivo requerido pelo C4.5. O **c4.5rules** executa **c4.5** e podem ser fixadas as opções apropriadas usando **C45R_FLAGS1** e **C45R_FLAGS2**. A menos que se use a versão 7 do **c4.5rules**, o valor de retorno dá um erro, porque foi adicionada uma declaração "echo dummy" para **C45R_FLAGS2**.

A opção **C45_STATS** permite gerar estatísticas sobre as árvores de decisão geradas, incluindo o número de nós e o número de atributos. É principalmente útil quando se quer ver estas estatísticas para 10-fold CV, ao invés de uma única execução. A opção **MAX_C45_TRIES** determina o número de vezes para chamar o C4.5 no caso de existir um problema executando o C4.5 ou analisando gramaticalmente sua saída.

CART
O CART é um sistema para montar árvores de decisão, e foi desenvolvido por estatísticos. Como regra geral, técnicas estatísticas tendem a focar tarefas em que todos os atributos têm valores contínuos ou ordinais.

A versão comercial do CART [Breiman 84] foi conectada a partir do sistema de Salford. Na MLC^{++} é usada a versão 2.0 com um limite de memória de 50.000.000 4-palavras na área de trabalho (200MB). O parâmetro **CVLEARN** foi aumentado para ser maior do que a medida do conjunto de treinamento, só que a estimativa de *cross-validation* mais precisa seria usando a fase de poda.

CN2
O CN2 é um indutor externo que faz interface com a MLC^{++}, e está descrito em [Clark 89] e [Clark 91]. O CN2 é um algoritmo de indução de regras, e é um algoritmo de aprendizado não-incremental que recebe como entrada um conjunto de exemplos e gera com saída um conjunto de regras de produção para classificar esses exemplos, além de permitir avaliar a exatidão desse conjunto. Este algoritmo é designado para uma eficiente geração de regras de indução do tipo "if...then" simples, e compreensíveis em domínios onde ruídos nos dados de entrada podem estar presentes.

Ver http://www.cs.utexas.edu/users/pclark/software.html ou contactos pclark@cs.utexas.edu para mais informação.

Decision Tables

O Decision tables é um dos mais simples indutores. Ele armazena uma tabela de todas as instâncias e prediz futuras instâncias de acordo com ela. Se uma instância não é encontrada, a tabela majoritária prediz a classe majoritária da tabela e a tabela não majoritária retorna um "unknown" (não encontrado no conjunto de teste). Quando acoplada com a **Feature Subset Selection**, é fornecido um poderoso indutor para dados discretos [Kohavi 95c]. Se a discretização é feita, isto é também útil para dados com atributos contínuos. Por exemplo, para executar discretização com **Decision Tables e Feature Subset Selection**, pode-se definir as seguintes opções trabalhando no *dataset* crx, como mostra o exemplo 12:

Exemplo 12:

```
setenv INDUCER disc-filter
setenv DISCF_INDUCER FSS
setenv DISCF_FSS_INDUCER table-majority
setenv DISPLAY_CONFUSION_MAT yes
setenv OPTION DISP_CONFUSION_MAT ascii
setenv DATAFILE crx
Inducer
```

A saída será:

```
MLC++ Debug level is 0, log level is 1
OPTION PROMPTLEVEL = required-only
OPTION INDUCER = disc-filter
OPTION INDUCER_NAME = disc-filter
OPTION DISCF_INDUCER = FSS
OPTION DISCF_INDUCER_NAME = FSS
OPTION DISCF_FSS_INDUCER = table-majority
OPTION DISCF_FSS_INDUCER_NAME = table-majority
OPTION DISCF_FSS_DOT_FILE = DISCF_FSS.dot
OPTION DISCF_FSS_SEARCH_METHOD = best-first
OPTION DISCF_FSS_SHOW_TEST_SET_PERF = best-only
OPTION DISCF_FSS_EVAL_LIMIT = 0
OPTION DISCF_FSS_MAX_STALE = 5
OPTION DISCF_FSS_EPSILON = 0.001
OPTION DISCF_FSS_PERF_ESTIMATOR = automatic
OPTION DISCF_FSS_CV_WEIGHT_THRESHOLD = 5000
OPTION DISCF_FSS_ERROR_TYPE = error
OPTION DISCF_FSS_PERF_EST_SEED = 7258789
OPTION DISCF_FSS_ERROR_TRIM = 0
OPTION DISCF_FSS_CV_FOLDS = 10
OPTION DISCF_FSS_CV_TIMES = 1
OPTION DISCF_FSS_CV_FRACT = 1
OPTION DISCF_FSS_HO_TIMES = 1
OPTION DISCF_FSS_HO_NUMBER = 0
OPTION DISCF_FSS_HO_PERCENT = 0.666666666667
OPTION DISCF_FSS_USE_COMPOUND = true
OPTION DISCF_FSS_CMPLX_PENALTY = 0
OPTION DISCF_FSS_DIRECTION = forward
OPTION DISCF_DISC_TYPE = entropy
OPTION DISCF_DISC_NUM_INTR = Algo-heuristic
```

```
OPTION DISCF_LBOUND_MIN_SPLIT = 0
OPTION DISCF_UBOUND_MIN_SPLIT = 25
OPTION DISCF_MIN_SPLIT_WEIGHT = 0
OPTION DISCF_STOPPING_CRIT = MDL
OPTION DATAFILE = crx
OPTION NAMESFILE = crx.names
OPTION WEIGHT_IS_ATTRIBUTE = true
OPTION REMOVE_UNKNOWN_INST = false
OPTION CORRUPT_UNKNOWN_RATE = 0
Reading crx.data.... done.
OPTION TRAIN_WITHOUT_LOSS = false
OPTION TESTFILE = crx.test
Reading crx.test.. done.
OPTION COMPUTE_LOG_LOSS = false
Discretizing 6 attributes: 1 2 3 4 5 6 done.
Discretization vector:*, 2, 2, *, *, *, *, 2, *, *, 2, *, *, 2, 2
New best node (1 evals) #0[]: error: 44.29% +- 2.43% (32.65% - 55.10%).
Test Set: 45.00% +- 3.53% [38.26% - 51.92%].  Bias: -0.71% cost: 10
complexity: 0
.................
New best node (19 evals) #9[8]: error: 13.47% +- 2.00% (6.12% - 24.49%).
Test Set: 17.00% +- 2.66% [12.43% - 22.82%].  Bias: -3.53% cost: 10
complexity: 1
.................
New best node (35 evals) #32[1, 4, 8]: error: 12.24% +- 1.85% (6.12% - 22.45%).
Test Set: 18.00% +- 2.72% [13.29% - 23.91%].  Bias: -5.76% cost: 10
complexity: 3
.............
New best node (50 evals) #48[1, 3, 4, 8, 9, 12]: error: 11.22% +- 1.83% (2.04%
  - 20.41%).
Test Set: 17.50% +- 2.69% [12.86% - 23.36%].  Bias: -6.28% cost: 10
complexity: 6
.....................................................................
New best node (113 evals) #103[3, 4, 8, 9, 12, 14]: error: 11.02% +- 1.86%
  (2.04% - 20.41%).
Test Set: 17.50% +- 2.69% [12.86% - 23.36%].  Bias: -6.48% cost: 10
complexity: 6
.............
New best node (128 evals) #125[3, 4, 8, 9, 12, 13, 14]: error: 10.61% +- 1.66%
  (2.04% - 20.41%).
Test Set: 17.50% +- 2.69% [12.86% - 23.36%].  Bias: -6.89% cost: 10
complexity: 7
.............
New best node (142 evals) #132[3, 4, 6, 8, 9, 12, 13, 14]: error: 9.80% +-
  1.69% (0.00% - 16.33%).
Test Set: 17.00% +- 2.66% [12.43% - 22.82%].  Bias: -7.20% cost: 10
complexity: 8
.....................................................................
Final best node #132[3, 4, 6, 8, 9, 12, 13, 14]: error: 9.80% +- 1.69% (0.00% -
  16.33%).
Test Set: 17.00% +- 2.66% [12.43% - 22.82%].  Bias: -7.20% cost: 10
complexity: 8
Expanded 16 nodes
Classifying (% done): 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% done.
OPTION DISP_LIFT_CURVE = false
```

```
OPTION DISPLAY_STRUCT = none
OPTION DISP_MISCLASS = false
Number of training instances: 490
Number of test instances: 200.  Unseen: 179,  seen 21.
Number correct: 166.  Number incorrect: 34
Generalization error: 17.318%.  Memorization error: 14.286%
Error: 17.000% +- 2.663% [12.428% - 22.816%]
Average Normalized Mean Squared Error: 17.000%
Average Normalized Mean Absolute Error: 17.000%
Displaying confusion matrix...
  (a)     (b)     <-- classified as
-------- --------
  69.00    21.00    (a): class yes
  13.00    97.00    (b): class no
OPTION DISP_CONF_SCATTERGRAM = false
```

Incremente o **LOGLEVEL** para **2**, a fim de verificar o seu progresso. É possível usar o utilitário **project** no nó final para estudar os atributos selecionados isoladamente.

Algoritmos Instance Based
O IB é um indutor baseado em instâncias. É um algoritmo robusto e bom, mas lento quando há muitos atributos [Aha 92], [Wettschereck 94].

A opção **NUM_NEIGHBORS** determina o número de vizinhos a serem usados. Em domínios discretos, muitos vizinhos terão a mesma distância, assim a opção **NNKVALUE** determina se o número de vizinhos seriam vizinhos de distâncias diferentes, com as instâncias amarrar-quebrar (tie-breaking) sendo consideradas como um único vizinho. A opção **NORMALIZATION** determina se os dados deveriam ser normalizados de acordo com os valores extremos, com o intervalo *interquartise* (25% a 75%), ou se nenhuma normalização deveria acontecer. A opção **NEIGHBOR_VOTE** determina se os vizinhos votam igualmente ou com que poder votam inversamente proporcional à distância entre eles. A opção **MANUAL_WEIGHTS** permite fixar os pesos manualmente para cada atributo.

ID3 e MC4
O ID3 é um algoritmo de árvore de decisão básico sem poda baseado em [Quinlan 86]. O MC4 inclui uma poda semelhante ao C4.5 [Quinlan 93]. O MC4 deveria dar resultados semelhantes ao C4.5 com exceção da manipulação dos valores desconhecidos que é diferente, mas ambos são o mesmo algoritmo mas com diferentes parâmetros *default*.

A opção **MIN_SPLIT_WEIGHT** é a porcentagem mínima de instâncias de treinamento dividida pelo número de classes que são requeridas para descer, até pelo menos, dois ramos em um determinado nó. A opção **LBOUND_MIN_SPLIT** e **UBOUND_MIN_SPLIT** delimitam este número. Isto provê um mecanismo semelhante ao C4.5 para manipulação de divisões. A determinação do limite é como segue. Primeiro, o número mínimo de instâncias é calculado usando a opção **WEIGHT** (isto é calculado como um número de ponto flutuante), então se este número for maior que o **UBOUND**, então se torna o **UBOUND**. Finalmente, se este número for menor do que o **LBOUND**, estão ele se torna **LBOUND**.

O **ID3_DEBUG** acrescenta informações a cada nó, indicando o número de instâncias, entropia, e informação mútua. Fixando a **DISPLAY_STRUCT** para *dot* e vendo o resultado no arquivo Inducer.dot em *dot*/*dotty* depois de executar o utilitário **Inducer** com o indutor **ID3**.

A opção **ID3_UNKNOWN_EDGES** determina se uma extremidade é gerada para manipular valores desconhecidos. Se esta opção for **false**, se obtêm uma vista da árvore mais agradável, mas falhará se existem instâncias com valores desconhecidos. Note que o C4.5 têm um melhor

mecanismo para manipular valores desconhecidos. A opção **ID3_SPLIT_BY** determina o critério de divisão. Usa-se a informação mútua regular ou a informação mútua normalizada pelo número de valores. Um exemplo usando ID3 com o *dataset* crx é mostrado a seguir, no exemplo 13:

Exemplo 13:

```
setenv DATAFILE crx.data                    #estabelecimento do dataset
setenv TESTFILE crx.test
setenv INDUCER ID3                  #escolha do ID3
setenv ID3_UNKNOWN0_EDGES no  #não trabalha com extremos desconhecidos
setenv DISP_CONFUSION_MAT ascii#mostra a matriz de confusão
setenv DISPLAY_STRUCT dotty         #mostra a árvore usando dotty
Inducer

dot -Tps -Gpage=8.5,11 -Gmargin=0,0 Inducer.dot -o crx.ps
```

A saída será:

```
MLC++ Debug level is 0, log level is 1
OPTION PROMPTLEVEL = required-only
OPTION INDUCER = ID3
OPTION INDUCER_NAME = ID3
OPTION ID3_MAX_LEVEL = 0
OPTION ID3_PRUNING_METHOD = Confidence
OPTION ID3_PRUNING_FACTOR = 0
OPTION ID3_LBOUND_MIN_SPLIT = 1
OPTION ID3_UBOUND_MIN_SPLIT = 25
OPTION ID3_MIN_SPLIT_WEIGHT = 0
OPTION ID3_NOMINAL_LBOUND_ONLY = false
OPTION ID3_DEBUG = false
OPTION ID3_UNKNOWN_EDGES = true
OPTION ID3_SPLIT_BY = normalized-mutual-info
OPTION ID3_PARENT_TIE_BREAKING = true
OPTION ID3_ADJUST_THRESHOLDS = false
OPTION ID3_CONT_MDL_ADJUST = false
OPTION ID3_SMOOTH_INST = 0
OPTION ID3_LEAF_DIST_TYPE = all-or-nothing
OPTION ID3_EVAL_METRIC = error
OPTION DATAFILE = crx
OPTION NAMESFILE = crx.names
OPTION WEIGHT_IS_ATTRIBUTE = true
OPTION REMOVE_UNKNOWN_INST = false
OPTION CORRUPT_UNKNOWN_RATE = 0
Reading crx.data.... done.
OPTION TRAIN_WITHOUT_LOSS = false
Tree has 129 nodes, 75 leaves, and 11 attributes.
Histogram for all nodes: 1, 2, 6, 12, 13, 16, 17, 18, 10, 4, 2, 4, 2, 8, 8
OPTION TESTFILE = crx.test
Reading crx.test.. done.
OPTION COMPUTE_LOG_LOSS = false
Classifying (% done): 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% done.
OPTION DISP_MISCLASS = false
Number of training instances: 490
Number of test instances: 200.  Unseen: 200,  seen 0.
Number correct: 145.  Number incorrect: 55
```

```
Generalization error: 27.500%.  Memorization error: unknown
Error: 27.500% +- 3.165% [21.780% - 34.068%]
Average Normalized Mean Squared Error: 27.500%
Average Normalized Mean Absolute Error: 27.500%

Displaying confusion matrix...
  (a)     (b)     <-- classified as
-------- --------
  66.00    24.00    (a): class yes
  31.00    79.00    (b): class no
OPTION DISP_CONF_SCATTERGRAM = false

OPTION DISP_LIFT_CURVE = false
OPTION BACKFIT = false
OPTION DISPLAY_STRUCT = none
OPTION CAT_NAME =
Average nodes for trees: 129.00 over 1 trees.
Average attributes for trees: 11.00.
```

A saída gráfica do resultado do arquivo crx.ps gerada pelo ID3 sobre o *dataset* crx é mostrada na Figura 13 através do visualizador *dotty*.

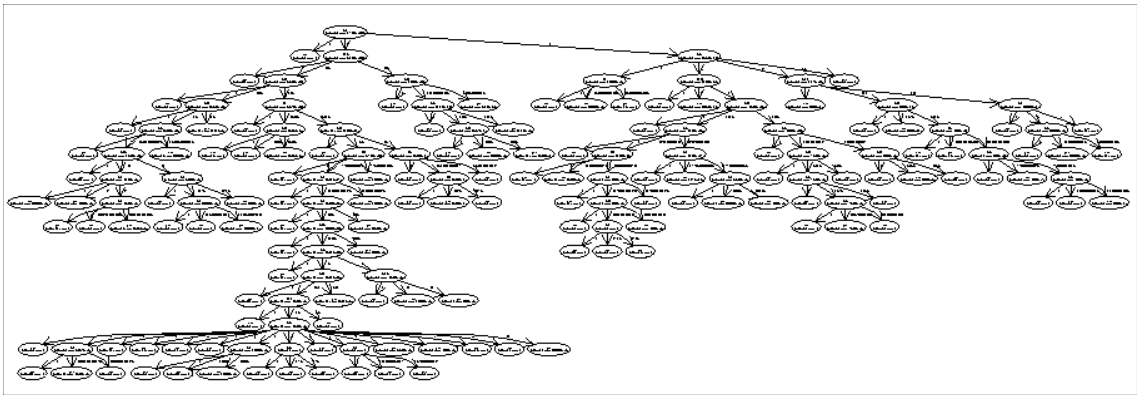

Figura 13: Árvore de decisão gerada pelo ID3 sobre o *dataset* crx.

HOODG HOODG /List-HOODG

O HOODG (Gráfico de Decisão não Evidentes) é um indutor para construir gráficos de decisão não evidentes de baixo para cima [Kohavi 94b], [Kohavi 94c] e não manipula valores desconhecidos. O HOODG tolera características fracamente relevantes ou irrelevantes, o que justifica que se deve usar a **Feature Subset Selection**. O HOODG também requer dados discretizados, portanto o **disc-filter** deve ser usado. O exemplo 14 apresenta uma execução com **disc-filter** e uma saída *dotty* mostrada na Figura 14.

Exemplo 14:

```
setenv DATAFILE crx.data
setenv TESTFILE crx.test
setenv DRIBBLE false
setenv INDUCER disc-filter          #loodg só pode trabalhar sobre dados discretos
setenv DISCF_INDUCER fss            #Feature Subset Selection
setenv DISCF_FSS_INDUCER HOODG
setenv DISCF_FSS_MAX_STALE 3        #quanto deve pesquisar antes de parar
setenv DISCF_FSS_CV_TIMES 0         #heurística para executar cv várias vezes
```

```
setenv DISCF_FSS_CV_FOLDS 5       #5-fold CV
setenv DISCF_FSS_CMPLX_PENALTY 0.0001 #penalidade pequena para mais
                                      # características
setenv DISCF_FSS_SHOW_REAL_ACC never  #caso contrário, é um indutor base
setenv REMOVE_UNKNOWN_INST yes
setenv OPTION DISP_CONFUSION_MAT ascii
setenv DISPLAY_STRUCT dotty               #saída dotty do gráfico final
setenv DISCF_FSS_DOT_FILE crx_DISCF_FSS.dot
setenv INDUCER_DOT crx.dot                #onde manter a saída dot
Inducer
```

A saída será:

```
MLC++ Debug level is 0, log level is 1
OPTION PROMPTLEVEL = required-only
OPTION INDUCER = disc-filter
OPTION INDUCER_NAME = disc-filter
OPTION DISCF_INDUCER = fss
OPTION DISCF_INDUCER_NAME = FSS
OPTION DISCF_FSS_INDUCER = hoodg
OPTION DISCF_FSS_INDUCER_NAME = HOODG
OPTION DISCF_FSS_DISCF_DISC_TYPE = entropy
OPTION DISCF_FSS_DISCF_DISC_NUM_INTR = Algo-heuristic
OPTION DISCF_FSS_DISCF_LBOUND_MIN_SPLIT = 0
OPTION DISCF_FSS_DISCF_UBOUND_MIN_SPLIT = 25
OPTION DISCF_FSS_DISCF_MIN_SPLIT_WEIGHT = 0
OPTION DISCF_FSS_DISCF_STOPPING_CRIT = MDL
OPTION DISCF_FSS_DOT_FILE = crx_DISCF_FSS.dot
OPTION DISCF_FSS_SEARCH_METHOD = best-first
OPTION DISCF_FSS_SHOW_TEST_SET_PERF = best-only
OPTION DISCF_FSS_EVAL_LIMIT = 0
OPTION DISCF_FSS_MAX_STALE = 3
OPTION DISCF_FSS_EPSILON = 0.001
OPTION DISCF_FSS_PERF_ESTIMATOR = automatic
OPTION DISCF_FSS_CV_WEIGHT_THRESHOLD = 5000
OPTION DISCF_FSS_ERROR_TYPE = error
OPTION DISCF_FSS_PERF_EST_SEED = 7258789
OPTION DISCF_FSS_ERROR_TRIM = 0
OPTION DISCF_FSS_MAX_TIMES = 5
OPTION DISCF_FSS_HO_TIMES = 1
OPTION DISCF_FSS_HO_NUMBER = 0
OPTION DISCF_FSS_HO_PERCENT = 0.666666666667
OPTION DISCF_FSS_USE_COMPOUND = true
OPTION DISCF_FSS_CMPLX_PENALTY = 0.0001
OPTION DISCF_FSS_DIRECTION = forward
OPTION DISCF_DISC_TYPE = entropy
OPTION DISCF_DISC_NUM_INTR = Algo-heuristic
OPTION DISCF_LBOUND_MIN_SPLIT = 0
OPTION DISCF_UBOUND_MIN_SPLIT = 25
OPTION DISCF_MIN_SPLIT_WEIGHT = 0
OPTION DISCF_STOPPING_CRIT = MDL
OPTION DATAFILE = crx.data
OPTION NAMESFILE = crx.names
OPTION WEIGHT_IS_ATTRIBUTE = true
OPTION REMOVE_UNKNOWN_INST = yes
OPTION CORRUPT_UNKNOWN_RATE = 0
```

```
Reading crx.data.... Removed 25 instances.
Discretization vector:*, 2, 2, *, *, *, *, 2, *, *, 2, *, *, 3, 2

OPTION DISP_LIFT_CURVE = false
OPTION DISPLAY_STRUCT = dotty
OPTION DISP_MISCLASS = false
Number of training instances: 465
Number of test instances: 188.  Unseen: 174,  seen 14.
Number correct: 157.  Number incorrect: 31
Generalization error: 16.092%.  Memorization error: 21.429%
Error: 16.489% +- 2.714% [11.867% - 22.454%]
Average Normalized Mean Squared Error: 16.489%
Average Normalized Mean Absolute Error: 16.489%
OPTION DISP_CONFUSION_MAT = ascii

Displaying confusion matrix...
  (a)     (b)     <-- classified as
-------- --------
  76.00    8.00   (a): class yes
  23.00   81.00   (b): class no
OPTION DISP_CONF_SCATTERGRAM = false
```

Nota: este tipo de categorizador não suporta persistência

Invocar à **Feature Subset Selection** pode tornar esse indutor muito lento. Uma aproximação alternativa muito mais rápida é usar entropia para achar um bom conjunto de atributos. O indutor **List-HOODG** encapsula tudo o que é necessário para a pesquisa. A opção **AO_GROW_CONF_RATIO**, que é a proporção de instâncias não-classificadas em um dado nível, determina o critério de parada. Assim, um número mais alto significa que menos atributos serão usados. Ver [Kohavi 95e] para mais informação.

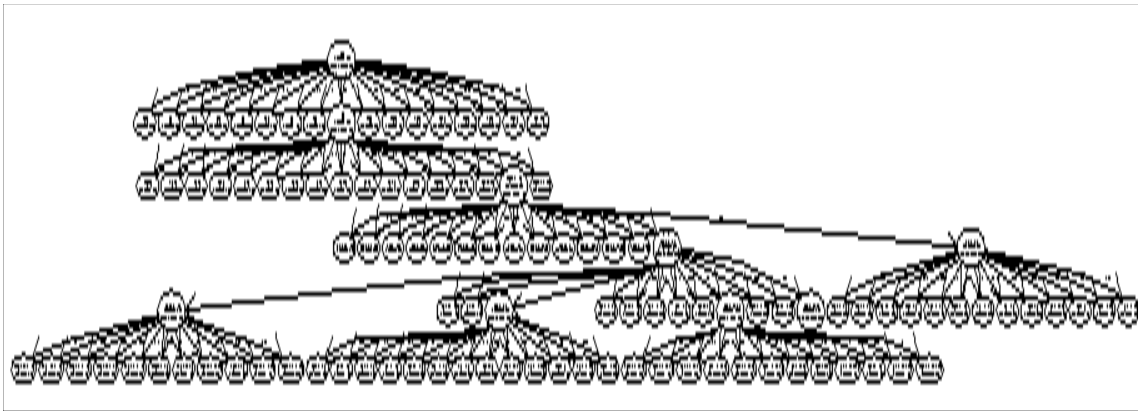

Figura 14: Gráfico de Decisão não Evidente (HOODG) para crx.

Naive Bayes

O indutor *Naive-Bayes* [Langley 92], [Duda 73], [Good 65] provê um modelo que determina o valor de um dado desconhecido baseado nas probabilidades observadas das classes dada uma instância. Os atributos (características de cada classe) são assumidos para serem independentes, uma suposição que não sempre é verdade, mas o algoritmo é todavia muito robusto para violações dessa suposição.

As probabilidades para atributos nominais (discretos) são estimadas através de contagens, e uma distribuição Gaussiana é usada para atributos contínuos. A probabilidade para zero contagens é

1/2m para *m* instâncias. As probabilidades para atributos contínuos são estimadas assumindo uma distribuição normal para cada atributo e classe. Valores desconhecidos dentro da instância testada não são levados em consideração (equivalente a marginalizá-los).

Os melhores resultados são comumente alcançados discretizando os atributos contínuos. O indutor **disc-naive-bayes** provê este passo de preprocessamento encadeando o **disc-filter-inducer** ao indutor **naive-bayes** [Dougherty 95]. Além disso, melhorias podem ser alcançadas executando a **Feature Subset Selection** [Langley 94], [Kohavi 95c] como é mostrado a seguir no exemplo 15:

Exemplo 15:

```
setenv INDUCER disc-filter
setenv DISCF_INDUTOR FSS
setenv DISCF_FSS_INDUTOR naive
setenv DISCF_FSS_CMPLX_PENALTY 0.001
setenv DISCF_FSS_CV_TIMES 0
setenv DISCF_FSS_ACC_ESTIMATOR cv
setenv DISCF_FSS_CV_FOLDS 5
setenv DISCF_FSS_DIRECTION backward
Inducer
```

A saída será:

```
MLC++ Debug level is 0, log level is 1
OPTION PROMPTLEVEL = required-only
OPTION INDUCER = disc-filter
OPTION INDUCER_NAME = disc-filter
OPTION DISCF_INDUCER = FSS
OPTION DISCF_INDUCER_NAME = FSS
OPTION DISCF_FSS_INDUCER = naive
OPTION DISCF_FSS_INDUCER_NAME = naive-bayes
OPTION DISCF_FSS_NB_LAPLACE_CORRECTION = false
OPTION DISCF_FSS_NB_NO_MATCHES_FACTOR = 0
OPTION DISCF_FSS_NB_UNKNOWN_IS_VALUE = no
OPTION DISCF_FSS_NB_EVIDENCE_PROJECTION = false
OPTION DISCF_FSS_DOT_FILE = DISCF_FSS.dot
OPTION DISCF_FSS_SEARCH_METHOD = best-first
OPTION DISCF_FSS_SHOW_TEST_SET_PERF = best-only
OPTION DISCF_FSS_EVAL_LIMIT = 0
OPTION DISCF_FSS_MAX_STALE = 3
OPTION DISCF_FSS_EPSILON = 0.001
OPTION DISCF_FSS_PERF_ESTIMATOR = automatic
OPTION DISCF_FSS_CV_WEIGHT_THRESHOLD = 5000
OPTION DISCF_FSS_ERROR_TYPE = error
OPTION DISCF_FSS_PERF_EST_SEED = 7258789
OPTION DISCF_FSS_ERROR_TRIM = 0
OPTION DISCF_FSS_CV_FOLDS = 5
OPTION DISCF_FSS_CV_TIMES = 0
OPTION DISCF_FSS_CV_FRACT = 1
OPTION DISCF_FSS_DES_STD_DEV = 0.01
OPTION DISCF_FSS_MAX_TIMES = 5
OPTION DISCF_FSS_HO_TIMES = 1
OPTION DISCF_FSS_HO_NUMBER = 0
OPTION DISCF_FSS_HO_PERCENT = 0.666666666667
OPTION DISCF_FSS_USE_COMPOUND = true
```

```
OPTION DISCF_FSS_CMPLX_PENALTY = 0.001
OPTION DISCF_FSS_DIRECTION = backward
OPTION DISCF_DISC_TYPE = entropy
OPTION DISCF_DISC_NUM_INTR = Algo-heuristic
OPTION DISCF_LBOUND_MIN_SPLIT = 0
OPTION DISCF_UBOUND_MIN_SPLIT = 25
OPTION DISCF_MIN_SPLIT_WEIGHT = 0
OPTION DISCF_STOPPING_CRIT = MDL
OPTION DATAFILE = crx.data
OPTION NAMESFILE = crx.names
OPTION WEIGHT_IS_ATTRIBUTE = true
OPTION REMOVE_UNKNOWN_INST = yes
OPTION CORRUPT_UNKNOWN_RATE = 0
Reading crx.data.... Removed 25 instances.
OPTION TRAIN_WITHOUT_LOSS = false
OPTION TESTFILE = crx.test
Reading crx.test.. Removed 12 instances.
OPTION COMPUTE_LOG_LOSS = false
Discretization vector:*, 2, 2, *, *, *, *, 2, *, *, 2, *, *, 3, 2
OPTION DISP_LIFT_CURVE = false
OPTION DISPLAY_STRUCT = dotty
OPTION DISP_MISCLASS = false
Number of training instances: 465
Number of test instances: 188.  Unseen: 174,  seen 14.
Number correct: 157.  Number incorrect: 31
Generalization error: 16.092%.  Memorization error: 21.429%
Error: 16.489% +- 2.714% [11.867% - 22.454%]
Average Normalized Mean Squared Error: 16.489%
Average Normalized Mean Absolute Error: 16.489%
OPTION DISP_CONFUSION_MAT = ascii
Displaying confusion matrix...
  (a)     (b)     <-- classified as
-------- --------
  73.00    11.00    (a): class yes
  20.00    84.00    (b): class no
OPTION DISP_CONF_SCATTERGRAM = false
```

Nota: São usados *datasets* que estão no formato da MLC^{++} e que são bem parecidos ao formato do C4.5 [Quinlan 93], onde as pequena diferenças são que a MLC^{++} não suporta **ignore** no arquivo de nomes. O utilitário **project** pode ser usada em vez de usar **ignore**. Cada *dataset* pode incluir um arquivo de nomes que descreve como analisar gramaticalmente os dados, um arquivo de dados contendo os dados, e um arquivo de teste opcional para calcular a precisão.

Nearest-neighbor

O Nearest-neighbor é um indutor baseado no paradigma estatístico, é conhecido por depender fortemente da distância métrica usada entre os atributos do conjunto de dados [Dasarathy 90] [Aha 92] [Wettschereck 94]. Tem fortes propriedades asimptióticas, tal que para conjuntos de dados finitos elas são fortemente afetadas pelo “espaço de dimensionalidade”, onde incrementando o número de características pode requerer exponencialmente mais instâncias para manter o mesma taxa de erro. Um problema comum ao usar distâncias métricas Euclidiana, é que dão igual peso para todas as características. Taxas de erro para problemas práticos com algumas características podem afetar significativamente se as características com relevância fraca ou não relevância estiver participando com pouco peso dentro da métrica de distância. Dentro da MLC++, nearest-neighbor conta com opções para fixar o peso, normalizações e edição.

Neural Network: Aspirin/MIGRAINES
O ambiente da rede neural de Aspirin/MIGRAINES trabalha na versão 6.0 do dia 29 agosto 1996 usando o padrão backpropagation [Hertz 91], [Rumelhart 86]. A interface é muito simples: os atributos nominais são convertidos a codificação local e a rede construída contém três camadas com uma camada escondida que contém *k* nós. *k* é automaticamente fixado ao número de entradas e saídas de nós divididos por dois, que é uma regra comumente chamada de dedo polegar. Embora esta regra é criticada freqüentemente (por exemplo, ftp://ftp.sas.com/pub/neural/FAQ.html), não há nenhum outro candidato bom para selecionar este número automaticamente e para propósitos de comparação se tenta evitar afinação manual.

OC1
Oblique **C**lassifier **1** é um sistema que induz árvores de decisão [Murthy 94]., e foi especialmente projetado para aplicações onde as instâncias têm *features* que assumem valores numéricos. O sistema OC1 constrói árvores de decisão que contém combinações lineares de um ou mais atributos em cada nó interno da árvore; essas árvores particionam então o espaço de exemplos em hiperplanos. OC1 também constrói árvores de decisão usuais, que contém testes de apenas um atributo em cada nó.

OC1 é um indutor externo que conecta com o OC1 versão 3. A fonte pode ser obtida da Johns Hopkins University, Departament of Computer Science. A mais recente versão do sistema OC1 pode ser obtida diretamente por FTP anônimo de blaze.cs.jhu.edu, no diretório pub/oc1.

As questões devem ser dirigidas a seus autores, Sreerama Murthy, Steven Salzberg e Simon Kasif (E-mail: lastname@cs.jhu.edu URL: http://www.cs.jhu.edu/lastname). Quando utiliza o software OC1 no contexto de quaisquer publicações, por favor referencie a [Murthy 94].

As opções seguintes são suportadas:

OC1_SEED: fixa o ponto de partida.
OC1_AXIS__PARALLEL_ONLY: força as divisões paralelas do eixo.
OC1_CART_LINEAR_COMBINATION_MODE: força a CART-like a fazer divisões [Breiman 84].
OC1_PRUNING_RATE: fixa a taxa de poda.

OneR
O OneR é um classificador simples que cria uma regra única, por exemplo, uma regra baseada no valor de um único atributo [Holte 93]. A opção **MIN_INST** é o número mínimo de instâncias para um intervalo de discretização. Holte recomenda o valor seis para a maioria dos *datasets*. O OneR está implementado atualmente somente como um indutor base.

O OneR mostra que é fácil adquirir uma precisão razoável em muitas tarefas olhando simplesmente um atributo. Ao contrário de indicações comuns e falta de interpretações relativas dos resultados de Holte, o indutor é *significativamente* inferior ao C4.5. A precisão média do OneR para os *datasets* testados por Holte é de 5.7% abaixo do C4.5 [Holte 93]. A taxa de erro do C4.5 tem um erro de 14.07% e o OneR faz 40% mais erros do que o C4.5.

PEBLS
PEBLS é um indutor externo que faz interface com o Sistema de Aprendizado Paralelo Baseado em Exemplos versão 2.0 de [Cost 93]. Contacte com salzberg@cs.jhu.edu para mais informação. Os códigos fontes podem ser obtidos da URL ftp://ftp.cs.jhu.edu/pub/pebls /.

Foram feitas várias mudanças no código fonte original: o número de máximo de instâncias foi aumentado de 110 para 5.000, e o número máximo de caracteres em nomes de classe foi aumentado para 20 (config.h). O programa foi mudado para retornar estado 0 em saída com execução bem sucedida.

Opções suportadas são: **PEBLS_DISCRETIZATION_LEVELS; PEBLS_NEAREST_ NEIGHBORNS** e **PEBLS_VOTING_SCHEME**.

Perceptron e Winnow
O Perceptron e Winnow são indutores que constróem discriminadores lineares. Eles são apenas capazes de manipular atributos contínuos sem valores desconhecidos e problemas de duas classes. Para dados discretos, pode-se usar o utilitário **conv** para converter os atributos de entrada em codificação local ou codificação binária. A opção **REMOVE_UNKNOWN_INST** pode ser usada para remover instâncias com valores desconhecidos.

O Perceptron usa a regra de correção de erro [equação 5.19 em Hertz 91]. O Winnow usa o algoritmo descrito em [Littlestone 88]. Todos os atributos são normalizados para o intervalo [0 - 1] usando normalização extrema (mapa de valores menores de 0 e maiores de 1). Para diferentes tipos de normalização pode-se usar o indutor **cont-filter** como um preprocessador ou executar o utilitário **conv**. A razão para esta normalização é que o Winnow sobrecarrega realmente rápido quando eleva os números a potências. O Perceptron e o Winnow estão limitados para duas classes de problemas que reduzem a utilidade deles em alguns problemas.

T2
O T2 é uma árvore de decisão de dois níveis que minimiza o número de erros e discretiza atributos contínuos [Auer 95]. Requer grande quantidade de memória quando se tem muitas classes.

Alguns indutores para a MLC^{++} ainda estão sendo desenvolvidos ou têm uso limitado, são eles:

CatDT: constrói árvores de decisão com categorizadores onde é possível escolher as folhas. Requer que os indutores suportem cópias, o que não é feito por muitos (por exemplo, naive-bayes).

Const: const prediz uma classe constante, a classe majoritária no conjunto de treinamento. A estimativa de precisão do indutor const é comumente chamada de precisão *baseline*.

EODG: é um indutor para construir gráficos de decisão não evidentes top-down [Kohavi 95e]. Não pode manipular valores desconhecidos.

Lazy Decision Trees: um algoritmo para construir a "melhor" árvore de decisão para toda instância de teste descrita em [Friedman 96].

NULL: sempre prediz o desconhecido e assim adquire 0% de precisão. É usado internamente, mas pode ser usado com o utilitário Inducer e fixar **DISP_CONFUSION_MAT** para validar a vista da distribuição dos níveis. O utilitário **info** é provavelmente o melhor modo de adquirir estatísticas básicas sobre um arquivo de dados.

Option Decision Tree: as árvores de decisão têm nós de opção que permitem várias divisões opcionais, que são votadas com ajuda de especialistas durante a classificação [Kohavi 97a].

Ordem-fss: procura por um atributo ordenado.

Ripper: é um sistema de aprendizado de regras descrito por [Cohen 95].

APÊNDICE B

MIndUtil utilizando MLC^{++}

MIndUtil é o programa que possibilita o software *Mineset®* utilizar a biblioteca MLC^{++}. O *Mineset®* foi desenvolvido pela *Silicon Graphics®* para ajudar na extração de conhecimento nas bases de dados (Data Mining). Ao utilizar o programa **MIndUtil**, está-se utilizando alguns algoritmos da biblioteca MLC^{++}.

Para executar **MIndUtil** são necessários dois arquivos: o arquivo de dados e o arquivo schema, que indica o conteúdo do arquivo de dados e o descritor dos campos do arquivo de dados, respectivamente.

Para exemplificar a utilização do **MIndUtil**, será utilizado um exemplo referente às características de carros. O formato dos arquivos car.schema e car.data são mostrados no Exemplo 16:

Exemplo 16:

Arquivo car.schema:

```
input {
        file "carros.data";

         double `mpg`;
        string `cylinders`;
        double `cubicinches`;
        double `horsepower`;
        double `time_to_sixty`;
        }
```

Arquivo car.data:

14	8	455	225	10
24	4	113	95	15
22	6	198	95	16
18	6	199	97	16
21	6	200	85	16
27	4	97	88	15
25	4	104	95	18
13	8	400	175	12
18	6	258	110	14
22	4	140	72	19
18	6	250	88	15
23	4	122	86	14
28	4	116	90	14
14	8	440	215	9

Para executar **MIndUtil** é necessário especificar alguns parâmetros de configuração.

PARÂMETROS DE CONFIGURAÇÃO
MIndUtil é executado utilizando os seguintes parâmetros:

MIndUtil -s -o <arquivo de opções> -O <opção>=<valor>

Com a opção -s, **MIndUtil** não utilizará o valor das variáveis de ambiente. A opção -o <arquivo de opções> define o nome do arquivo texto que possui opções necessárias para a execução de **MIndUtil**. Esse arquivo deve possuir sufixo *.classify-opt*. A opção -O (maiúscula) permite fixar o valor de uma opção específica.

Existe uma hierarquia para se passar as opções ao **MIndUtil**. Elas podem ser passadas das seguintes formas:

- **Valores default:** quando uma opção que possui valor *default* não tem seu valor fixado, **MIndUtil** utilizará o valor *default*.
- **Variáveis de ambiente:** uma variável de ambiente pode ter seu valor utilizado por **MIndUtil**.
- **Opções por linha de comando:** são as opções que possuem valor fixado da forma -O <Opção>=<Valor> ou -o <arquivo de opções>.
- **Entrada do Usuário:** quando uma opção que não tem valor *default* não teve seu valor fixado, o usuário deve fornecer seu valor. A opção que indica ao MInUtil quando perguntar ao usuário o valor de uma opção não fixada é PROMPTLEVEL. PROMPTLEVEL pode ser fixada pelo arquivo *.classify-opt* ou na forma -O <Opção>=<valor>. Ela pode possuir os seguintes valores:
  - **required-only:** o usuário é solicitado somente quando uma variável que não tem valor *default* não foi fixada.
  - **basic:** somente os valores das variáveis mais comuns são pedidos.
  - **all:** os valores de todas as variáveis são pedidos.

OPÇÕES GERAIS
As seguintes opções são comuns e necessárias à execução do **MIndUtil** em todos os modos:

- **MODE:** opção que indica em qual modo **MIndUtil** será executado.
- **SCHEMA:** string que indica o nome do arquivo .schema.
- **LABEL:** string que indica a coluna do arquivo de dados que será usada como *label*.
- **DISC_TYPE:** essa opção determina a forma de discretização dos atributos. Pode possuir os seguintes valores:
  - *binning*: discretização uniforme.
  - *entropy*: discretização automática.
- **MIN_SPLIT:** inteiro que determina o número mínimo de registros que cada "pacote binário" deve conter.

Além dessas opções, cada modo de execução exige a especificação de várias outras. A seguir será dado uma explicação sobre cada modo e serão especificadas as opções necessárias à execução do mesmo.

MODOS DE EXECUÇÃO
MIndUtil pode ser executado nos seguintes modos:

- **Modo induce: MIndUtil** criará um arquivo classificador e um arquivo visualizador correspondente.
- **Modo train-and-test: MIndUtil** dividirá o arquivo de dados em duas partes. De uma será criado um classificador e a outra será utilizado para testar esse classificador.
- **Modo estimate-accuracy: MIndUtil** avaliará a precisão de um arquivo classificador.
- **Modo discretize:** modo utilizado para discretizar atributos contínuos.
- **Modo auto-select:** modo utilizado para encontrar um conjunto de atributos importantes dado um conjunto de atributos.
- **Modo compute-importance:** modo em que **MIndUtil** calcula a importância de atributos especificados.

- **Modo visualize**: **MIndUtil** criará um arquivo visualizador dado um arquivo classificador.
- **Modos mineset-to-mlc e mlc-to-mineset:** modos em que **MIndUtil** converte arquivos do formato *Mineset®* para o formato MLC^{++} ou vice-versa.

Para a execução de **MIndUtil** em cada modo, é necessário que algumas opções sejam fixadas. A seguir será dado uma descrição das opções necessárias para a execução em cada modo.

MODO Induction
O modo *Induction* corresponde aos valores *induce* e *train-and-test* da opção *mode*. No modo *induce* **MIndUtil** cria um arquivo classificador e um arquivo visualizador utilizando o arquivo de dados e os parâmetros passados. No modo *train-and-test*, **MindUtil** divide o arquivo de dados em duas partes. De uma, criará um arquivo classificador e da outra, testará esse classificador.

Para ser executado nesses modos, é necessário fixar os valores das seguintes opções:

- **INDUCER:** essa opção determina se será criado classificador para Decision Tree ou para Evidence Visualizer.
- **VIZ_NAME:** string que corresponde ao nome do arquivo visualizador que será criado.
- **CLASSIFIER_NAME:** string que corresponde ao nome do arquivo classificador que será criado.
- **DISP_CONFUSION_MAT:** booleano que define se será mostrado uma matriz de confusão.
- **TRAIN_FRACTION:** essa opção deve ser um número real entre 0 e 1 que indicará ao **MIndUtil** a fração do arquivo de dados que será usado como teste. Essa opção deve ser utilizada somente no modo train-and-test.
- **ACC_EST_SEED:** inteiro usado pelo **MIndUtil** como ponto de partida para a geração de números aleatórios.

OPÇÃO INDUCE=DECISION-TREE
A opção *INDUCE=decision-tree* indica que o arquivo classificador a ser criado deve ser um classificador para árvore de decisão. Para a execução de **MIndUtil** nesse modo são necessárias as seguintes opções:

- **DT_MAX_LEVEL:** número inteiro que determina o número máximo de níveis da árvore. Zero indica arvore sem limite.
- **DT_PRUNING_FACTOR:** número real que determina o “pruning factor”.
- **DT_LBOUND_MIN_SPLIT:** número inteiro que indica o menor número de registros necessários para se criar pelo menos dois ramos de um nó.
- **DT_MIN_SPLIT_WEIGHT:** número real maior que zero que indica a menor taxa de registros de treino dividido pelo número de classes requerido para se criar dois ramos de um nó.
- **DT_SPLIT_BY:** opção que indica ao **MIndUtil** o critério de avaliação para dividir cada nó. Essa opção pode possuir os seguintes valores: mutual-info, normalized-mutual-info, e gain-ratio.

O Exemplo 17 mostra o conteúdo do arquivo carros_induce-dt.classify-opt.

Exemplo 17:

```
MODE=induce
SCHEMA=carros.schema
LABEL=cylinders
INDUCER=decision-tree
```

DISC_TYPE=bining
MIN_SPLIT=1
DT_MAX_LEVEL=0
DT_SPLIT_BY=normalized-mutual-info
DT_PRUNING_FACTOR=0.85
DT_LBOUND_MIN_SPLIT=5
CLASSIFIER_NAME=carros-dt.class
VIZ_NAME=carros-dt.treeviz
ACC_EST_SEED=7258789
PROMPT_LEVEL=all
LOGLEVEL=1

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

MIndUtil -o carros_induce-dt.classify-opt

Dessa forma, o sistema criará o arquivo classificador carros-dt.class e o arquivo visualizador carros-dt.treeviz.

OPÇÃO INDUCE=evidence
A opção *INDUCE=evidence* indica ao **MIndUtil** que o arquivo classificador a ser criado deve ser um classificador para visualizador de evidência.

Opções:

- **EVI_LAPLACE_CORRECTION:** booleano que indica se será aplicado correção de Laplace.
- **EVI_FSS:** booleano que determina se "feature subset selection" deve ser aplicado.

O Exemplo 18 mostra o conteúdo do arquivo carros_induce-evi.classify-opt.

Exemplo 18:

MODE=induce
SCHEMA=carros.schema
LABEL=cylinders
INDUCER=evidence
EVI_LAPLACE_CORRECTION=true
EVI_FSS=false
DISC_TYPE=entropy
MIN_SPLIT=1
CLASSIFIER_NAME=carros-evi.class
VIZ_NAME=carros-evi.eviviz
ACC_EST_SEED=7258789
PROMPT_LEVEL=all
LOGLEVEL=1

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

MIndUtil -o carros_induce-evi.classify-opt

Dessa forma, o sistema criará o arquivo classificador carros-evi.class e o arquivo visualizador carros-evi.eviviz.

MODO estimate-accuracy
Modo em que **MIndUtil** utiliza o arquivo de dados para avaliar a precisão de um arquivo classificador. As opções necessárias para a execução desse modo são as mesmas do modo *induce* e as seguintes opções:

- **CV_FOLDS:** número inteiro que determina o número de "*cross-validation* folds".
- **CV_TIMES:** número inteiro que determina o número de vezes a se repetir *cross-validation*.

O Exemplo 19 mostra o conteúdo do arquivo carros_estimate-accuracy.classify-opt.

Exemplo 19:

MODE=estimate-accuracy
SCHEMA=carros.schema
LABEL=cylinders
INDUCER=decision-tree
CV_FOLDS=5
CV_TIMES=1
DISC_TYPE=entropy
MIN_SPLIT=1
DISP_CONFUSION_MAT=true
DT_MAX_LEVEL=0
DT_SPLIT_BY=normalized-mutual-info
DT_PRUNING_FACTOR=0.85
DT_LBOUND_MIN_SPLIT=5
CLASSIFIER_NAME=carros-dt.class
VIZ_NAME=carros-dt.treeviz
ACC_EST_SEED=7258789
PROMPT_LEVEL=all
LOGLEVEL=1

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

MIndUtil -o carros_estimate-accuracy.classify-opt

MODO discretize
Modo em que **MIndUtil** discretiza os atributos determinados pelo usuário. A opções são :

- **OUTPUT_NAME:** nome do arquivo que conterá o resultado fornecido pelo sistema. Será um arquivo texto.
- **ATTR_X:** string que representa o nome dos atributos que se quer discretizar. X deve ser um inteiro que inicia em 0.
- **BINS_X:** número inteiro que determina o número de bins a se discretizar o atributo determinado por ATTR_X. Quando BINS_X = 0, será usado discretização heurística.

O Exemplo 20 mostra o conteúdo do arquivo carros_discretize.classify-opt.

Exemplo 20:

MODE=discretize
SCHEMA=carros.schema
LABEL=cylinders
OUTPUT_NAME=carros_dicretize.out

ATTR_0=cubicinches
BINS_0=0
ATTR_1=mpg
BINS_1=4
DISC_TYPE=entropy
MIN_SPLIT=1
ACC_EST_SEED=7258789
PROMPT_LEVEL=all
LOGLEVEL=1

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

MIndUtil -o carros_estimate-discretize.classify-opt

MODO auto-select e compute-importance
Modos utilizados para se encontrar colunas importantes do arquivo de dados dado um conjunto de colunas determinadas pelo usuário (modo *auto-select*) e para calcular a importância de uma ou mais colunas determinadas pelo usuário (modo *compute-importance*). As opções são:

- **OUTPUT_NAME:** nome do arquivo que conterá a resultado fornecido pelo sistema. Será um arquivo texto contendo a importância de cada atributo.
- **ATTR_X**: string que corresponde aos nomes dos atributos pré selecionados. X deve ser um número inteiro que inicia em 0.
- **SELECT_N:** número inteiro que determina o número de atributos a serem selecionados pelo sistema no modo auto-select.

O Exemplo 21 mostra o conteúdo do arquivo carros_auto-select.classify-opt para a execução de **MIndUtil** no modo auto-select.

Exemplo 21:

MODE=auto-select
SCHEMA=carros.schema
LABEL=cylinders
OUTPUT_NAME=carros.out
ATTR_0=cubicinches
SELECT_N=3
DISC_TYPE=entropy
MIN_SPLIT=1
ACC_EST_SEED=7258789
PROMPT_LEVEL=all
LOGLEVEL=1

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

MIndUtil -o carros_auto-select.classify-opt

O Exemplo 22 mostra o conteúdo do arquivo carros_compute-impotance.classify-opt para modo compute-importance.

Exemplo 22:

MODE=compute-importance
SCHEMA=carros.schema
LABEL=cylinders

OUTPUT_NAME=carros.out
ATTR_0=horsepower
ATTR_1=mpg
DISC_TYPE=entropy
MIN_SPLIT=1
ACC_EST_SEED=7258789
PROMPT_LEVEL=all
LOGLEVEL=1

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

MIndUtil -o carros_ compute-importance.classify-opt

MODO visualize
Modo em que **MIndUtil** cria um arquivo visualizador correspondente a um arquivo classificador fornecido pelo usuário. As opções são:

- **CLASSIFIER_NAME:** nome do arquivo classificador que será utilizado pelo sistema para gerar o arquivo visualizador. O arquivo classificador deve incluir a sufixo *.class*.
- **VIZ_NAME:** nome do arquivo visualizador que será criado pelo sistema. Deve incluir o sufixo *.treeviz* caso o arquivo classificador seja arquivo de árvore de decisão, ou *.eviviz* caso o arquivo classificador seja de evidence visualizer.

O Exemplo 23 mostra o conteúdo do arquivo carros_viz.treeviz.

Exemplo 23:

MODE=visualize
CLASSIFIER_NAME=carros-dt.class
VIZ_NAME=visualize-dt.treeviz
PROMPT_LEVEL=all

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

***MIndUtil** -o carros_viz.treeviz*

MODOS mineset-to-mlc e mlc-to-mineset
No modo *mineset-to-mlc* **MIndUtil** converterá os arquivos .data e .schema, especificados pelo usuário, para um formato em que possam ser utilizados pela biblioteca MLC^{++}. Será criado um arquivo .names, correspondente ao arquivo .schema, e um arquivo .all, correspondente ao arquivo .data.

No modo *mlc-to-mineset* **MIndUtil** converterá os arquivos especificados pelo usuário do formato MLC^{++} para o formato utilizado no *Mineset*. As opções para o modo m*ineset-to-mlc*:

- **SPLIT_TRAIN_TEST:** booleano que indica ao **MindUtil** se o arquivo de dados deve ser dividido nas partes treino e teste.
- **MLC_FILE:** string que corresponde ao nome dos arquivos MLC^{++} criados. Será criado um arquivo com sufixo .names, que corresponde ao .schema e um arquivo .all, que corresponde ou arquivo .data.

O Exemplo 24 mostra o conteúdo do arquivo carros_mineset-to-mlc.

Exemplo 24:

MODE=mineset-to-mlc
SCHEMA=carros.schema
MLCFILE=carros_mlc
SPLIT_TRAIN_TEST=no
LABEL=cylinders
PROMPT_LEVEL=all
LOGLEVEL=1

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

MIndUtil -o carros_ mineset-to-mlc

As opções para o modo *mlc-to-mineset* são:

- **DATAFILE:** nome do arquivo de dados MLC^{++} a importar.
- **NAMESFILES:** nome do arquivo MLC^{++} que descreve as colunas do arquivo de dados.
- **OUTPUT_DATA:** nome do arquivo de dados no formato Mineset a ser criado pelo sistema.
- **OUTPUT_SCHEMA:** nome do arquivo .schema a ser criado pelo sistema.
- **OUTPUT_LABEL:** string que indica ao **MIndUtil** o nome do atributo Label.
- **REMOVE_UNKNONOWN_INST:** booleano que indica se o sistema deve remover dos arquivos criado registros com valores não conhecidos.

O Exemplo 25 mostra o conteúdo do arquivo carros_mlc-to-mineset.

Exemplo 25:

MODE=mlc-to-mineset
DATAFILE=carros_mlc.all
NAMESFILE=carros_mlc.names
OUTPUT_DATA=carros_mlc-mineset.data
OUTPUT_SCHEMA=carros_mlc-mineset.schema
OUTPUT_LABEL=cylinders
REMOVE_UNKNOWN_INST=yes
PROMPT_LEVEL=all
LOGLEVEL=1

Para executar **MIndUtil** com essas opções basta utilizar a seguinte linha de comando:

MIndUtil -o carros_mlc-to-mineset

10. BIBLIOGRAFIA

[Aha 92] D. W. Aha, *Tolerating Noisy, Irrelevant and Novel Attributes in Instance-Based Learning Algorithms*, *International Journal of Man-Machine Studies*, vol. 36, no. 1, pp. 267-287, 1992.

[Ali 96] K. M. Ali, *Learning Probabilistic Relational Concept Descriptions*, PhD thesis, University of California, Irvine, 1996. Available at http://www.ics.uci.edu/~ali

[Auer 95] P. Auer, R. Holte, & W. Maass, *Theory and Applications of Agnostic Pac-Learning with Small Decision Trees*, in A. Prieditis & S. Russell eds., *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann, 1995.

[Breiman 84] L. Breiman, J. H. Friedman, R. A. Olshen, & C. J. Stone, *Classification and Regression Trees*, Wadsworth International Group, 1984.

[Breiman 94] L. Breiman, *Bagging predictors*, *Technical Report Statistics Department*, University of California at Berkeley, 1994.

[Breiman 96] L. Breiman, *Bagging Predictors*, *Machine Learning*, vol. 24, pp. 123-140, 1996.

[Brodley 92] C. E. Brodley, & P. Utgoff, *Multivariate Decision Trees*, *Technical Report MASSCS 92-93*, University of Massachusetts, Amherst, 1992.

[Clark 89] P. Clark, & T. Niblett, *The CN2 Induction Algorithm*, *Machine Learning*, vol. 3, no. 4, pp. 261-283, 1989.

[Clark 91] P. Clark, & R. Boswell, *Rule Induction with CN2: Some Recent Improvements*, in Y. Kodratoff, ed., *Proceedings of the Fifth European Conference (EWSL-91)*, Springer Verlag, pp. 151-163, 1991. Available at http://www.cs.utexas.edu/users/pclark/papers/newcn.ps

[Cohen 95] W. W. Cohen, *Fast Effective Rule Induction*, in A. Prieditis & S. Russell, eds, *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann, 1995.

[Cost 93] S. Cost, & S. Salzberg, *A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features*, *Machine Learning*, vol.10, no. 1, pp. 57-78, 1993.

[Dasarathy 90] B. V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, California, 1990.

[Devijver 82] P. A. Devijver, & J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice-Hall International, 1982.

[Domingos 96] P. Domingos, & M. Pazzani, *Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier*, in L. Saitta, ed., *Machine Learning: Proceedings of the Thirteenth International Conference*, Morgan Kaufmann, pp. 105-112, 1996.

[Dougherty 95] J. Dougherty, R. Kohavi, & M. Sahami, *Supervised and Unsupervised Discretization of Continuous Features*, in A. Prieditis & S. Russell, eds., *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann, pp. 194-202, 1995.

[Duda 73] R. Duda, & P. Hart, *Pattern Classification and Scene Analysis*, Wiley, 1973.

[Efron 93] B. Efron, & R. Tibshirani, *An Introduction to the Bootstrap*, Chapman & Hall eds., 1993.

[Fayyad 93] U. M. Fayyad, & K. B. Irani, *Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning*, in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers Inc., pp. 1022-1027, 1993.

[Friedman 96] J. Friedman, R. Kohavi, & Y. Yun, *Lazy Decision Trees*, in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, AAAI Press

and the MIT Press, pp. 717-724, 1996.

[Gansner 93] E. R. Gansner, E. Koutsofios, S. C. North, & K. P. Vo, *A Technique for Drawing Directed Graphs*, in *IEEE Transactions on Software Engineering*, pp. 214-230, 1993.

[Geman 92] S. Geman, E. Bienenstock, & R. Doursat, *Neural Networks and the bias/variance Dilemma*, *Neural Computation*, vol. 4, pp. 1-48, 1992.

[Good 65] I. J. Good, *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*, M.I.T. Press, 1965.

[Henery 94] R. J. Henery, *Classification*, *Machine Learning*, Neural and Statistical Classification, D. Michie, D. J. Spiegelhalter, & C. C. Taylor (eds.), Ellis Horwood, pp. 6-16, 1994.

[Hertz 91] J. Hertz, A. Krogh, & R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, 1991.

[Holland 86] J. H. Holland, *Escaping Brittleness: The Possibilities os General-purpose Learning Algorithms Applied to Parallel Rule-based Systems*, in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, (Eds.), Morgan Kaufmann, vol. 2, 1986.

[Holte 93] R. C. Holte, *Very Simple Classification Rules Perform Well on Most Commonly Used Datasets*, *Machine Learning*, vol. 11, pp. 63-90, 1993.

[Ileath 93] D. Ileath, S. Kasif, & S. Salzberg, *Induction of Oblique Decision Trees*, *in 13th International Joint Conference on Artificial Intelligenge*, Morgan Kaufmann, pp. 1002-1007, 1993.

[Kearns 94] M. J. Kearns, & U. V. Vazirani, *An Introduction to Computational Learning Theory*, MIT Press, 1994.

[Kohavi 94a] R. Kohavi, G. John, R. Long, D. Manley, & K. Pfleger, *MLC++: A Machine Learning Library in C++*, in *Tools with Artificial Intelligence*, IEEE Computer Society Press, pp. 740-743, 1994. Available at http://www.sgi.com/Technology/mlc

[Kohavi 94b] R. Kohavi, *Bottom-Up Induction of Oblivious, Read-Once Decision Graphs: Strengths and Limitations*, in *Twelfth National Conference on Artificial Intelligence*, pp. 613-618, 1994.

[Kohavi 94c] R. Kohavi, *Bottom-Up Induction of Oblivious, Read-Once Decision Graphs*, in F. Bergadano & L. D. Raedt, eds, *Proceedings of the European Conference on Machine Learning*, pp. 154-169, 1994.

[Kohavi 95a] R. Kohavi, *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*, in C. S. Mellish, ed., *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 1137-1143, 1995.

[Kohavi 95b] R. Kohavi, *The Power of Decision Tables*, in N. Lavrac & S. Wrobel, eds, *Proceedings of the European Conference on Machine Learning*, *Lecture Notes in Artificial Intelligence*, Springer Verlag, Berlin, Heidelberg, New York, vol. 914, pp. 174-189, 1995.

[Kohavi 95c] R. Kohavi, & D. Sommerfield, *Feature Subset Selection using the Wrapper Model: Overfitting and Dynamic Search Space Topology*, in *The First International Conference on Knowledge Discovery and Data Mining*, pp. 192-197, 1995.

[Kohavi 95d] R. Kohavi, & G. John, *Automatic Parameter Selection by Minimizing Estimated Error*, in A. Prieditis & S. Russell, eds, *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann, pp. 304-312, 1995.

[Kohavi 95e] R. Kohavi, *Wrappers for Performance Enhancement and Oblivious Decision Graphs*, PhD Thesis, Stanford University, Computer Science Department. STAN-CS-TR-95-1560, 1995. Available at ftp://starry.stanford.edu/pub/ronnyk/teza.ps

[Kohavi 95f] R. Kohavi, & G. H. John, *Wrappers for feature subset selection*, in

Proceedings of the First International Conference on Knowledge Discovery and Data Mining, 1995.

[Kohavi 96a] R. Kohavi, & D. H. Wolpert, *Bias Plus Variance Decomposition for Zero-One Loss Functions*, in L. Saitta, ed., *Machine Learning: Proceedings of the Thirteenth International Conference*, Morgan Kaufmann Publishers, Inc, 1996. Available at http://robotics.stanford.edu/users/ronnyk

[Kohavi 96b] R. Kohavi, *Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision Tree Hybrid*, in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 202-207, 1996. Available at http://robotics.stanford.edu/users/ronnyk

[Kohavi 97a] R. Kohavi, & C. Kunz, *Option Decision Trees with Majority Votes*, in D. Fisher, ed., *Machine Learning: Proceedings of the Fourteenth International Conference*, Morgan Kaufmann, 1997. Available at http://robotics.stanford.edu/users/ronnyk

[Kohavi 97b] R. Kohavi, D. Sommerfield, J. Dougberty, *Data Mining using MLC++, A Machine Learning Library in C++*, in *International Journal on Artificial Intelligence Tools*, vol. 0, no. 0, 1997.

[Koutsofios 94] E. Koutsofios, & S. C. North, *Drawing Graphs with Dot*, 1994. Available by anonymous ftp from ftp://research.att.com/dist/drawdag/dotdoc.ps.Z and available at http://www.research.att.com/sw/tools/graphviz

[Krogh 95] A. Krogh, & J. Vedelsby, *Neural Network Ensembles, Cross Validation, and Active Learning*, in *Advances in Neural Information Processing Systems*, MIT Press, vol. 7, 1995.

[Langley 92] P. Langley, W. Iba, & K. Thompson, *An Analysis of Bayesian Classifiers*, in *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI Press and MIT Press, pp. 223-228, 1992.

[Langley 94] P. Langley, & S. Sage, *Induction of Selective Bayesian Classifiers*, In *Proceedings of The Tenth Conference On Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., Seattle, WA, pp. 399-406, 1994.

[Lavrac 94] N. Lavrac, & S. Dzeroski, *Inductive Logic Programming : Techniques and Applications*, E. Horwood, New York, 1994.

[LeBlank 90] J. LeBlank, M. Ward, & N. Wittels, *Exploring n-dimensional Databases*, in *Proceedings of Visualization*, pp. 230-237, 1990.

[Littlestone 88] N. Littlestone, *Learning Quickly when Irrelevant Attributes Abound: A New Linear-Threshold Algorithm*, *Machine Learning*, vol. 2, pp. 285-318, 1988.

[Maass 94] W. Maass, *Efficient Diagnostic PAC-Learning with Simple Hypotheses*, in *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pp. 67-75, 1994.

[Michalski 78] R. S. Michalski, *A Planar Geometric Model for Representing Multidimensional Discrete Spaces and Multiple-Valued Logic Functions*, *Technical Report UIUCDCS-R-78-897*, University of Illinois at Urbaba-Champaign, 1978.

[Muggleton 92] S. Muggleton, *Inductive Logic Programming*, Academic Press, 1992.

[Murphy 95] P. M. Murphy, & C. J. Merz, *UCI Repository of Machine Learning Databases*, 1995. Available at http://www.ics.uci.edu/~mlearn/MLRepository.html

[Murthy 94] S. K. Murthy, S. Kasif, & S. Salzberg, *A System for the Induction of Oblique Decision Trees*, *Journal of Artificial Intelligence Research*, vol. 2, pp. 1-33, 1994.

[Näeher 92] S. Naeher, *LEDA: A Library of Efficient Data Types and Algorithms*, Max-Planck-Institut fuer Informatik, IM Stadtwald, D-66123 Saarbruecken, FRG, 3.0 edition, 1992. Available by anonymous ftp in ftp://ftp.cs.uni-sb.de/LEDA

[Parsaye 96] K. Parsaye, *Rules are Much More Than Decision Trees*, 1996. Available at

http://www.datamining.com/datamine/trees.htm

[Perrone 93] M. Perrone, *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization*, PhD Thesis, Brown University, Physics Dept, 1993.

[Quinlan 86] J. R. Quinlan, *Induction of decision trees*, *Machine Learning*, Reprinted in Shavlik and Dietterich (eds.) Readings in Machine Learning, vol. 1, pp. 81-106, 1986.

[Quinlan 93] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, California, 1993.

[Rice 88] J. A. Rice, *Mathematical Statistics and Data Analysis*, Wadsworth & Brooks/Cole, 1988.

[Rumelhart 86] D. E. Rumelhart, G. E. Hinton, & R. J. Williams, *Learning Internal Representations by Error Propagation*, MIT Press, chapter 8, 1986.

[Taylor 94] C. Taylor, D. Michie, & D. Spiegalhalter, *Machine Learning, Neural and Statistical Classification*, Paramount Publishing International, 1994.

[Thrun 91] Thrun et al, *The Monk's Problems: A Performance Comparison of Different Learning Algorithms*, *Technical Report CMU-CS-91-197*, Carnegie Mellon University, 1991.

[Valiant 84] L. G. Valiant, *A Theory of the Learnable*, *Communications of the ACM 27*, pp. 1134-1142, 1984.

[Weiss 91] S. M. Weiss, & C. A. Kulikowski, *Computer Systems that Learn*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[Wettschereck 94] D. Wettschereck, *A Study of Distance-Based Machine Learning Algorithms*, PhD Thesis, Oregon State University, 1994.

[Wnek 90] J. Wnek, J. Sarma, A. A. Wahab, & R. S. Michalski, *Comparing Learning Paradigms Via Diagrammatic Visualization*, in *Methodologies for Intelligent Systems, In Proceedings of the Fifth International Symposium*, Also technical report MLI90-2, University of Illinois at Urbaba-Champaign, pp. 428-437, 1990.

[Wnek 94] J. Wnek, & R. S. Michalski, *Hypothesis-driven Constructive Induction in AQ17-HCI : A Method and Experiments*, *Machine Learning*, vol.14, no. 2, pp. 139-168, 1994.

[Wolpert 92] D. H. Wolpert, *Stacked generalization*, *Neural Networks*, vol. 5, pp. 241-259, 1992.