Extração Automática de Conhecimento por Múltiplos Indutores

José Augusto Baranauskas

Orientação:
Maria Carolina Monard

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional.

USP – São Carlos

Junho de 2001

Este documento foi preparado com o formatador de textos LaTeX. O sistema de citações de referências bibliográficas utiliza o padrão *Chicago* do sistema BibTeX (Prati, Baranauskas & Monard 1999).

Além disso, no decorrer desta tese são utilizados vários termos estrangeiros que não foram traduzidos para a nossa língua. Não se trata, em momento algum, de desrespeito à língua portuguesa mas reside no simples fato de que tais termos encontram-se amplamente difundidos e utilizados pela comunidade de Aprendizado de Máquina.

Revisão Setembro/2001

Orientação
Maria Carolina Monard

Autor
José Augusto Baranauskas

Extração Automática de Conhecimento por Múltiplos Indutores

Resumo

A luta contra o erro tipográfico tem algo de homérico. Durante a revisão os erros se escondem, fazem-se positivamente invisíveis. Mas assim que o livro sai, tornam-se visibilíssimos, verdadeiros sacis a nos botar a língua em todas as páginas. Trata-se de um mistério que a ciência ainda não conseguiu decifrar ...

—*Monteiro Lobato*

Nesta tese são investigados três problemas básicos em aprendizado supervisionado: seleção de atributos, composição de atributos e combinação de classificadores simbólicos.

A seleção de atributos é uma atividade de pré-processamento de dados que seleciona um subconjunto de atributos do conjunto original de exemplos. Existem, basicamente, três abordagens que são empregadas para a seleção de atributos: embutida, filtro e *wrapper*; as duas últimas pesquisadas neste trabalho. Os experimentos realizados, utilizando diversos indutores e conjuntos de exemplos, para avaliar as abordagens filtro e *wrapper* nos permitem concluir que o uso de filtros devem ser considerados antes de se cogitar a utilização de *wrappers*, no caso de existirem muitos atributos para descrever os exemplos. Sob a perspectiva de compreensibilidade sintática do conhecimento induzido, a análise sobre o impacto da seleção de atributos em um classificador simbólico mostrou um aumento do número de regras e do número de condições por regra.

A composição de atributos, também conhecida como indução construtiva, é outra atividade de pré-processamento de dados. Dentre as várias abordagens de composição de atributos (guiada por dados, por hipótese, por conhecimento e multi-estratégia), nesta tese é proposta uma metodologia para composição de atributos guiada pelo conhecimento. Os resultados dos experimentos realizados utilizando a metodologia proposta em repositórios naturais (pré-processados) mostram que, mesmo com o auxílio do usuário/especialista, não é simples construir atributos derivados que sejam realmente relevantes para aprender o conceito embutido nos conjuntos de exemplos analisados. Entretanto, com dados do mundo real, a metodologia proposta tem mostrado seu verdadeiro potencial.

A combinação de classificadores, simbólicos ou não, é uma atividade de mineração de dados. Na realidade, uma das preocupações do Aprendizado de Máquina simbólico é que os classificadores induzidos devem ser fáceis de serem compreendidos pelos seres humanos. Para isso, deve-se escolher o indutor com *bias* mais adequado para cada tipo de situação, já que pesquisas mostraram que não existe o *melhor* indutor para todos os domínios. Aliada a essa escolha, é possível fazer uso de vários classificadores, combinando-os num único classificador final, formando um *ensemble*. Os *ensembles* possuem a tendência de melhorar o desempenho na classificação de exemplos não vistos durante o processo de aprendizado. Entretanto, o emprego de *ensembles* dificulta a compreensão humana sobre o comportamento do classificador final, já que ele deixa de ser simbólico, mesmo assumindo que cada classificador individual que o compõe seja simbólico. Na realidade, a combinação de classificadores simbólicos — provenientes de diferentes indutores — em um classificador final também simbólico é um tópico novo de pesquisa, ainda com poucos resultados divulgados. Com o objetivo de preencher essa lacuna, é proposto e desenvolvido neste trabalho o sistema XRULER. Para isso, inicialmente foi definido o formato padrão de regras PBM, o qual fornece uma perspectiva unificada sob a qual todo classificador simbólico pode ser convertido e analisado.

Dentre outros componentes, o sistema XRULER possui um algoritmo de cobertura que pode ser aplicado ao conjunto de regras induzidas por diversos indutores para se obter um classificador simbólico final. Nos experimentos realizados com o sistema XRULER os resultados obtidos mostraram aumento da

precisão e redução do número de regras. Isso pode ser considerado um avanço no sentido de uma maior compreensibilidade por seres humanos do conjunto final de regras.

Advisor
Maria Carolina Monard

Author
José Augusto Baranauskas

Automatic Knowledge Extraction using Multiple Inducers

Abstract

An ounce of knowledge is worth a ton of data.

—Brian R. Gaines

In this work we investigate three supervised Machine Learning problems: feature selection, feature composition and combination of symbolic classifiers.

Feature selection is a data pre-processing task which selects a subset of relevant features in a dataset. There are three approaches for feature selection: embeded, filter and wrapper; the last two approaches are investigated in this work. The experiments performed using several inducers as well as datasets to evaluate filters and wrappers allow us to conclude that the use of filters should be considered before wrappers, especially if the dataset contains many features. Considering the degree of comprehensibility of the induced knowledge, the impact of feature selection in symbolic classifiers showed an increase in the number of rules as well as in the number of conditions per rule.

Feature composition, also known as constructive induction, is another data pre-processing task. Among several approaches for feature composition (guided by data, by hypothesis, by knowledge and multi-strategy), we propose a methodology for feature composition guided by knowledge. The results of the experiments performed using this methodology in pre-processed dataset showed that, even with the help of an user/expert, it is not simple to construct derived features that are relevant to learn the concept embeded in the analyzed datasets. However, further experiments using real world dataset showed the proposed methodology has a potential.

Combination of classifiers, symbolic or not, is another important issue for Machine Learning and Data Mining tasks. Furthermore, one of the central issues in symbolic Machine Learning is related to the fact that the induced classifiers should be accurated and easily understandable by humans. Thus, it is important to choose inducers with appropriate bias, since it has been shown that there is no *best* inducer for all domains. Besides that, it is also possible to create an ensemble of classifiers by inducing a set of classifiers and combining its individual decisions to classify new examples. An ensemble is often more accurate than the individual classifiers that make it up. However, an ensemble of classifiers, even assuming that each individual classifier in the ensemble is symbolic, is no longer symbolic. In fact, the combination of symbolic classifiers into a final one that is still symbolic is a new research area with very few results. To fill this gap, we propose and develop the XRULER system. Initially, we define the standard rule form PBM, an unified view that allows to convert and to analyze symbolic classifiers in a common framework.

The XRULER system contains, among other components, a cover algorithm that can be used to select a set of rules from several rule sets induced by different learning algorithms. This allows to obtain a final classifier that is still symbolic. The experiments performed using the XRULER system show an increase in accuracy and decrease in the number of final rules. This can be considered an advance towards human comprehensibility of the induced knowledge.

Aos meus pais

Agradecimentos

Those that know, do. Those that understand, teach.

—*Aristotle*

Aristósteles, especificamente a respeito de orientadoras, escreveu:

> "Este é o ponto essencial: a partir de sua preparação científica, a orientadora deve instigar não apenas a capacidade, mas o desejo de observar fenômenos naturais. No nosso sistema, ela deve tornar-se uma influência passiva, muito mais do que ativa, e sua passividade deve ser composta de uma ansiosa curiosidade científica, e um respeito absoluto pelo fenômeno que ela deseja observar. A orientadora deve entender e sentir sua posição de observadora; a atividade deve permanecer no fenômeno."

Uma orientadora é tanto uma mentora quanto uma fonte de conselhos científicos; ela encoraja o interesse de seus orientados, ao invés de promover seus próprios interesses; ela encontra-se disponível para dar conselhos em sua tese e na sua carreira científica. Uma boa orientadora, além de tudo, sabe ouvir, científica e pessoalmente, seus orientados, compreendendo-nos, mesmo quando sequer tenhamos verbalizado palavra alguma. Ela possui o misterioso dom de perceber e *sentir* o que se passa em nossas mentes, quando nós mesmos não somos capazes de fazê-lo. Ela é como um farol, no meio das tempestades, que sabemos estar lá para nos orientar; um porto seguro, onde podemos ancorar nossas dúvidas.

Nesse sentido, tive a sorte de conseguir uma orientadora, tanto no mestrado quanto no doutorado, que preencheu não só os requisitos de boa orientadora, como muito mais do que é possível conceber. Tenho duplamente sorte pela orientadora-amiga e amiga-orientadora, reunidas numa única pessoa; poucos orientados têm essa oportunidade.

Eu gostaria de agradecer minha orientadora-amiga, Maria Carolina Monard, por ensinar-me a entender Aprendizado de Máquina e *Data Mining* e a ver muitas coisas sob uma perspectiva diferente. Sua intuição e comentários foram sempre de grande valia, mantendo minha mente fervilhando de pensamentos muitas horas após nossos debates. Nossos debates, sem dúvida alguma, foram um dos pontos mais interessantes e esclarecedores, dos quais terei muita saudade.

Eu gostaria de agradecer minha amiga-orientadora, Maria Carolina Monard, por surpreender-me com sua sagacidade, compreendendo e relevando épocas difíceis pelas quais passei. Em várias situações, ela simplesmente parecia ler meus pensamentos ... Seu senso de justiça e imparcialidade sempre me fascinaram, pois são ponderados com o bom senso; como ela costuma dizer, "cada caso é um caso".

Gostaria, também, de agradecer meus outros professores ligados ao LABIC, André Carvalho e Solange Rezende, ambos muito estimados. André foi quem me introduziu os conceitos sobre Redes Neurais Artificiais e Algoritmos Genéticos. Solange, mesmo não tendo tido a oportunidade de ser seu aluno, aconselhou-me em várias situações e agradeço seu empenho. O esforço de ambos, juntamente com a Carolina, em proporcionar aos seus orientados a oportunidade de participar de congressos e eventos da área deve ser ressaltado. Além de viabilizar financeiramente nossa participação, os três preferiram, em todas ocasiões possíveis, abdicar do seu direito a uma viagem de avião, mais confortável e rápida, para usar tal recurso em nosso favor. Tal abdicação é digna de louvor e não será esquecida. Citando Aristóteles, *"In the arena of human life the honors and rewards fall to those who show their good qualities in action"*.

Eu agradeço aos meus outros professores do ICMC, local onde iniciei minha formação científica. Dentre eles, Rosane Minghim introduziu-me conceitos sobre computação gráfica. Embora não tendo a oportunidade de aplicar esses conhecimentos em minha tese, visualização é

uma questão importante em *Data Mining*, à qual prometo dar mais atenção daqui para frente. José Carlos Maldonado, juntamente com a Solange, promoveu o auxílio necessário para que eu apresentasse nosso trabalho na *Second International Conference on Data Mining*, realizada em Cambridge, UK.

Quero agradecer a todo pessoal de suporte do ICMC, especialmente as *meninas* da pós-graduação Elisabeth Moreti e Silva, Laura Turi e Marília Marino (recentemente transferida para o setor de eventos), pela paciência que têm com todos nós pós-graduandos. Agradeço às bibliotecárias Giselda Solfa, Gislene de Oliveira, Gláucia Cristianini, Juliana Moraes, Maria Lima, Regina Medeiros, Rosemari Casali, Rosemeire Zambon e Sandra Soligon, sempre solícitas às nossas necessidades.

Eu também quero agradecer Nitin Indurkhya pelos comentários e sugestões sobre meu trabalho e a Ross Quinlan, com o qual tive a honra de trocar alguns poucos *e-mails*, entretanto muito esclarecedores. Agradeço também Peter Flach e Nada Lavrač, que me receberam tão bem na rápida visita à Bristol University, UK. Conversar com Peter Flach foi um dos momentos da vida no qual aprendi mais do que em muitos anos de estudo em livros, e nossa conversa motivou-me a utilizar a novidade de uma regra, além da precisão de Laplace, no sistema XRULER. Pelas poucas oportunidades que tive, fica claro que é fundamental para o progresso científico de nosso país trocar experiências com pesquisadores estrangeiros.

Em muitos debates com a Carolina tive a oportunidade de contar também com a presença de Gustavo Batista, o "lindinho", outro doutorando com quem tive a honra de trocar idéias, além de Ronaldo Prati, orientado de iniciação científica da Carolina, que inicia seu programa de mestrado em breve. Pude acompanhar seu notável progresso desde dois anos atrás. Sua capacidade, sem dúvida alguma, será devidamente explorada pela Carolina, de quem você já é o "xodó".

Eu gostaria de agradecer aos demais colegas do LABIC Adriano Pila, Cláudia Martins, Cláudia Milaré, Cristiane Imamura, Marcos Geromini, Walter Nagai, entre outros, pela amizade e troca de experiência durante esses anos de convívio.

Em especial, agradeço à querida Jaqueline Brigladori Pugliesi pela atenção e carinho ao revisar vários artigos que escrevi, bem como esta tese. Sinto-me em profundo débito, pois não pude ajudá-la em seu exame de qualificação de doutorado, bem como em outras situações. Agradeço à Jaqueline pelos momentos sublimes que compartilhamos juntos e peço mil desculpas pelo pouco tempo que pude dispensar-lhe.

Eu gostaria também de agradecer ao então diretor da Faculdade de Medicina de Ribeirão Preto — USP, José Antunes Rodrigues por liberar-me parte do tempo para iniciar o programa de doutorado. Agradeço também a Kátia Suzuki, Gladys Pierri, Filipe Mesquita, Jony dos Santos e Rogério Castania, colegas com os quais convivo diariamente na FMRP, pelo suporte, troca de experiência, pela paciência e por terem me compreendido, especialmente na fase final desta tese. Wilson Silva Jr. é outra pessoa a quem agradeço, não só pela amizade e consideração, como também pela oportunidade e paciência em tentar agregar-me ao seu grupo de bioinformática.

Eu quero agradecer enormemente à minha família, especialmente aos meus pais, João e Wilma, pela minha formação de caráter, pelas oportunidades proporcionadas e pela consideração que têm por mim. A parte boa em mim é fruto deles. Agradeço também minha querida irmã Cláudia Regina, e minhas sobrinhas Maria Augusta, Maria Angélica e Ana Laura, jóias de minha vida, bem como meu cunhado Arthur. Maria Augusta, você tem sido fonte de inspiração constante para mim, devido à sua força e serenidade. Lamento que você, uma de minhas jóias, tenha sido tirada de mim tão prematuramente pelo Criador, de forma muito dolorosa. Confesso

que sua ausência muito me faz falta.

A todos meus familiares, minhas profundas desculpas pelos momentos que deixei de compartilhar com vocês. Peço apenas que relevem isso por eu estar fazendo algo que muito gosto. Por fim, os erros que porventura eu tenha cometido sobre o que escrevi nesta tese são de responsabilidade única e exclusivamente minha; os acertos são méritos de todas essas pessoas.

Em sua tese, Usama Fayyad escreveu que ele não sabia por que as pessoas exageram agradecendo "quase todas as pessoas no planeta". Sem exageros, tenho certeza que agradeci apenas uma pequena fração das pessoas a quem devo muito e peço especial perdão àquelas que eu omiti inconscientemente. Obrigado a todos!

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

> Imagination is more important than knowledge. Knowledge is limited while imagination embraces the entire world.
>
> —*Albert Einstein, On Science*

Neste capítulo é fornecida uma descrição geral desta tese. Inicialmente são apresentados alguns conceitos sobre Aprendizado de Máquina — AM. Esses conceitos introdutórios são apresentados de modo informal, sendo que definições e conceitos adicionais são tratados no capítulo seguinte. É também apresentada a organização dos demais capítulos, finalizando-se com um breve histórico sobre as atividades desenvolvidas no decorrer desta tese.

Nas próximas seções é descrito o problema da classificação em AM supervisionado, tópico sobre o qual este trabalho está baseado, juntamente com alguns fatores que motivam seu estudo.

1.1 A Hierarquia do Aprendizado

> ... scientific knowledge through deduction is impossible unless a man knows the primary immediate premises ... We must get to know the primary premises by induction; for the method by which even sense-perception implants the universal is inductive ...
>
> —*Aristotle, Posterior Analytics, Book II, Chapter 19, 339 B.C.*

Simon (1983) define aprendizado como:

> "Aprendizado denota alterações no sistema que são adaptativas, no sentido que elas capacitam o sistema a realizar a mesma tarefa, ou tarefas provenientes da mesma população, de forma mais eficiente e eficaz na próxima vez."

Essa definição é demasiadamente abrangente para o contexto deste trabalho, sendo necessária uma definição mais específica, denominada *aprendizado indutivo* ou *aprendizado empírico*. O aprendizado indutivo é definido por Weiss & Kulikowski (1991) como:

> "Um sistema de aprendizado [supervisionado] é um programa de computador que toma decisões baseadas na experiência contida em exemplos solucionados com sucesso."

A indução é a forma de inferência lógica que permite obter conclusões genéricas sobre um conjunto particular de exemplos. Ela é caracterizada como o raciocínio que se origina em um conceito específico e o generaliza, ou seja, da parte para o todo. Na indução, um conceito é aprendido efetuando-se inferência indutiva sobre os exemplos apresentados. Portanto, as hipóteses geradas através da inferência indutiva podem ou não preservar a verdade.

Mesmo assim, a inferência indutiva é um dos principais métodos de derivar conhecimento novo e predizer eventos futuros. Foi através da indução que Arquimedes descobriu a primeira lei da hidrostática e o princípio da alavanca, que Kepler descobriu as leis do movimento planetário, que Darwin descobriu as leis da seleção natural das espécies.

Por exemplo, a indução da primeira lei da hidrostática ocorreu quando o rei de Siracusa, uma colônia da antiga Grécia, solicitou que Arquimedes descobrisse se um ourives havia adulterado a coroa real com prata, sendo que deveria conter apenas ouro. Um dia, enquanto Arquimedes estava entrando em uma banheira, ele notou o vazamento de água e percebeu que havia encontrado a solução[1], que também é conhecida como o princípio de Arquimedes: quando um corpo é parcial ou completamente imerso em um fluído, a perda aparente de peso é igual ao peso do fluído deslocado. Dessa forma, Arquimedes induziu corretamente um princípio que é aplicado a outros corpos. Como o ouro é mais denso que a prata, um dado peso em ouro representa um volume menor do que um mesmo peso em prata. Portanto, ele simplesmente precisava pesar a coroa imersa na água e pesar a água deslocada[2].

Apesar da indução ser o recurso mais utilizado pelo cérebro humano, para derivar conhecimento novo, ela deve ser utilizada com cautela, pois se o número de exemplos for insuficiente, ou se os exemplos não forem bem escolhidos, as hipóteses obtidas podem ser de pouco valor.

Por exemplo, em uma situação adaptada de Monard, Batista, Kawamoto & Pugliesi (1997), um cientista estava pesquisando a audição das formigas. A formiga está parada e o cientista

[1] Em sua excitação, Arquimedes saiu da banheira e correu pela ruas de Siracusa gritando 'Eureka, eureka', que significa 'Eu encontrei'. Para os pasmos siracusianos, pareceu que o homem nú havia, na realidade, perdido algo — suas roupas ou seu juízo.

[2] Infelizmente para o ourives, a coroa deslocou mais água do que o seu peso equivalente em ouro, indicando que ela não era feita de ouro puro ...

dá um grito. A formiga sai correndo. Ele então arranca uma das pernas da formiga, e dá outro grito, da mesma intensidade que o primeiro. A formiga corre, mas não tão depressa como anteriormente. O cientista então arranca as outras pernas e dá outro grito. A formiga não corre. Então ele conclui que *as formigas ouvem pelas pernas*. A conclusão, baseada no experimento do cientista, não é válida porque ele escolheu mal as características relevantes na determinação da audição das formigas.

Assim, o aprendizado indutivo é efetuado a partir de raciocínio sobre exemplos fornecidos por um processo externo ao sistema de aprendizado. O aprendizado indutivo pode ser dividido em *supervisionado* e *não-supervisionado*. No aprendizado supervisionado é fornecido ao algoritmo de aprendizado, ou *indutor*, um conjunto de exemplos de treinamento para os quais o rótulo da classe associada é conhecido. O rótulo da classe é fornecido por um agente, normalmente denominado *professor*, ou o próprio processo que originou os exemplos pode fornecer os rótulos associados.

Em geral, cada exemplo é descrito por um vetor de valores de características, ou atributos, e o rótulo da classe associada. O objetivo do algoritmo de indução é construir um classificador que possa determinar corretamente a classe de novos exemplos ainda não rotulados, ou seja, exemplos que não tenham o rótulo da classe. Para rótulos de classe discretos, esse problema é conhecido como *classificação* e para valores contínuos como *regressão*.

Já no aprendizado não-supervisionado, o indutor analisa os exemplos fornecidos e tenta determinar se alguns deles podem ser agrupados de alguma maneira, formando *agrupamentos* ou *clusters* (Cheeseman & Stutz 1990). Após a determinação dos agrupamentos formados, normalmente, é necessária uma análise para determinar o que cada agrupamento significa no contexto do problema que está sendo analisado (Martins & Monard 2000).

Na Figura 1.1 na página seguinte é mostrada a hierarquia de aprendizado descrita, na qual os nós sombreados levam ao aprendizado supervisionado utilizando classificação, objeto de estudo desta tese.

Para exemplificar o processo de classificação em termos práticos, suponha que se queira aprender uma forma para predizer se um paciente tem problemas cardíacos. Para isso, é necessário verificar os históricos dos pacientes nos quais seriam encontrados registros contendo atributos, tais como idade, sexo, dor no peito, nível de colesterol, taxa máxima de batimentos cardíacos, a presença de angina induzida por exercícios, entre outros. Presume-se que cada registro histórico tenha sido rotulado por um especialista médico como um paciente saudável ou doente. O conjunto de exemplos composto por históricos de pacientes é então fornecido como entrada para um algoritmo de indução. A saída resultante, ou seja, a hipótese induzida,

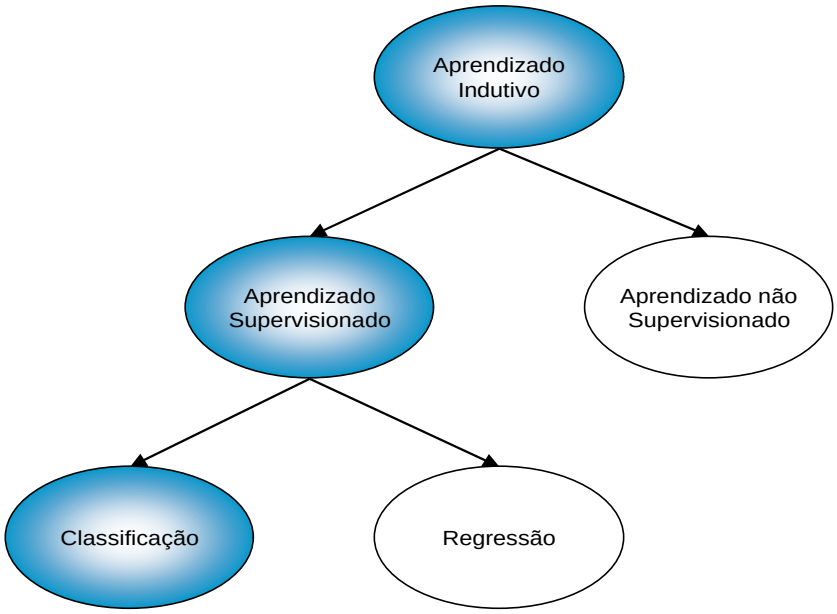


Figura 1.1: A hierarquia do aprendizado

normalmente consiste em algumas regras que permitem classificar novos pacientes, isto é, que permitem determinar se um novo paciente apresenta ou não problema cardíacos. Na Figura 1.2 na próxima página é mostrada uma árvore de decisão induzida a partir de dados reais provenientes do conjunto de exemplos cleve — Cleveland *heart disease* (Blake & Merz 1998). Essa árvore pode ser utilizada para classificar novos pacientes: começando pela raiz da árvore, repetidamente segue-se o ramo de acordo com o atributo testado até que um nó folha seja encontrado, o qual rotula o paciente como saudável (*healthy*) ou doente (*sick*).

De maneira geral, o processo de classificação pode ser ilustrado pela Figura 1.3 na página oposta. O conhecimento sobre o domínio pode ser usado ao escolher os dados, ou fornecer alguma informação previamente conhecida, como entrada ao indutor. Após induzido, o classificador é geralmente avaliado e o processo de classificação pode ser repetido, se necessário, por exemplo, adicionando outros atributos ou exemplos ou mesmo ajustando alguns parâmetros no processo de indução.

Um fator importante que deve ser considerado é que os classificadores devem fornecer uma descrição mais compacta do conceito embutido nos dados caso esses classificadores sejam analisados por seres humanos. Assim sendo, supondo um conjunto de n exemplos, é esperado que o classificador induzido forneça uma descrição menor do que n (caso contrário, os exemplos descreveriam melhor a si próprios — e de forma mais compacta — do que o próprio classificador). Por exemplo, uma árvore de decisão induzida a partir de n exemplos deve ter menos do

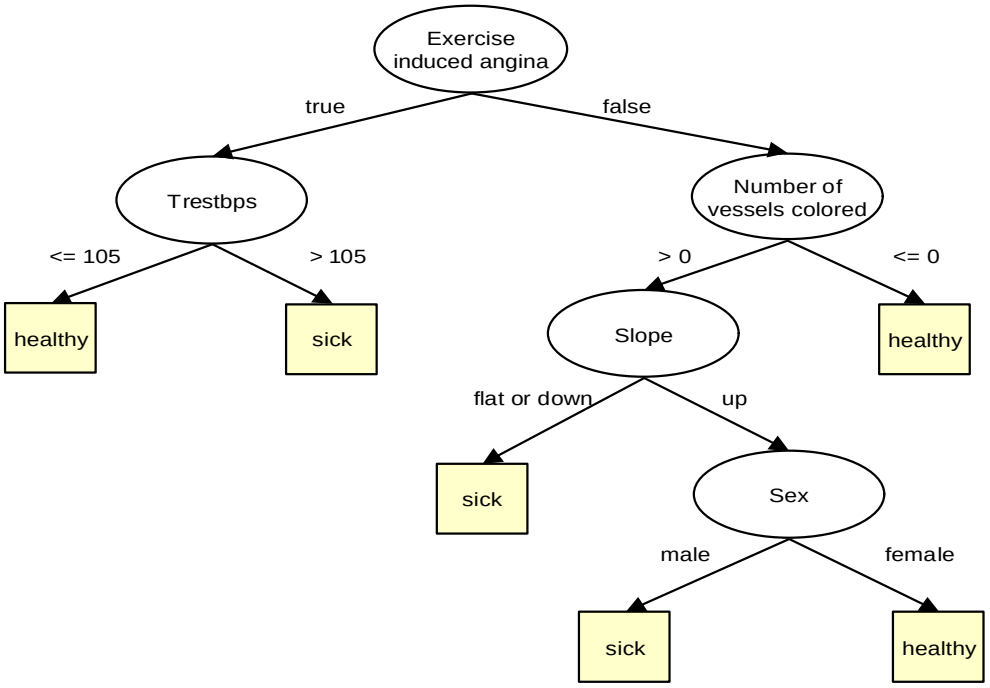
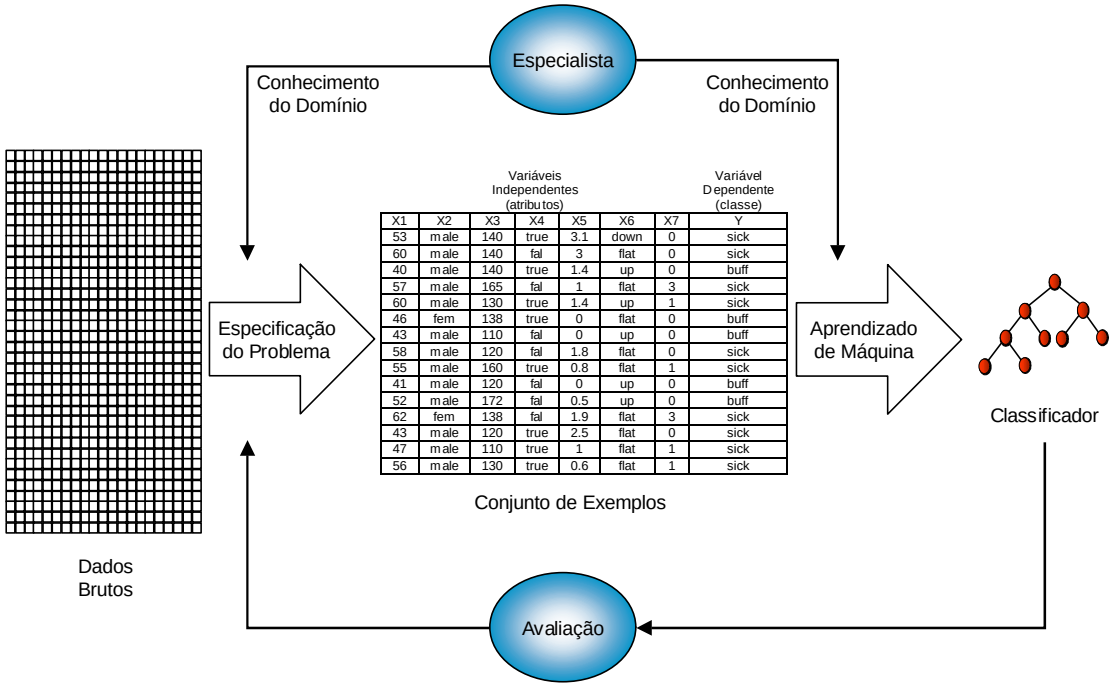


Figura 1.2: Parte da árvore de decisão induzida por C4.5 para o conjunto de exemplos Cleveland *heart disease*



X1	X2	X3	X4	X5	X6	X7	Y
53	male	140	true	3.1	down	0	sick
60	male	140	fal	3	flat	0	sick
40	male	140	true	1.4	up	0	buff
57	male	165	fal	1	flat	3	sick
60	male	130	true	1.4	up	1	sick
46	fem	138	true	0	flat	0	buff
43	male	110	fal	0	up	0	buff
58	male	120	fal	1.8	flat	0	sick
55	male	160	true	0.8	flat	1	sick
41	male	120	fal	0	up	0	buff
52	male	172	fal	0.5	up	0	buff
62	fem	138	fal	1.9	flat	3	sick
43	male	120	true	2.5	flat	0	sick
47	male	110	true	1	flat	1	sick
56	male	130	true	0.6	flat	1	sick

Figura 1.3: O classificador à direita fornece uma interpretação compacta dos dados

que n nós folhas. Outro requisito importante é que os indutores devem ser capazes de utilizar exemplos imperfeitos pois, em geral, os exemplos contêm uma certa quantidade de erros na sua descrição ou mesmo pode ocorrer que uma parcela dos exemplos não apresente valores para alguns atributos.

Um outro fator relevante é o grau de compreensibilidade proporcionado ao ser humano. De acordo com Michalski (1983b) e Kubat, Bratko & Michalski (1998), os sistemas de aprendizado são classificados em duas grandes categorias:

1. sistemas *caixa-preta* que desenvolvem sua própria representação do conceito, isto é, sua representação interna pode não ser facilmente interpretada por humanos e não fornecem nem esclarecimento, nem explicação do processo de reconhecimento;

2. sistemas *orientados a conhecimento* que objetivam a criação de estruturas simbólicas que sejam compreensíveis por humanos.

Na realidade, uma forma alternativa de classificar os sistemas de aprendizado foi formulada por Michie (1988) em termos de três critérios:

1. *critério fraco*: o sistema utiliza exemplos para gerar uma base atualizada para a melhoria do desempenho em exemplos subseqüentes. Redes neurais e métodos estatísticos satisfazem este critério;

2. *critério forte*: o critério *fraco* é satisfeito. Além disso, o sistema é capaz de comunicar sua representação interna de forma simbólica explícita;

3. *critério ultra-forte*: os critérios *fraco* e *forte* são satisfeitos. Além disso, sua representação interna simbólica pode ser utilizada por um humano sem a ajuda de um computador, ou seja, utilizando apenas sua própria mente.

Assim, no aprendizado de conceitos, o interesse principal consiste em obter descrições simbólicas que sejam fáceis de serem compreendidas e utilizadas por meio de modelos mentais. Segundo o *postulado da compreensibilidade* de Michalski (1983a):

> "Os resultados da indução por computador devem ser descrições simbólicas das entidades dadas, sendo semântica e estruturalmente similares àquelas que um especialista humano poderia produzir observando as mesmas entidades. Os componentes dessas descrições devem ser compreensíveis como simples 'pedaços' de informação, diretamente interpretáveis em linguagem natural, bem como reportar conceitos quantitativos e qualitativos de maneira integrada."

Como regra prática, Michalski assume que os componentes de descrição, tais como regras ou nós em uma árvore de decisão, devem ser expressões contendo menos de cinco condições em uma conjunção; poucas condições em uma disjunção; no máximo um nível de parênteses; no máximo uma implicação; não mais de dois quantificadores e nenhuma recursão. Embora esses valores possam ser flexíveis, descrições geradas por indução dentro dos limites propostos são similares à representação do conhecimento humano e, portanto, fáceis de serem compreendidas. Embora tais medidas sejam simples de serem avaliadas, é importante salientar que elas são meramente sintáticas e que, muitas vezes, medidas semânticas devam ser consideradas (Pazzani 2000).

Em AM existem muitos algoritmos de aprendizado que induzem classificadores. Este trabalho se concentra em indutores que contribuem para a compreensão dos dados em contraste com indutores que visam apenas uma grande precisão. Por exemplo, a indução de regras ou árvores de decisão pode auxiliar médicos a compreenderem melhor os dados, enquanto uma rede neural convencional, mesmo com precisão similar, pode ser extremamente difícil de ser compreendida por seres humanos[3]. Por exemplo, no desenvolvimento de sistemas especialistas é importante que especialistas humanos possam, fácil e confiavelmente, verificar o conhecimento extraído e relacioná-lo ao seu próprio domínio de conhecimento. Além disso, algoritmos de aprendizado que induzem estruturas compreensíveis, contribuindo para a compreensão do domínio considerado, podem produzir conhecimento novo (Dietterich 1986).

Em resumo, este trabalho se concentra em aprendizado simbólico supervisionado para resolver problemas de classificação. O termo *simbólico* indica que os classificadores devem ser legíveis e interpretáveis por humanos — aqueles que satisfazem os critérios forte ou ultra-forte. O termo *supervisionado* sugere que algum processo, às vezes denominado *agente externo* ou *professor*, previamente rotulou os dados. Finalmente, o termo *classificação* denota o fato que o rótulo da classe é discreto, ou seja, consiste de valores nominais sem uma ordem definida.

1.2 Motivação

All men by nature desire knowledge.

—Aristotle, Metaphysics

É notório o fato que os seres humanos são naturalmente observadores. Normalmente, observamos um processo ocorrendo na natureza, ou mesmo criado pela mente humana, e pensamos em uma forma de compreendê-lo. Em algumas ocasiões, decidimos medir algumas características — ou atributos — do processo em questão, na esperança que isso facilite nossa compreensão.

[3]Existem, entretanto, vários mátodos desenvolvidos para a extração de regras a partir de redes neurais

Por exemplo, se o processo de interesse é o controle de uma caldeira, alguns possíveis atributos que podem ser medidos são a temperatura, a pressão, a quantidade de líquido de entrada. Se o processo de interesse consiste no diagnóstico de uma determinada doença em pacientes, possíveis atributos são a idade, o peso, a resistência da pele, a temperatura, a concentração de uma determinada substância no sangue e assim por diante.

Independentemente das características medidas, é possível manter esses atributos em cinco estágios básicos de armazenamento:

1. memória humana;
2. manual em papel[4], objetivando estender a memória humana ou a permanência dos dados ao longo do tempo;
3. arquivos dentro de um computador, incluindo arquivos texto e planilhas eletrônicas, cujo objetivo principal é armazenar os dados no formato digital;
4. base de dados, cujo objetivo principal é o armazenamento seguro e confiável bem como o acesso rápido aos dados;
5. *data marting* ou *data warehousing*, cujo objetivo principal é o armazenamento de grande volume de dados para suporte à tomada de decisão.

Para um determinado processo, é possível começar no primeiro estágio, passando pelos demais estágios até o último, dependendo da quantidade de dados sendo armazenada, bem como da tecnologia disponível.

Uma vez armazenados no formato digital, os dados de um processo são geralmente utilizados de uma maneira limitada, através de questões formuladas por um aplicativo ou através de um gerador de relatórios. Enquanto essa forma tradicional de interação é satisfatória para questões bem definidas ou fáceis de serem expressas através do aplicativo sendo utilizado, ela não foi projetada para suportar questões do tipo:

- É possível predizer o comportamento do processo sendo analisado?
- Como os dados podem ser usados para construir classificadores a partir do processo que os gerou?
- Como entender melhor os dados e utilizá-los para ganhar algum tipo de vantagem ou melhorar o processo?

[4]Embora outros meios de armazenamento possam ser usados, tais como pele de animais, pintura em rochas, etc.

Essas questões surgem naturalmente quando o usuário não sabe como descrever seu objetivo em termos de uma consulta tradicional. Neste caso, uma forma mais natural de interagir com os dados é na forma de exemplos — tipicamente mais factível, uma vez que os seres humanos acham mais natural esse tipo de interação. Isto é feito, normalmente, rotulando-se um subconjunto de todos os exemplos (ou registros) — denominado *conjunto de treinamento* — com as respectivas classes, e deixando que um algoritmo de Aprendizado de Máquina construa um classificador que separe as classes.

Após isso, é possível utilizar o classificador induzido para predizer a classe de novos exemplos (e não apenas o conjunto de treinamento). Como descrito por Breiman, Friedman, Olshen & Stone (1984), classificações derivadas de exemplos possuem dois propósitos básicos:

1. predizer a classe correspondente a novas medidas dos atributos, ou seja, predizer a classe de novos exemplos tão precisamente quanto possível;

2. compreender a relação estrutural existente entre a classe e os atributos medidos.

Nas próximas seções são descritas situações nas quais há um grande interesse na utilização de técnicas de Aprendizado de Máquina, dentre elas a Mineração de Dados, a aquisição de conhecimento e a melhoria do desempenho obtido quando comparadas com especialistas humanos.

1.2.1 Mineração de Dados

A Mineração de Dados consiste em uma das etapas do processo de Extração de Conhecimento de grande volume de dados. A Extração de Conhecimento é definida por Fayyad, Piatetsky-Shapiro & Smyth (1996a) como:

> "O processo não trivial de identificar padrões válidos, novos, potencialmente úteis e compreensíveis nos dados existentes."

Com o avanço nas tecnologias de armazenamento e melhorias nos processos de aquisição de dados, a quantidade de dados coletada nos últimos anos cresceu rapidamente. Alguns exemplos de grandes bases de dados criadas recentemente são:

1. a rede de supermercados Wal-Mart criou uma base de dados que armazena 20 milhões de transações diariamente;

2. a Mobil Oil Corporation está desenvolvendo uma base de dados capaz de armazenar mais de 100 terabytes de dados referentes à exploração de petróleo;

3. o sistema EOS da NASA, compreendendo satélites em órbita e outros instrumentos astronômicos, está projetado para gerar cerca de 50 gigabytes de imagens por hora;

4. o catálogo cósmico do observatório de Palomar contém bilhões de dados em imagens não processadas, com cerca de 3 terabytes.

Devido ao grande volume, a habilidade de automaticamente extrair informação interessante e compreender os dados é de relevada importância. Além disso, a tomada de decisão, utilizando consultas tradicionais em SQL, é difícil de ser expressa, ou seja, muitas consultas de interesse para os seres humanos são difíceis de serem elaboradas em SQL. Por exemplo, consultas do tipo:

1. "quais os usuários que poderão praticar alguma fraude", em uma base de dados na qual o campo *fraude* não se encontra definido;

2. "quais clientes gostariam de comprar o novo produto X", em uma base de dados de produtos vendidos por uma rede de supermercados;

3. "quais imagens (ou objetos contidos em uma imagem) são similares a uma dada imagem (ou objeto) Y".

1.2.2 Aquisição de Conhecimento

Os sistemas baseados em conhecimento e sistemas especialistas solucionam problemas que, normalmente, requerem habilidades humanas para resolvê-los. Um requisito importante nos sistemas especialistas, que nem sempre é necessário para outras atividades de aprendizado, é a habilidade de explicar suas decisões. Para tanto, os sistemas especialistas possuem uma grande base de conhecimento cuja construção é geralmente atribuída a um engenheiro de conhecimento. A construção manual de bases de conhecimento, ou aquisição de conhecimento explícito, é considerada um gargalo no desenvolvimento de sistemas especialistas (Rezende & Pugliesi 1998).

Uma motivação, para pesquisas em Aprendizado de Máquina, consiste na redução considerável do tempo gasto no processo de aquisição de conhecimento. Uma maneira de diminuir o tempo necessário para adquirir conhecimento é através do uso de exemplos reais — ou definidos por especialistas em um domínio de conhecimento específico — apresentando-os aos algoritmos de AM. Este processo é denominado de aquisição de conhecimento implícito. Porém, quando a quantidade de dados é grande, o uso direto de algoritmos de AM torna-se impraticável, ou pelo fato de restrições impostas pelos próprios algoritmos — tais como manter todo o conjunto de exemplos em memória principal — ou pelo enorme tempo computacional necessário para induzir um classificador.

Por exemplo, em 1986 foi desenvolvido o sistema especialista GASOIL para o projeto de separação de petróleo. Esta separação é feita por um sistema muito grande, complexo e dispendioso, cujo projeto depende de um grande número de atributos, incluindo proporções relativas de gás, petróleo e água, taxa de fluxo, pressão, densidade, viscosidade, temperatura, etc.

Naquela época, GASOIL era o maior sistema especialista comercial no mundo, contendo aproximadamente 2500 regras. A construção manual deste sistema teria tomado 10 pessoas/ano. Utilizando métodos de aprendizado por árvores de decisão, aplicados a uma base de dados de projetos existentes, o sistema foi desenvolvido com menos de 4 pessoas/mês (Michie 1986). Além disso, este sistema permitiu a economia de milhões de dólares e apresentou um desempenho melhor do que especialistas humanos, tópico da próxima seção.

1.2.3 Melhor Desempenho Comparado com Especialistas Humanos

Em algumas situações, o processo de aprendizado automático produz resultados que são superiores quando comparados com os resultados obtidos pelos seres humanos. Como exemplo, considere o projeto de um controlador automático para um sistema complexo. Uma forma de construir um controlador consiste em definir um modelo preciso da dinâmica do sistema e utilizar uma variedade de métodos formais (incluindo métodos de planejamento em Inteligência Artificial) para projetar um controlador que tenha certas propriedades garantidas. Uma forma alternativa consiste em simplesmente aprender o mapeamento correto de um estado do sistema para uma ação correta. Para sistemas muito complexos, tais como aeronaves, o desenvolvimento de um modelo detalhado pode ser inviável, o que leva a considerar a segunda alternativa.

Essa alternativa foi adotada por Sammut, Hurst, Kedzier & Michie (1992) para a tarefa de aprender a pilotar um Cessna em um simulador de vôo. Os dados foram gerados pela observação de três pilotos experientes em ação, em planos de vôo, repetidos 30 vezes para cada piloto. Um exemplo de treinamento era criado a cada momento em que o piloto tomava uma ação que alterasse uma das variáveis de controle, tais como *flaps* ou velocidade. No total, 90 000 exemplos foram obtidos, descritos por 20 atributos, bem como a ação tomada pelo piloto (usada como rótulo da classe). A partir desses exemplos, uma árvore de decisão foi induzida utilizando o algoritmo de AM C4.5. A árvore foi então convertida em código C e inserida no simulador de vôo, de modo que ele pudesse pilotar a aeronave sozinho.

Como resultado, o sistema não apenas aprendeu a voar, mas a voar melhor que seus professores humanos. Isso foi devido ao processo de aprendizado, que eliminou erros ocasionais cometidos pelos pilotos. Esses resultados sugerem que técnicas de Aprendizado de Máquina podem produzir controladores mais robustos que os convencionais, ou seja, do que os pilotos automáticos

programados manualmente. Para tarefas difíceis, tal como o vôo de helicópteros carregando cargas pesadas em ventos fortes, ainda não existem pilotos automáticos e pouquíssimos seres humanos são competentes nessas áreas. Tais tarefas são potencialmente situáveis para sistemas de aprendizado automático.

Um outro exemplo é o projeto SKICAT, cujo objetivo consiste em determinar a classe de objetos cósmicos presentes em chapas fotográficas, totalizando 3 terabytes de imagens (Fayyad, Djorgovski & Weir 1996). Este projeto também apresenta desempenho melhor que astrônomos na classificação de objetos através de imagens. Para a maioria dos objetos, os astrônomos não foram capazes de determinar a classe do objeto olhando apenas para as imagens apresentadas para SKICAT, necessitando olhar imagens com maior resolução.

Referenciando mais de 20 artigos nos quais Aprendizado de Máquina é aplicado na área médica — tais como oncologia, patologia do fígado, prognóstico de sobrevivência de pacientes com hepatite, urologia, diagnóstico de doenças de tiróide, reumatologia, cardiologia, neuropsicologia, ginecologia, entre outras — Kononenko (1993) salienta que:

> " ... tipicamente, regras de diagnóstico geradas automaticamente têm um desempenho e precisão sensivelmente melhores do que especialistas humanos."

1.3 Objetivos

> One machine can do the work of fifty ordinary men. No machine can do the work of one extraordinary man.
>
> —*Elbert Hubbard*

Os objetivos desta tese incluem (i) a avaliação da seleção de atributos e seu impacto em um classificados simbólico; (ii) a avaliação da composição de atributos guiada pelo conhecimento fornecido por um usuário/especialista e, finalmente, (iii) a proposta de uma metodologia capaz de promover a combinação de classificadores simbólicos em um classificador final também simbólico.

1.4 Organização

> Good order is the foundation of all things.
>
> —*Edmund Burke*

Esta tese está dividida em três grandes partes: seleção de atributos, composição de atributos e combinação de classificadores simbólicos. A seguir, a organização deste trabalho é descrita,

através do resumo dos próximos capítulos, finalizando com o histórico das atividades desenvolvidas no decorrer desta tese.

1.4.1 Resumo dos Capítulos

No Capítulo 2 são introduzidos alguns conceitos e definições sobre aprendizado supervisionado, no qual está baseado este trabalho, bem como uma notação padrão que é utilizada no decorrer desta tese. São também descritas as linguagens de representação freqüentemente utilizadas por algoritmos de AM.

No Capítulo 3 é descrita a metodologia adotada para avaliar e comparar algoritmos. Também são apresentados os conjuntos de exemplos, indutores e ferramentas utilizados em (ou relacionados a) este trabalho.

No Capítulo 4 é descrito o processo de Extração de Conhecimento de conjuntos de exemplos, normalmente de grande dimensão. O processo, basicamente, é composto por três etapas: Pré-processamento, Mineração de Dados e Pós-processamento.

No Capítulo 5 é descrita uma das atividades de pré-processamento para Extração de Conhecimento, a seleção de atributos, que tem como objetivo diminuir a dimensão dos dados originais. São descritos também diversos experimentos realizados utilizando *wrappers* e filtros para a seleção de atributos. Além disso, são analisadas algumas conseqüências do processo de seleção de atributos em um classificador simbólico.

No Capítulo 6 é considerada a combinação de atributos com o objetivo de melhorar o desempenho das hipóteses induzidas, outra atividade de pré-processamento para Extração de Conhecimento. É proposta uma metodologia para combinação de atributos utilizando conhecimento de um usuário/especialista e são descritos diversos experimentos realizados para testar essa metodologia.

No Capítulo 7 são mostradas as diferenças entre os *biases* de aprendizado e *bias* e variância estatísticos. Embora diferentes, é mostrado que existe um relacionamento entre eles e que o *bias* e a variância podem ser vistos como componentes do erro de um classificador. A construção de *ensembles*, uma das maneiras de reduzir o *bias*, a variância ou ambos, é também descrita nesse capítulo.

No Capítulo 8 são descritos alguns problemas resultantes do uso de *ensembles*, entre eles o fato que o emprego de *ensembles* claramente dificulta a compreensão humana sobre o comportamento do classificador final. Para tentar solucionar essa dificuldade, é proposto e desenvolvido o sistema XRULER. São também relatados os experimentos realizados para avaliar o desempenho do sistema XRULER.

No Capítulo 9 são apresentadas as conclusões deste trabalho juntamente com um resumo dos resultados obtidos.

No Apêndice A estão relacionadas as abreviaturas e no Apêndice B encontra-se a definição dos símbolos utilizados com freqüência no decorrer deste trabalho. Finalmente, são relacionadas as Referências Bibliográficas bem como o Índice remissivo.

1.4.2 Um Breve Histórico

Durante os últimos anos, muitos algoritmos de AM supervisionado foram desenvolvidos, alguns utilizando forte embasamento teórico, empírico ou uma combinação de ambos. No início, a preocupação principal dos algoritmos de AM era obter a melhor precisão possível com a disponibilidade de uma quantidade limitada de exemplos.

Com um número crescente de algoritmos de AM disponíveis, a dificuldade que surgiu foi o árduo trabalho necessário para conversão do formato de arquivos de exemplos, de descrição dos tipos de dados e outros para cada algoritmo específico. O projeto $\mathcal{MLC}$++ desenvolvido na Universidade de Stanford teve como objetivo facilitar o uso dos algoritmos de AM, padronizando o formato dos arquivos de entrada e realizando as conversões necessárias para os diversos indutores (Kohavi, Sommerfield & Dougherty 1994). Novamente, a preocupação principal da biblioteca $\mathcal{MLC}$++ foi voltada à precisão do processo de aprendizado.

Após solucionado o problema de conversão entre diversos indutores, novos desafios surgiram. Com a evolução tecnológica e a redução de custos dos computadores, tornou-se cada vez mais simples e barato armazenar informações, sejam elas coletadas manualmente ou através de algum processo de controle, gerando bases de dados da ordem de terabytes. Um desafio é como adequar os algoritmos de AM para trabalharem com muitos exemplos. Várias abordagens têm sido utilizadas para ultrapassar essa limitação. Entre elas tem-se o uso de paralelismo, desenvolvimento de novas técnicas orientada a bases de dados, redução da dimensão dos dados e amostragem dos dados (Araujo, Lopes & Freitas 1999; Freitas & Lavington 1998).

Outro desafio consiste em como extrair, automaticamente, algum tipo de conhecimento que possa estar embutido no conjunto de exemplos, mas que não é trivial de ser encontrado através de técnicas convencionais, automáticas ou não. Este último desafio consiste em uma abordagem totalmente diferente daquela que vinha sendo adotada até então: além da precisão, a forma na qual o conhecimento extraído é expresso também passou a ter importância.

Extração de Conhecimento de Bases de Dados, também é conhecida como *Knowledge Discovery in Databases* — KDD — constitui um campo recente de pesquisa em Inteligência Artificial. O termo KDD foi criado em 1989 por Piatetsky-Shapiro (1989) quando também ocorreu o pri-

meiro *workshop* em KDD; esse *workshop* evoluiu para uma conferência anual internacional, a primeira ocorrida em 1995: *The First International Conference on Knowledge Discovery and Data Mining*, com Usama Fayyad liderando as pesquisas nesta área e colocando a Mineração de Dados como uma etapa no processo geral de KDD (Fayyad, Piatetsky-Shapiro & Smyth 1996a; Fayyad, Piatetsky-Shapiro & Smyth 1996b; Fayyad, Piatetsky-Shapiro & Smyth 1996c). Além disso, Fayyad também formalizou as demais etapas do processo de Extração de Conhecimento. Mais recentemente, uma proposta alternativa foi efetuada por Weiss & Indurkhya (1998) que, por se tratar de um livro, fornece maiores esclarecimentos sobre vários conceitos. O Capítulo 4, que trata sobre o processo de KDD, consiste em uma união de todos esses conceitos.

Com base no processo de KDD, inicialmente concentramos os estudos na etapa de pré-processamento focalizando-se na redução da dimensão dos dados. Dentre os métodos para redução, o que provoca uma redução maior no espaço de descrição de exemplos é aquele que remove atributos. Esse método é conhecido como seleção de um subconjunto de atributos ou *Feature Subset Selecion* — FSS. Dentre as formas de realizar a seleção de atributos, optou-se, neste trabalho, pelo uso de *wrappers* e filtros, já que não envolvem modificações nos algoritmos de AM existentes. Embora o apelo ao ganho de precisão de *wrappers* sobre filtros fosse grande (Kohavi & Sommerfield 1995; Kohavi & John 1997), os experimentos por nós efetuados mostraram que o uso de *wrappers* para Mineração de Dados é inviável (exceto para pequenos conjuntos de dados), devido ao seu alto custo computacional, sendo filtros mais adequados, com perda de precisão não significativa. Os experimentos com *wrappers* e filtros fomentaram novos experimentos adicionais, com os quais constatamos que a retirada de atributos geralmente aumenta o tamanho (número de regras) ou a complexidade (número de condições nas regras) do classificador simbólico obtido. Os resultados mais significativos sobre seleção de atributos encontram-se no Capítulo 5. Experimentos realizados sobre FSS geraram os seguintes trabalhos (Baranauskas & Monard 1998a; Baranauskas & Monard 1998b; Baranauskas, Monard & Horst 1999a; Baranauskas, Monard & Horst 1999b; Baranauskas & Monard 1999; Lee, Monard & Baranauskas 1999).

Com os resultados obtidos pela seleção de atributos foram conduzidos estudos no sentido oposto: combinação de atributos ou aprendizado construtivo. Existem poucos sistemas de aprendizado construtivo, sendo a maioria muito específica a um domínio restrito de conhecimento. Assim, utilizando combinação de atributos guiada pelo conhecimento do usuário/especialista foi proposta uma metodologia por Lee, Monard & Baranauskas (2000) e experimentos foram conduzidos para testar a metodologia, os quais podem ser encontrados no Capítulo 6.

Concluídas as pesquisas sobre seleção e combinação de atributos, voltamos a atenção para

a etapa de Mineração de Dados. Bem conhecido pela comunidade de KDD, o processo definido por Fayyad, Djorgovski & Weir (1996) e utilizado na catalogação de objetos cósmicos obteve, como já mencionado, grande sucesso. Partindo desse processo o nosso interesse foi descobrir porque isso acontecia e se seria possível estender tal processo.

Para isso, realizou-se um estudo sobre seis indutores bem difundidos na comunidade de AM, os quais induzem classificadores simbólicos (regras ou árvores de decisão), considerando a escolha de atributos para a construção do classificador, tratamento de valores desconhecidos ou exemplos com ruído, bem como o processo de classificação (atribuição de uma classe) de um novo exemplo. Esse estudo resultou no relatório técnico Baranauskas & Monard (2000d), fornecendo uma abordagem unificada para os seis indutores. Paralelamente, foi efetuado outro estudo revisando alguns conceitos e métodos usados em AM, resultando no relatório técnico Baranauskas & Monard (2000c). Partes desses relatórios que contribuíram de forma significativa para este trabalho são apresentadas, em fragmentos ou estendidas, nos Capítulos 2, 3 e 7.

Com isso, surgiu a proposta de estender o processo proposto por Fayyad, Djorgovski & Weir (1996): no processo original usado por Fayyad, apenas um único indutor foi utilizado, gerando como classificador uma árvore de decisão sem poda, que posteriormente foi transformada em um conjunto de regras. Entretanto, resultados teóricos e empíricos mostram que não existe um único algoritmo que pode apresentar, de maneira uniforme, o melhor desempenho em todos os domínios (Dietterich 1997a; Kohavi, Sommerfield & Dougherty 1996; Schaffer 1994).

Com base nisso, uma extensão, por nós proposta, foi a de utilizar vários indutores no processo de aprendizado (Baranauskas, Monard & Batista 2000; Baranauskas & Monard 2000a; Baranauskas & Monard 2000b). Entretanto, havia uma dificuldade existente é que cada algoritmo de aprendizado induz um classificador cuja forma sintática difere consideravelmente dos demais. Dessa forma, em um trabalho conjunto entre Prati, Baranauskas & Monard (2001b), foi definido o formato padrão de regras PBM, descrito no Capítulo 8, para o qual todo classificador simbólico é transformado.

O processo descrito anteriormente gera uma quantidade considerável de regras, que podem ser selecionadas através de diversos métodos. Por exemplo, o usuário pode procurar no conjunto de regras aquelas que são as mais indicadas para o problema sendo abordado, baseado em sua experiência. Todavia, para um grande conjunto de regras, essa avaliação manual torna-se inviável. Para efetuar essa seleção de forma automática, foi por nós proposto e desenvolvido o sistema XRULER, descrito no Capítulo 8.

1.5 Considerações Finais

Doubt is the father of invention.

—Galileo Galilei

Um ponto interessante sobre os seres humanos está relacionado à sua habilidade de fazer generalizações precisas a partir de fatos. Além disso, o ser humano é capaz de encontrar estruturas ou padrões apenas observando um processo (aparentemente caótico) do mundo real. Em ciência de computação, essa habilidade pode ser obtida a partir de um conjunto de exemplos, fornecidos pelo usuário ou por um processo do mundo real, através da inferência indutiva. A indução, mesmo sendo o recurso mais utilizado pelo cérebro na produção de conhecimento novo, deve ser utilizada cuidadosamente, pois, se os exemplos não forem bem escolhidos, as hipóteses induzidas podem não ter valor.

O aprendizado indutivo supervisionado, no qual os rótulos das classes são discretos, é conhecido como classificação, sendo que diversos fatores motivam seu estudo. Por exemplo, as abordagens convencionais para o desenvolvimento de bases de conhecimento envolvem uma formalização manual do conhecimento do especialista e subseqüente codificação em estruturas de dados apropriadas. Uma aplicação importante de AM visa a construção (semi) automática de bases de conhecimento utilizando inferência indutiva. De fato, AM pode fornecer uma melhoria das técnicas atuais e um embasamento para o desenvolvimento de abordagens alternativas de aquisição de conhecimento (Flach 2000).

Neste capítulo, foram apresentados alguns conceitos introdutórios sobre o aprendizado supervisionado por indução. Entretanto, alguns conceitos e definições adicionais são usados no decorrer desta tese e são descritos, de forma unificada, no capítulo que se segue.

Capítulo 2

Aprendizado Supervisionado: Conceitos e Definições

All things good to know are difficult to learn.

—*Greek Proverb*

Alguns conceitos sobre a hierarquia de aprendizado foram introduzidos no capítulo anterior, fornecendo uma distinção entre sistemas de aprendizado. Este capítulo tem como objetivo descrever o problema de classificação no aprendizado supervisionado, assim como fornecer algumas definições de termos amplamente usados, tanto neste trabalho como na literatura de AM.

2.1 Definições

The beginning of wisdom is the definition of terms.

—*Socrates*

Indutor Informalmente, o objetivo de um indutor (ou programa de aprendizado ou algoritmo de indução) consiste em extrair um *bom* classificador a partir de um conjunto de exemplos rotulados. A saída do indutor, o classificador, pode então ser usada para classificar exemplos novos (ainda não rotulados) com a meta de predizer corretamente o rótulo de cada um. Após isso, o classificador pode ser avaliado considerando sua precisão (Salzberg 1995b; Dietterich 1997c), compreensibilidade ou grau de interesse (Freitas 1999; Horst 1999; Horst & Monard 2000), velocidade de aprendizado, requisitos de armazenamento, grau de compactação ou qualquer outra propriedade desejável que determine quão bom e apropriado ele é para a tarefa em questão (Michie, Spiegelhalter & Taylor 1994). Embora

este trabalho se concentre em classificação, vários aspectos aqui definidos se aplicam aos casos de regressão. Maiores detalhes sobre alguns indutores, incluindo aqueles utilizados em experimentos apresentados neste trabalho, são fornecidos no Capítulo 3.

Exemplo Um exemplo, também denominado *caso*, *registro* ou *dado* na literatura, é uma tupla de valores de atributos (ou um vetor de valores de atributos). Um exemplo descreve o objeto de interesse, tal como um paciente, dados médicos sobre uma determinada doença ou histórico de clientes de uma dada companhia.

Atributo Um atributo, algumas vezes denominado *campo*, descreve alguma característica ou aspecto de um exemplo. Normalmente, há pelo menos dois tipos de atributos: nominal, quando não existe uma ordem entre os valores (*e.g.*, cor: vermelho, verde, azul) e contínuo, quando existe uma ordem linear nos valores (*e.g.*, peso $\in \mathbb{R}$, um número real).

Para qualquer tipo de atributo, usualmente existe também um símbolo importante que significa *desconhecido*, ou seja, a ausência de um valor para aquele atributo. Este símbolo especial é bem diferente, por exemplo, do valor zero (às vezes usado para números) ou de cadeias de caracteres vazias. Na maioria dos indutores disponíveis, este valor é representado por um ponto de interrogação '**?**'.

Um outro símbolo especial, mesmo não sendo reconhecido por vários indutores, é o *não-se-aplica*. Por exemplo, para o atributo *número de gestações*, pode ser utilizado o símbolo *não-se-aplica* caso o paciente seja do sexo masculino. Normalmente, este símbolo é representado por um ponto de exclamação '**!**'.

Além disso, vários indutores assumem que os atributos originais que descrevem os exemplos são *relevantes* o suficiente para aprender a tarefa em questão. Entretanto, alguns atributos podem não ser diretamente relevantes e outros até *irrelevantes*. Um atributo é irrelevante se existe uma descrição *completa* e *consistente* das classes a serem aprendidas que não usa aquele atributo (Michalski & Kaufman 1998). Assim, um atributo *não essencial* é relevante ou irrelevante mas pode ser dispensado quando o aprendizado tem início (Lee 2000).

Um ponto importante a ser considerado é a escolha de atributos com boa capacidade preditiva. Não importa qual método seja empregado, os conceitos que podem ser aprendidos estão à mercê dos dados e da qualidade dos atributos (Weiss & Kulikowski 1991). Por exemplo, para a tarefa de determinar se uma pessoa está ou não com gripe, pode-se escolher atributos com baixo poder preditivo, tais como (cor-do-cabelo, cor-do-olho, modelo-do-carro, número-de-filhos) ou atributos com alto poder preditivo, tais como (temperatura, resistência-da-pele, exame-do-pulmão). Para esta tarefa específica, no segundo

caso, melhores previsões em exemplos não-rotulados provavelmente ocorrerão do que com o primeiro conjunto de atributos.

Classe No aprendizado supervisionado todo exemplo possui um atributo especial, o rótulo ou *classe*, que descreve o fenômeno de interesse, isto é, a meta que se deseja aprender e poder fazer previsões a respeito. Um exemplo não-rotulado consiste do exemplo, exceto o rótulo, ou seja, um vetor de valores dos atributos. Os rótulos são tipicamente pertencentes a um conjunto discreto (nominal) de classes $\{C_1, C_2, \ldots, C_k\}$ no caso de *classificação* ou de valores reais no caso de *regressão*.

Conjunto de Exemplos Um *conjunto de exemplos* ou *dataset*[1] é composto por exemplos contendo valores de atributos bem como a classe associada. Na Tabela 2.1 é mostrado o formato padrão de um conjunto de exemplos T com n exemplos e m atributos. Nessa tabela, a linha i refere-se ao i-ésimo exemplo ($i = 1, 2, \ldots, n$) e a entrada x_{ij} refere-se ao valor do j-ésimo ($j = 1, 2, \ldots, m$) atributo X_j do exemplo i.

	X_1	X_2	$\cdots$	X_m	Y
T_1	x_{11}	x_{12}	$\cdots$	x_{1m}	y_1
T_2	x_{21}	x_{22}	$\cdots$	x_{2m}	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
T_n	x_{n1}	x_{n2}	$\ldots$	x_{nm}	y_n

Tabela 2.1: Conjunto de exemplos no formato atributo-valor

Como pode ser notado, exemplos são tuplas $T_i = (x_{i1}, x_{i2}, \ldots, x_{im}, y_i) = (\vec{x}_i, y_i)$ também denotados por (x_i, y_i), onde fica subentendido o fato que x_i é um vetor. A última coluna, $y_i = f(x_i)$, é a função que tenta-se predizer a partir dos atributos. Observa-se que cada x_i é um elemento do conjunto $\mathsf{dom}(X_1) \times \mathsf{dom}(X_2) \times \ldots \times \mathsf{dom}(X_m)$, onde $\mathsf{dom}(X_j)$ é o domínio do atributo X_j e y_i pertence a uma das k classes, isto é, $y_i \in \{C_1, C_2, \ldots, C_k\}$.

Usualmente, um conjunto de exemplos é dividido em dois subconjuntos disjuntos: o *conjunto de treinamento* que é usado para o aprendizado do conceito e o *conjunto de teste* usado para medir o grau de efetividade do conceito aprendido. Os subconjuntos são normalmente disjuntos para assegurar que as medidas obtidas utilizando o conjunto de teste sejam de um conjunto diferente do usado para realizar o aprendizado, tornando a medida estatisticamente válida.

Note que, após induzir uma hipótese, é possível avaliá-la no conjunto de treinamento bem

[1] O termo *dataset* é usado como sinônimo e de forma indistinta de *conjunto de exemplos* neste trabalho, por se tratar de um termo amplamente difundido em AM.

como no conjunto de teste. É usual denominar as medidas de desempenho de um classificador efetuadas sobre o conjunto de treinamento como *aparentes* (também conhecidas como medidas de *resubstituição*) e as medidas efetuadas sobre o conjunto de teste como medidas *reais* (ou *verdadeiras*). Por exemplo, caso a medida seja o *erro*, pode-se ter o *erro aparente* e o *erro verdadeiro*. Para a maioria das hipóteses, a medida aparente é um estimador ruim do seu desempenho futuro, uma vez que ela tem a tendência de possuir um *bias* otimista.

Ruído É comum, no mundo real, trabalhar com dados imperfeitos. Eles podem ser derivados do próprio processo que gerou os dados, do processo de aquisição de dados, do processo de transformação ou mesmo devido a classes rotuladas incorretamente (*e.g.*, exemplos com os mesmos valores de atributos mas com classes diferentes). Neste caso, diz-se que existe *ruído* nos dados. Além disso, atributos que não são mais preditivos que o acaso para a tarefa sendo considerada podem ser considerados ruído (Weiss & Kulikowski 1991, Chapter 1).

Conhecimento do Domínio Em geral, o conhecimento prévio do domínio, ou *background knowledge*, inclui informação a respeito dos valores válidos dos atributos, um critério de preferência para a escolha entre possíveis atributos ou mesmo hipóteses. O conhecimento do domínio pode também incluir restrições de relacionamentos entre atributos, regras para a geração de conceitos de alto nível, novos atributos possivelmente derivados dos atributos originais, assim como alguma hipótese inicial. Nem todos os indutores são capazes de utilizar conhecimento do domínio quando aprendem conceitos, isto é, eles usam apenas os exemplos fornecidos. Observe que o número de unidades ocultas e conexões, assim como a topologia em uma rede neural, é uma forma de conhecimento do domínio.

Classificador Dado um conjunto de exemplos de treinamento, um indutor gera como saída um *classificador* (também denominado *hipótese* ou *descrição de conceito*) de forma que, dado um novo exemplo, ele possa predizer precisamente sua classe.

Formalmente, em classificação, um exemplo é um par $(x_i, f(x_i))$ onde x_i é a entrada e $f(x_i)$ é a saída. A tarefa de um indutor é, dado um conjunto de exemplos, induzir uma função h que aproxima f, normalmente desconhecida. Neste caso, h é chamada uma *hipótese* sobre a função objetivo f, ou seja, $h(x_i) \approx f(x_i)$.

Na Figura 2.1 na página oposta é dado um exemplo desse conceito (Russel & Norvig 1995). Suponha que os exemplos são pontos no plano mostrados como círculos em (a), sendo que a meta é encontrar uma função $h(x_i)$ com boa aproximação dos pontos. Uma

aproximação possível, para a verdadeira função f, poderia ser obtida conectando-se cada exemplo com o próximo através de linhas retas (b). De fato, essa seria uma aproximação muito simplificada para a função f. Assim sendo, seria possível aproximar f por uma função polinomial suave, como pode ser visto em (c). Mesmo assim, esta hipótese poderia ser considerada muito complicada pelo usuário final. Dessa forma, uma hipótese mais simples poderia ser induzida como a mostrada em (d), que completamente ignora um exemplo, tratando-o como ruído.

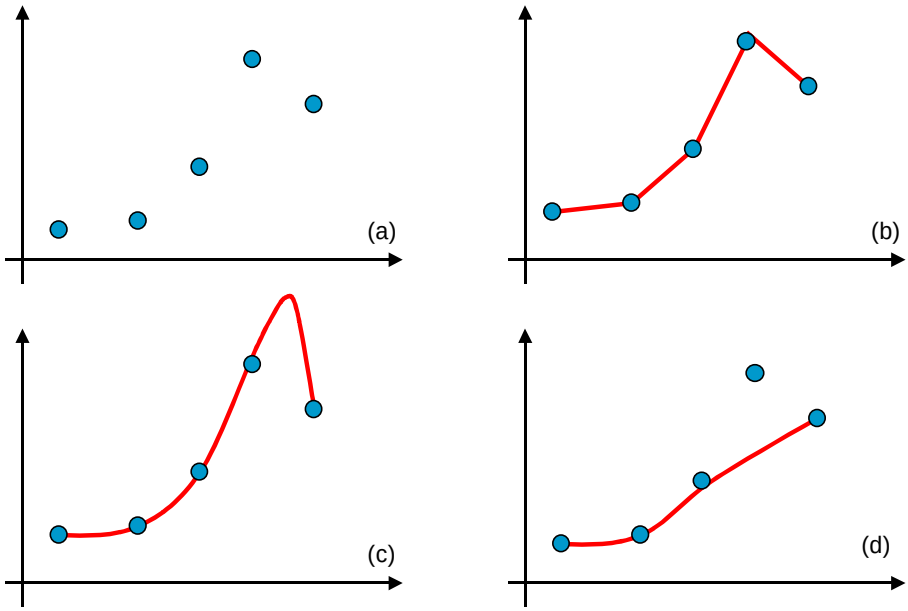


Figura 2.1: Dados os exemplos em (a), representados como pontos na forma $(x_i, f(x_i))$, (b), (c) e (d) são possíveis hipóteses consistentes h para aproximar a verdadeira função objetivo f, que é desconhecida

Bias Ainda considerando a Figura 2.1, já que a verdadeira função f é desconhecida, existem muitas escolhas possíveis para h, mas sem conhecimento adicional não é possível escolher entre as hipóteses (b), (c) ou (d). Qualquer preferência de uma hipótese sobre outra, além da simples consistência com os exemplos, é denominada um *bias* de aprendizado (Russel & Norvig 1995). Devido ao fato que quase sempre existe um número grande de hipóteses consistentes, todos os indutores possuem alguma forma de *bias*. De fato, aprendizado sem *bias* é impossível (Mitchell 1982). Maiores detalhes sobre o *bias* de AM são apresentados na Seção 7.2 na página 117.

Modo de Aprendizado Sempre que todo o conjunto de treinamento deva estar presente para o aprendizado, o modo de aprendizado de um algoritmo é *não-incremental*, também co-

nhecido como modo *batch*. Por outro lado, se o indutor não necessitar construir a hipótese a partir do início, quando novos exemplos são adicionados ao conjunto de treinamento, o modo de aprendizado é *incremental*. Portanto, no modo incremental o indutor apenas tenta atualizar a hipótese antiga sempre que novos exemplos são adicionados ao conjunto de treinamento.

Em geral, o aprendizado não-incremental deve fornecer resultados melhores, uma vez que é permitido, ao indutor, o acesso a todos os exemplos de treinamento de uma única vez, possibilitando que ele otimize suas decisões. Entretanto, se o tempo computacional é um fator importante e novos exemplos são freqüentemente adicionados ao conjunto de treinamento, a aprendizado incremental pode ser considerado para poupar tempo.

Espaço de Descrição Como já descrito, um exemplo $(x_{i1}, x_{i2}, \ldots, x_{im}, y_i) = (\vec{x}_i, y_i)$ possui m atributos $x_{i1}, x_{i2}, \ldots, x_{im}$ que podem ser interpretados como um vetor $\vec{x}_i$, no qual cada atributo x_{ij} corresponde a uma coordenada em um espaço m-dimensional (ou *espaço de descrição*), onde $1 \leq i \leq n$ e n é o número de exemplos. Além disso, cada ponto no espaço de descrição pode ser rotulado com a correspondente classe y_i associada.

Sob este ponto de vista, um classificador divide este espaço em regiões, cada região rotulada com uma classe. Um novo exemplo é classificado determinando-se a região onde o ponto correspondente se localiza e atribuindo, ao exemplo, a classe associada com aquela região.

Na Figura 2.2 na próxima página é mostrado um exemplo para duas classes $\{\mathsf{o}, +\}$ e dois atributos $\{X_1, X_2\}$. Suponha em (a) que o classificador induzido, na forma simbólica, seja dado por '**if** $X_1 < 5$ **and** $X_2 < 8$ **then** classe = o **else** classe = +'. Este classificador divide o espaço de descrição em duas regiões: a região dentro do retângulo onde $X_1 < 5$ e $X_2 < 8$ (considerando valores positivos para ambos atributos) e a região fora do retângulo. Assim, em (b) um novo exemplo não-rotulado '*' com $(X_1, X_2) = (2{,}5, 4)$ será classificado como classe o uma vez que ele se localiza dentro da região que define aquela classe.

Erro e Precisão Uma medida de desempenho comumente usada é a *taxa de erro* de um classificador h, também conhecida como *taxa de classificação incorreta* e denotada por $\mathsf{err}(h)$. Usualmente, a taxa de erro é obtida utilizando (2.1), a qual compara a classe verdadeira de cada exemplo com o rótulo atribuído pelo classificador induzido. O operador $\parallel E \parallel$ retorna 1 se a expressão E for verdadeira e zero caso contrário, e n é o número de exemplos. O complemento da taxa de erro, a *precisão* do classificador, denotada por $\mathsf{acc}(h)$ é dada por (2.2).

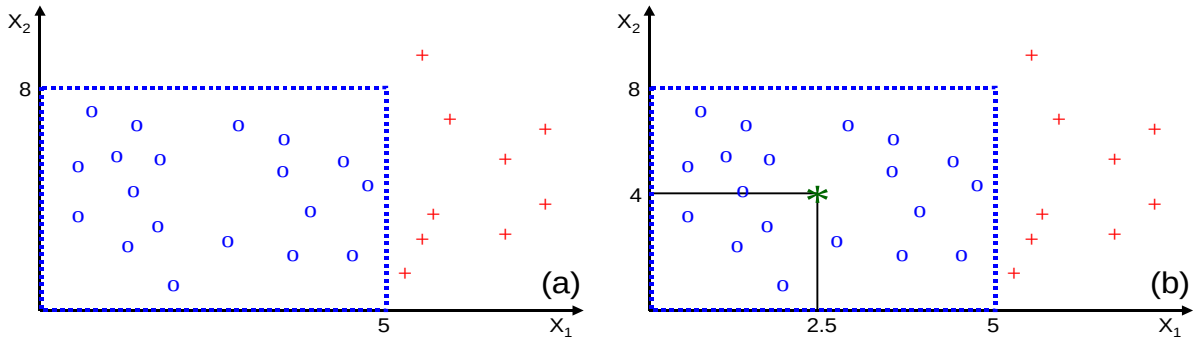


Figura 2.2: Um classificador divide o espaço de descrição em regiões, cada região rotulada com uma classe (a); o exemplo não-rotulado '*' é classificado de acordo com a região em que se localiza (b)

$$\mathsf{err}(h) = \frac{1}{n}\sum_{i=1}^{n} \parallel y_i \neq h(x_i) \parallel \tag{2.1}$$

$$\mathsf{acc}(h) = 1 - \mathsf{err}(h) \tag{2.2}$$

Para problemas de regressão, o *erro da hipótese* (err) pode ser estimado calculando-se a distância entre o valor real com o atribuído pela hipótese induzida. Usualmente, duas medidas são comumente usadas: o *erro médio quadrado* (mse-err ou *mean squared error*) e a *distância absoluta média* (mad-err ou *mean absolute distance*), dadas por (2.3) e (2.4), respectivamente.

$$\mathsf{mse\text{-}err}(h) = \frac{1}{n}\sum_{i=1}^{n} (y_i - h(x_i))^2 \tag{2.3}$$

$$\mathsf{mad\text{-}err}(h) = \frac{1}{n}\sum_{i=1}^{n} |y_i - h(x_i)| \tag{2.4}$$

Distribuição de Classes Dado um conjunto de exemplos T, é possível calcular sua *distribuição de classes*. Para cada classe C_j sua distribuição $\mathsf{distr}(C_j)$ é calculada como sendo o número de exemplos em T que possuem classe C_j dividido pelo número total de exemplos n, ou seja, a proporção de exemplos em cada classe, dada por (2.5).

$$\mathsf{distr}(C_j) = \frac{1}{n}\sum_{i=1}^{n} \parallel y_i = C_j \parallel \tag{2.5}$$

Por exemplo, se um conjunto com 100 exemplos possui 60 exemplos da classe C_1, 15 exemplos da classe C_2 e 25 exemplos da classe C_3 então sua distribuição de classes é $\mathsf{distr}(C_1, C_2, C_3) = (0{,}60,\ 0{,}15,\ 0{,}25) = (60{,}00\%,\ 15{,}00\%,\ 25{,}00\%)$. Nesse exemplo, a classe C_1 é a classe *majoritária* ou *prevalente* e a classe C_2 é a classe minonitária.

Erro Majoritário Uma vez calculada a distribuição de classes em um conjunto de exemplos T, é possível calcular seu *erro majoritário* utilizando (2.6).

$$\mathsf{maj\text{-}err}(T) = 1 - \max_{i=1,\ldots,k} \mathsf{distr}(C_i) \tag{2.6}$$

No exemplo anterior com $\mathsf{distr}(C_1, C_2, C_3) = (60{,}00\%,\ 15{,}00\%,\ 25{,}00\%)$, o erro majoritário é $\mathsf{maj\text{-}err}(T) = 1 - 0{,}60 = 40{,}00\%$. Como pode ser observado, o erro majoritário de um conjunto de exemplos é independente do algoritmo de aprendizado. Ele fornece um limiar máximo abaixo do qual o erro de um classificador deve ficar.

Prevalência de Classe Um ponto muito importante em AM refere-se ao desbalanceamento de classes em um conjunto de exemplos. Por exemplo, suponha um conjunto de exemplos T com a seguinte distribuição de classes $\mathsf{dist}(C_1, C_2, C_3) = (99{,}00\%,\ 0{,}25\%,\ 0{,}75\%)$, com *prevalência da classe* C_1. Um classificador simples que classifique sempre novos exemplos como pertencentes à classe majoritária C_1 teria uma precisão de 99,00% ($\mathsf{maj\text{-}err}(T) = 1{,}00\%$). Isto pode ser indesejável quando as classes minoritárias são aquelas que possuem uma informação muito importante, por exemplo, supondo C_1: paciente normal, C_2: paciente com doença A e C_3: paciente com doença B.

É importante estar ciente, quando se trabalha com conjuntos de exemplos desbalanceados, que é desejável utilizar uma medida de desempenho diferente da precisão (Kubat, Holte & Matwin 1998). Isto deve-se ao fato que a maioria dos sistemas de aprendizado é projetada para otimizar a precisão. Com isso, normalmente os algoritmos apresentam um desempenho ruim se o conjunto de treinamento encontra-se fortemente desbalanceado, pois os classificadores induzidos tendem a ser altamente precisos nos exemplos da classe majoritária, mas freqüentemente classificam incorretamente exemplos das classes mino-

ritárias. Algumas técnicas foram desenvolvidas para lidar com esse problema, tais como a introdução de custos de classificação incorreta (explicada mais adiante), a remoção de exemplos redundantes ou prejudiciais ou ainda a detecção de exemplos de borda e com ruído (Kubat & Matwin 1997; Kubat, Holte & Matwin 1997; Batista, Carvalho & Monard 2000).

Under- e Overfitting Ao induzir uma hipótese, a partir dos dados disponíveis, é possível que a hipótese seja muito específica para o conjunto de treinamento utilizado. Como o conjunto de treinamento é apenas uma amostra de todos os exemplos, é possível induzir hipóteses que melhorem seu desempenho no conjunto de treinamento, enquanto pioram o desempenho em exemplos diferentes daqueles pertencentes ao conjunto de treinamento. Nesta situação, o erro (ou outra medida) em um conjunto de teste independente leva a um desempenho ruim da hipótese (Weiss & Indurkhya 1998). Neste caso, diz-se que a hipótese ajusta-se em excesso ao conjunto de treinamento ou que houve um *overfitting*.

Por exemplo, em teoria o melhor classificador é equivalente a uma consulta em uma tabela, considerando a tabela composta pelos exemplos do conjunto de treinamento, denominado classificador de Bayes. Esse classificador tem uma taxa de erro aparente igual a zero, uma vez que não há erros na avaliação dos dados de treinamento (assumindo exemplos com os mesmos valores dos atributos pertencentes à mesma classe, ou seja, exemplos sem ruído). Mas quando novos exemplos são aplicados a esse classificador, será extremamente difícil obter uma taxa de erro igual a zero, devido ao grande número de possíveis combinações.

Por outro lado, também é possível que poucos exemplos representativos sejam dados ao sistema de aprendizado (por exemplo, algoritmos de indução de árvores de decisão ou de regras) ou o usuário pré-defina o tamanho do classificador como muito pequeno (por exemplo, um número insuficiente de neurônios e conexões são definidos em uma rede neural ou um alto fator de poda é definido para uma árvore de decisão) ou uma combinação de ambos. Portanto, é também possível induzir hipóteses que possuam uma melhora de desempenho muito pequena no conjunto de treinamento, assim como em um conjunto de teste. Neste caso, diz-se que a hipótese ajusta-se muito pouco ao conjunto de treinamento ou que houve um *underfitting*.

O impacto de *under-* e *overfitting* é ilustrado na Figura 2.3 na página seguinte. O eixo horizontal da figura indica a complexidade (tamanho) da hipótese (*e.g.*, número de nós em uma árvore de decisão, número de neurônios ou pesos em uma rede neural). O eixo vertical indica o erro das previsões da hipótese. A linha pontilhada indica o erro da hipótese

sobre o conjunto de treinamento, enquanto a linha sólida indica o erro medido sobre o conjunto de teste, o qual não está incluído no conjunto de treinamento. Como esperado, o erro sobre o conjunto de treinamento (erro aparente) diminui monotonicamente à medida que a hipótese é construída. Entretanto, o erro medido sobre o conjunto de teste (erro verdadeiro) primeiramente diminui até $N2$ e então aumenta.

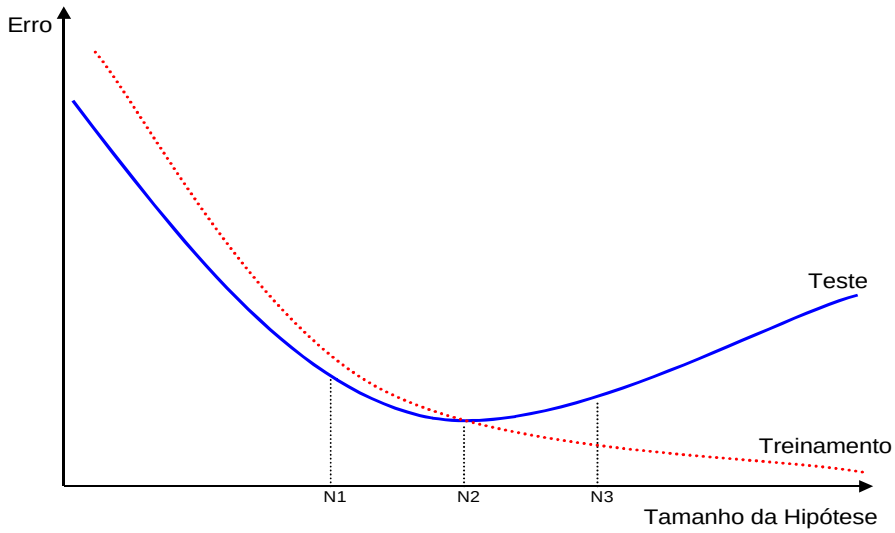


Figura 2.3: A relação entre o tamanho da hipótese e o erro

Considerando a Figura 2.3, uma questão que surge é qual hipótese deve ser escolhida. O princípio da navalha de Ockham fornece uma pista (Kearns & Vazirani 1994). Ele afirma que *a hipótese mais apropriada é aquela mais simples que seja consistente com todas as observações*. De fato, isto significa que, outros fatores sendo iguais, uma hipótese simples que seja consistente com os exemplos é preferível a uma hipótese complexa.

Overtuning O excesso de ajuste de um algoritmo, ou *overtuning*, pode ocorrer quando o desenvolvedor ajusta um algoritmo de aprendizado, ou os seus parâmetros, muito bem para otimizar seu desempenho em todos os exemplos disponíveis. Isso pode ser uma técnica prejudicial quando todos os exemplos disponíveis são usados para o desenvolvimento ou ajuste do algoritmo. Da mesma forma que o *overfitting*, *overtuning* pode ser detectado e evitado utilizando-se apenas uma parte dos exemplos disponíveis para a execução do indutor e o restante dos exemplos para um teste final do classificador induzido. De fato, assim como o *overfitting*, se os exemplos não são separados em conjuntos de treinamento e teste, permitindo uma avaliação final independente, o desempenho do sistema não pode ser usado de forma confiável como uma estimativa do desempenho futuro do sistema.

Poda Poda é uma técnica de lidar com ruído e *overfitting* quando a hipótese induzida é expressa por regras ou árvores de decisão. A idéia geral consiste em lidar com o problema de *overfitting* através do aprendizado de uma hipótese bem genérica a partir do conjunto de treinamento de forma a melhorar o desempenho em exemplos não vistos. Há, basicamente, dois métodos de poda:

1. *pré-poda*: durante a geração da hipótese, alguns exemplos de treinamento são deliberadamente ignorados, de forma que a hipótese final não classifique todos os exemplos de treinamento corretamente;
2. *pós-poda*: inicialmente, uma hipótese que explica os exemplos de treinamento é gerada. Após isso, a hipótese é generalizada através da eliminação de algumas partes, tais como o corte de alguns ramos em uma árvore de decisão ou algumas condições nas regras induzidas.

Completude e Consistência Uma vez induzida, uma hipótese pode ser avaliada a respeito de sua *completude*, se ela classifica todos os exemplos e *consistência*, se ela classifica corretamente os exemplos. Portanto, dada uma hipótese, ela pode ser: (a) completa e consistente; (b) incompleta e consistente; (c) completa e inconsistente ou (d) incompleta e inconsistente. Na Figura 2.4 na página seguinte é dado um exemplo dos quatro casos mencionados, considerando dois atributos X_1 e X_2, três classes (o, +, *), bem como as hipóteses induzidas para cada classe. Elas são representadas por duas regiões indicadas por linhas sólidas para a classe o, duas regiões indicadas por linhas pontilhadas para a classe + e uma região indicada por linhas tracejadas para a classe *.

Matriz de Confusão A matriz de confusão de uma hipótese h oferece uma medida efetiva do modelo de classificação, ao mostrar o número de classificações corretas *versus* as classificações preditas para cada classe, sobre um conjunto de exemplos T. Como mostrados na Tabela 2.2 na próxima página, os resultados são totalizados em duas dimensões: classes verdadeiras e classes preditas, para k classes diferentes $\{C_1, C_2, \ldots, C_k\}$. Cada elemento $M(C_i, C_j)$ da matriz, $i, j = 1, 2, \ldots, k$, calculado por (2.7), representa o número de exemplos de T que realmente pertencem à classe C_i, mas foram classificados como sendo da classe C_j.

$$M(C_i, C_j) = \sum_{\{\forall (x,y) \in T \,:\, y = C_i\}} \| \; h(x) = C_j \; \| \tag{2.7}$$

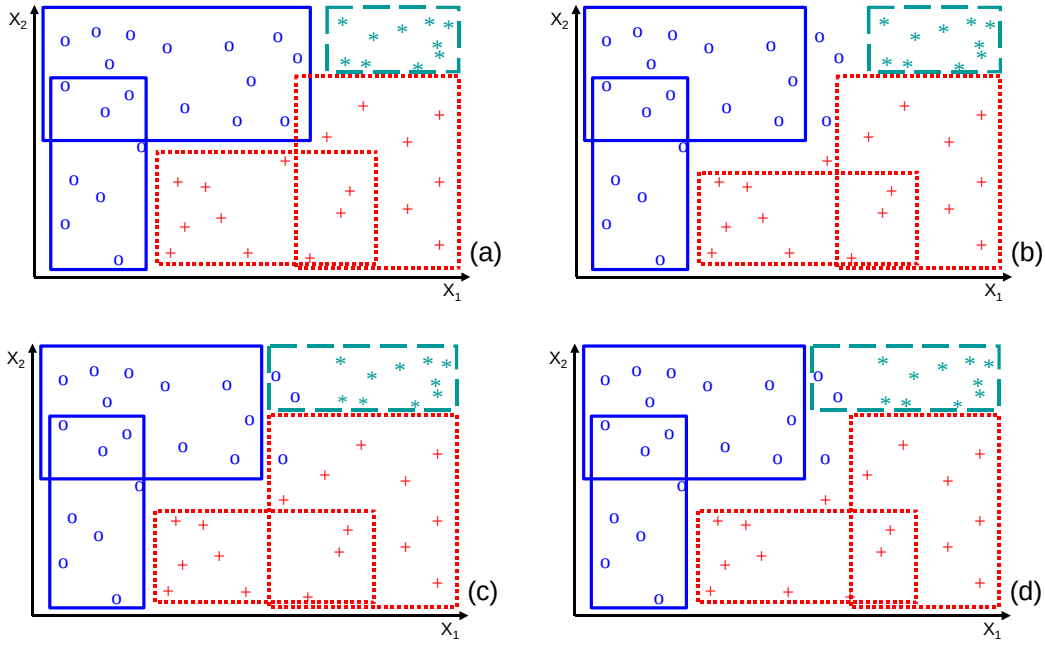


Figura 2.4: Completude e consistência de uma hipótese

Classe	predita C_1	predita C_2	$\cdots$	predita C_k
verdadeira C_1	$M(C_1, C_1)$	$M(C_1, C_2)$	$\cdots$	$M(C_1, C_k)$
verdadeira C_2	$M(C_2, C_1)$	$M(C_2, C_2)$	$\cdots$	$M(C_2, C_k)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
verdadeira C_k	$M(C_k, C_1)$	$M(C_k, C_2)$	$\cdots$	$M(C_k, C_k)$

Tabela 2.2: Matriz de confusão de um classificador

O número de acertos, para cada classe, se localiza na diagonal principal $M(C_i, C_i)$ da matriz. Os demais elementos $M(C_i, C_j)$, para $i \neq j$, representam os erros para um tipo particular de classificação. Na Tabela 2.3 é mostrada a matriz de confusão de um classificador ideal, o qual possui todos esses elementos iguais a zero uma vez que ele não comete erros.

Classe	predita C_1	predita C_2	$\cdots$	predita C_k
verdadeira C_1	$M(C_1, C_1)$	0	$\cdots$	0
verdadeira C_2	0	$M(C_2, C_2)$	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
verdadeira C_k	0	0	$\cdots$	$M(C_k, C_k)$

Tabela 2.3: Matriz de confusão de um classificador ideal

Por simplicidade, considere um problema de duas classes. Com apenas duas classes, usualmente rotuladas como "+" (positivo) e "−" (negativo), as escolhas estão estruturadas para

predizer a ocorrência ou não de um evento simples. Neste caso, os dois erros possíveis são denominados *falso positivo* (F_P) e *falso negativo* (F_N). Na Tabela 2.4 é ilustrada a matriz de confusão para o problema com duas classes, onde T_P é o número de exemplos positivos classificados corretamente e T_N é o número de exemplos negativos classificados corretamente do total de $n = (T_P + F_N + F_P + T_N)$ exemplos.

Classe	predita C_+	predita C_-	Taxa de erro da classe	Taxa de erro total
verdadeira C_+	Verdadeiros positivos T_P	Falsos negativos F_N	$\frac{F_N}{T_P+F_N}$	$\frac{F_P+F_N}{n}$
verdadeira C_-	Falsos positivos F_P	Verdadeiros negativos T_N	$\frac{F_P}{F_P+T_N}$	

Tabela 2.4: Matriz de confusão para o classificação com duas classes

Ainda considerando a Tabela 2.4, nota-se que quatro situações podem ocorrer. Na Figura 2.5 são ilustradas as situações possíveis para a classe C_+.

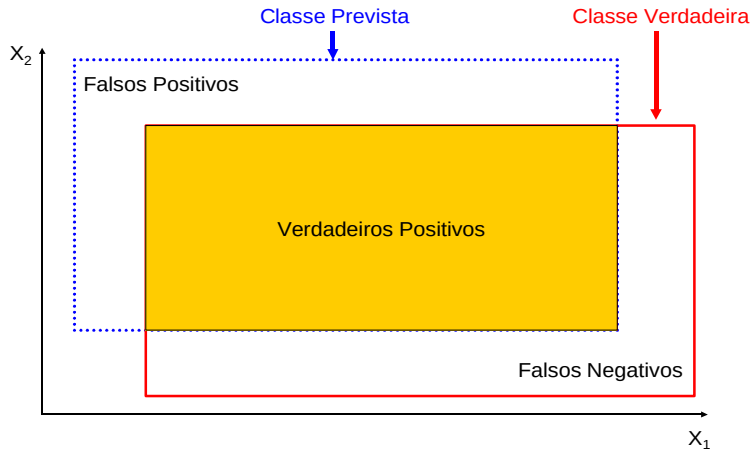


Figura 2.5: Verdadeiros positivos, falso positivos e falso negativos

1. o exemplo pertence à classe C_+ e é classificado como pertencente à classe C_+. Neste caso, o exemplo é um *verdadeiro positivo*;
2. o exemplo pertence à classe C_- e é classificado como pertencente à classe C_-. Neste caso, o exemplo é um *verdadeiro negativo*;
3. o exemplo pertence à classe C_- e é classificado como pertencente à classe C_+. Neste caso, o exemplo é um *falso positivo*;

4. o exemplo pertence à classe C_+ e é classificado como pertencente à classe C_-. Neste caso, o exemplo é um *falso negativo*.

Outras medidas podem ser derivadas a partir da matriz de confusão (mostrada na Tabela 2.4 na página precedente), tais como confiabilidade positiva prel (*positive reliability* ou C_+ *predictive value*), confiabilidade negativa nrel (*negative reliability* ou C_- predictive value), suporte sup, sensitividade sens (*sensitivity* ou *recall* ou *true* C_+ *rate*), especificidade spec (*specificity* ou *true* C_- *rate*), precisão total tacc (*total accuracy*) e cobertura cov (*coverage*) calculadas utilizando-se (2.8) à (2.14), respectivamente (Weiss & Kulikowski 1991).

$$\mathsf{prel}(h) = \frac{T_P}{T_P + F_P} \tag{2.8}$$

$$\mathsf{nrel}(h) = \frac{T_N}{T_N + F_N} \tag{2.9}$$

$$\mathsf{sup}(h) = \frac{T_P}{n} \tag{2.10}$$

$$\mathsf{sens}(h) = \frac{T_P}{T_P + F_N} \tag{2.11}$$

$$\mathsf{spec}(h) = \frac{T_N}{T_N + F_P} \tag{2.12}$$

$$\mathsf{tacc}(h) = \frac{T_P + T_N}{n} \tag{2.13}$$

$$\mathsf{cov}(h) = \frac{T_P + F_P}{n} \tag{2.14}$$

Por exemplo, alta sensitividade indica a habilidade de classificar corretamente exemplos positivos. Entretanto, a especificidade pode ser ruim se muitos exemplos positivos são classificados, incorretamente, como negativos.

Custos de Erro Medir adequadamente o desempenho de classificadores, através da taxa de erro (ou precisão) assume um papel importante em Aprendizado de Máquina, uma vez que o objetivo é construir classificadores com baixa taxa de erro em novos exemplos (Batista & Monard 1998; Batista 1997; Weiss & Kulikowski 1991). Entretanto, ainda considerando o problema anterior contendo duas classes, se o custo de ter falsos positivos e falsos negativos

não é o mesmo, então outras medidas de desempenho devem ser usadas. Uma alternativa natural, quando cada tipo de classificação incorreta possui um custo diferente ou mesmo quando existe prevalência de classes, consiste em associar um custo para cada tipo de erro.

O custo, denotado por $\mathsf{cost}(C_i, C_j)$, é um número que representa uma penalidade aplicada quando o classificador faz um erro ao rotular exemplos, cuja classe verdadeira é C_i, como pertencentes à classe C_j, onde $i, j = 1, 2, \ldots, k$ e k é o número de classes. Assim, $\mathsf{cost}(C_i, C_i) = 0$, uma vez que não constitui um erro e $\mathsf{cost}(C_i, C_j) > 0, i \neq j$. Em geral, os indutores assumem que $\mathsf{cost}(C_i, C_j) = 1, i \neq j$, caso esses valores não sejam definidos explicitamente.

No cálculo utilizando custos, os erros são convertidos em custos pela multiplicação do erro pelo custo correspondente, calculados utilizando-se (2.15), onde n representa o número de exemplos. É também possível obter os custos através da matriz de confusão utilizando-se (2.16). Assim, ao invés de projetar um algoritmo que minimize a taxa de erro, o objetivo poderia ser minimizar custos de classificação incorreta (Domingos 1999).

$$\mathsf{err\text{-}cost}(h) = \frac{1}{n} \sum_{i=1}^{n} \parallel y_i \neq h(x_i) \parallel \times \mathsf{cost}(y_i, h(x_i)) \tag{2.15}$$

$$\mathsf{err\text{-}cost}(h) = \frac{1}{n} \sum_{i=1}^{k} \sum_{j=1}^{k} M(C_i, C_j) \times \mathsf{cost}(C_i, C_j) \tag{2.16}$$

Complexo Um *complexo* é uma disjunção de conjunções de testes de atributos na forma:

$$X_i \; op \; Valor$$

onde X_i é um atributo, *op* é um operador pertencente ao conjunto $\{=, \neq, <, \leq, >, \geq\}$ e *Valor* é um valor constante válido para o atributo X_i. Para atributos contínuos, é também possível ter uma combinação linear de atributos na forma:

$$a_1 \times X_1 + a_2 \times X_2 + \ldots + a_m \times X_m \; op \; Valor$$

onde a_i é uma constante, X_i é um atributo contínuo (inteiro ou real), *op* é um operador pertencente ao conjunto $\{<, \leq, >, \geq\}$ e *Valor* é uma constante.

Regra Uma regra assume a forma

$$\textbf{if } L \textbf{ then } R$$

ou na forma simbólica

$$L \rightarrow R$$

que também pode ser representada pelas formas equivalentes $R \leftarrow L$ ou R **:-** L. Normalmente, as partes esquerda L e direita R da regra são complexos, sem atributos comuns entre eles, ou seja, $\mathsf{atributos}(L) \cap \mathsf{atributos}(R) = \emptyset$. A parte esquerda L é denominada *condição, premissa, cauda* ou *corpo* da regra, e a parte direita R é denominada *conclusão* ou *cabeça* da regra.

Regra de Classificação Uma regra de classificação assume a forma restrita de uma regra

$$\textbf{if } L \textbf{ then } \text{classe} = C_i$$

onde C_i pertence ao conjunto de k valores de classe $\{C_1, C_2, \ldots, C_k\}$.

Regra de Associação Sistemas de aprendizado por associação encontram uma regra, quando não existe uma definição explícita de classe e qualquer atributo (ou atributos) pode ser usado como parte da conclusão da regra. Por exemplo:

$$\textbf{if } X_3 = \text{s} \textbf{ and } X_5 > 2 \textbf{ then } X_1 = \text{n} \textbf{ and } X_2 < 1$$

Em geral, os sistemas convencionais de aprendizado por associação encontram regras de associação que satisfazem alguns critérios mínimos (Agrawal, Imielinski & Swami 1993; Agrawal & Srikant 1994; Klemettinen, Mannila, Ronkainen, Toivonen & Verkamo 1994; Bayardo & Agrawal 1999), tais como suporte ou confiança.

Classificador Simbólico Um *classificador simbólico* é uma hipótese cuja linguagem de descrição pode ser transformada em um conjunto de regras — por exemplo, indução de regras ou árvores de decisão. Embora os indutores possam produzir classificadores que difiram na sua forma sintática, se eles podem ser convertidos em regras, então os classificadores são simbólicos.

Cobertura Considerando uma regra $L \rightarrow R$, os exemplos que satisfazem a parte L da regra constituem o seu *conjunto de cobertura*, ou seja, os exemplos são *cobertos* pela regra ou

a regra *dispara* para esses exemplos. Exemplos que satisfazem tanto a condição L como a conclusão R são *cobertos corretamente* pela regra. Já os exemplos satisfazendo a condição L mas não a conclusão R são *cobertos incorretamente* pela regra. Por outro lado, os exemplos que não satisfazem a condição L não são cobertos pela regra. Um resumo dessas quatro situações pode ser visto na Tabela 2.5.

Exemplos satisfazendo ...	são ...
$\bar{L}$	não cobertos pela regra
L	cobertos pela regra
$L \wedge R$	cobertos corretamente pela regra
$L \wedge \bar{R}$	cobertos incorretamente pela regra

Tabela 2.5: Definições de cobertura da regra **if** L **then** R

Para exemplificar esse conceito, considere a Tabela 2.6 e a regra **if** $X_1 = \text{a}$ **e** $X_2 = \text{s}$ **then** classe $= +$. Neste caso, os exemplos $\{T_3, T_4, T_5\}$ não são cobertos pela regra; os exemplos $\{T_1, T_2\}$ são cobertos pela regra, sendo o exemplo T_1 coberto corretamente e o exemplo T_2 coberto incorretamente pela regra.

	Atributos				
Exemplo	X_1	X_2	X_3	Classe	Cobertura
T_1	a	s	2	$+$	coberto (corretamente)
T_2	a	s	1	$-$	coberto (incorretamente)
T_3	b	n	1	$+$	não coberto
T_4	b	s	2	$-$	não coberto
T_5	c	n	2	$*$	não coberto

Tabela 2.6: Conjunto de exemplos e cobertura da regra **if** $X_1 = \text{a}$ **and** $X_2 = \text{s}$ **then** classe $= +$

Matriz de Contingência Como descrito anteriormente, a matriz de confusão é aplicada ao classificador visto como uma caixa-preta, ou seja, o classificador pode ser simbólico ou não para se calcular essa matriz. Já a *matriz de contingência* é calculada para cada regra, exigindo, desta forma, que o classificador seja simbólico.

Considerando cada regra no formato $L \rightarrow R$, sua correspondente matriz de contingência é mostrada na Tabela 2.7 na página seguinte (Lavrač, Flach & Zupan 1999). Nesta tabela, L denota o conjunto de exemplos para os quais a condição da regra é verdadeira e seu complemento $\bar{L}$ denota o conjunto de exemplos para os quais a condição da regra é falsa e analogamente para R e $\bar{R}$. LR denota o conjunto de exemplos $L \cap R$ no qual ambos L e R são verdadeiros, $L\bar{R}$ denota o conjunto de exemplos $L \cap \bar{R}$ no qual L é verdadeiro e R é falso e assim por diante.

Por generalidade, denota-se a cardinalidade de um conjunto A por a, ou seja, $a = |A|$. Assim, l denota o número de exemplos no conjunto L, ou seja, $l = |L|$, r denota o número de exemplos no conjunto R, ou seja $r = |R|$, lr denota o número de exemplos no conjunto LR com $lr = |LR|$ e assim por diante. Como anteriormente, n indica o número total de exemplos.

	L	$\bar{L}$	
R	lr	$\bar{l}r$	r
$\bar{R}$	$l\bar{r}$	$\bar{l}\bar{r}$	$\bar{r}$
	l	$\bar{l}$	n

Tabela 2.7: Matriz de contingência para a regra $L \rightarrow R$

A freqüência relativa $|A|/n = a/n$ associada ao subconjunto A é denotada por $p(A)$, onde A é um subconjunto dos n exemplos. Dessa forma, a freqüência relativa é usada como uma estimativa de probabilidade. A notação $p(A|B)$ segue sua definição habitual em probabilidade, dada por (2.17), onde A e B são ambos subconjuntos dos n exemplos.

$$p(A|B) = \frac{p(AB)}{p(B)} = \frac{\frac{|AB|}{n}}{\frac{|B|}{n}} = \frac{\frac{ab}{n}}{\frac{b}{n}} = \frac{ab}{b} \tag{2.17}$$

Na Tabela 2.8 é mostrada a matriz de contingência do exemplo referente à Tabela 2.6 na página anterior.

	L		$\bar{L}$		
R	1	(T_1)	1	(T_3)	2
$\bar{R}$	1	(T_2)	2	(T_4, T_5)	3
	2		3		5

Tabela 2.8: Matriz de contingência correspondente ao exemplo da Tabela 2.6

Várias medidas podem ser usadas para avaliar o desempenho de um classificador (Horst 1999; Horst & Monard 2000), sendo a precisão a mais comum. Entretanto, com novos problemas a serem tratados, novas medidas considerando novidade, simplicidade e facilidade de compreensão humana são necessárias (Freitas 1998a; Freitas 1998b; Freitas 1999; Todorovski, Flach & Lavrač 2000).

Utilizando como base a matriz de contingência, é possível definir a maioria das medidas sobre regras, por exemplo, a confiabilidade positiva prel, confiabilidade negativa nrel, suporte

sup, sensitividade sens, especificidade spec, precisão total tacc e cobertura cov definidas pelas equações (2.18) até (2.24), respectivamente.

$$\mathsf{prel}(L \rightarrow R) = p(R|L) = \frac{lr}{l} \tag{2.18}$$

$$\mathsf{nrel}(L \rightarrow R) = p(\bar{R}|\bar{L}) = \frac{\bar{l}\bar{r}}{\bar{l}} \tag{2.19}$$

$$\mathsf{sup}(L \rightarrow R) = p(LR) = \frac{lr}{n} \tag{2.20}$$

$$\mathsf{sens}(L \rightarrow R) = p(L|R) = \frac{lr}{r} \tag{2.21}$$

$$\mathsf{spec}(L \rightarrow R) = p(\bar{L}|\bar{R}) = \frac{\bar{l}\bar{r}}{\bar{r}} \tag{2.22}$$

$$\mathsf{tacc}(L \rightarrow R) = p(LR) + p(\bar{L}\bar{R}) = \frac{lr + \bar{l}\bar{r}}{n} \tag{2.23}$$

$$\mathsf{cov}(L \rightarrow R) = p(L) = \frac{l}{n} \tag{2.24}$$

Além dessas medidas, Piatetsky-Shapiro (1991) propõe a novidade nov (*novelty*), também definida em Lavrač, Flach & Zupan (1999) juntamente com a satisfação sat (*satisfaction*), conforme (2.25) e (2.26), respectivamente.

$$\mathsf{nov}(L \rightarrow R) = p(LR) - p(L)p(R) = \frac{lr}{n} - \frac{l \cdot r}{n^2} \tag{2.25}$$

$$\mathsf{sat}(L \rightarrow R) = \frac{p(\bar{R}) - p(\bar{R}|L)}{p(\bar{R})} = 1 - \frac{n \cdot l\bar{r}}{l \cdot \bar{r}} \tag{2.26}$$

Considerando L e R, a novidade é definida verificando se LR é independente deles. Isto pode ser obtido comparando o resultado observado lr contra o valor esperado sob a consideração de independência $\frac{l \cdot r}{n}$. Quanto mais o valor observado diferir do valor esperado, maior a probabilidade de que exista uma associação verdadeira e inesperada entre L e R. Pode ser demonstrado que $-0,25 \leq \mathsf{nov} \leq 0,25$: quanto maior um valor positivo (perto de 0,25), mais forte é a associação entre L e R, enquanto que quanto menor um valor negativo (perto de $-0,25$), mais forte é a associação entre L e $\bar{R}$.

Já a satisfação é o aumento relativo na precisão entre a regra $L \rightarrow \mathsf{verdade}$ e a regra $L \rightarrow R$. Segundo Lavrač, Flach & Zupan (1999) esta medida é indicada para tarefas voltadas à descoberta de conhecimento, sendo capaz de promover um equilíbrio entre regras com diferentes condições e conclusões.

2.2 Linguagens de Representação

> It is of interest to note that while some dolphins are reported to have learned English — up to fifty words used in correct context — no human being has been reported to have learned dolphinese.
>
> —*Carl Sagan*

Ao solucionar problemas com o uso de computadores é importante definir como traduzi-los em termos computacionais. Especificamente, em AM isto significa como representar exemplos, hipóteses e conhecimento do domínio. Para descrevê-los, as seguintes *linguagens de representação* (ou *linguagens de descrição*) são usadas:

- Linguagem de Representação de Exemplos — LRE;
- Linguagem de Representação de Hipóteses — LRH;
- Linguagem de Representação de Conhecimento do Domínio — LRC.

A seguir são enumeradas algumas linguagens de representação freqüentemente utilizadas em AM em ordem crescente de complexidade e força expressiva. São fornecidas explicações intuitivas sobre essas linguagens evitando-se complexidade de notação. Uma vez que uma linguagem de representação pode descrever exemplos, hipóteses e conhecimento do domínio, por generalidade estes termos são referenciados como um *item* nas próximas seções.

2.2.1 Lógica de Ordem Zero ou Proposicional

Na lógica de ordem zero ou cálculo proposicional, o item a ser representado é descrito por conjunções, disjunções e negações de constantes booleanas que representam atributos individuais. Por exemplo:

$$\mathsf{fêmea} \wedge \mathsf{adulta} \rightarrow \mathsf{pode\_ter\_filhos}$$

Esta linguagem tem um baixo poder descritivo, não sendo capaz de descrever objetos entre os quais relações estão envolvidas.

2.2.2 Lógica de Atributos

De forma a representar itens, vários indutores proposicionais utilizam uma linguagem *baseada em atributos*. Formalmente, a lógica de atributos é equivalente ao cálculo proposicional, mas emprega uma notação mais poderosa e flexível. A melhoria é devido ao fato que os atributos são tratados como variáveis que podem assumir diversos valores. Por exemplo:

$$\mathsf{sexo{=}feminino} \wedge \mathsf{idade{=}adulta} \rightarrow \mathsf{classe{=}pode\_ter\_filhos}$$

ou equivalentemente,

$$\mathsf{sexo(feminino)} \wedge \mathsf{idade(adulta)} \rightarrow \mathsf{classe(pode\_ter\_filhos)}$$

Embora a maioria dos indutores utilize a lógica de atributos para descrever exemplos e hipóteses, sua forte limitação de expressão impede a representação de objetos estruturados, assim como as relações entre objetos ou entre seus componentes. Assim, aspectos relevantes dos exemplos que, de alguma forma poderiam caracterizar o conceito sendo aprendido, podem não ser representados.

2.2.3 Lógica de Primeira Ordem

De forma a superar as limitações de representação impostas por uma linguagem de atributos, o aprendizado utilizando representações que possuem maior poder, tais como algumas variações da lógica de primeira ordem, tem recebido mais atenção. A lógica de primeira ordem fornece uma maneira de descrever e raciocinar sobre *objetos* e *predicados* que especificam *propriedades* de objetos ou *relacionamentos* entre objetos.

Um subconjunto importante da lógica de primeira ordem é composto pelas *cláusulas de Horn*. Uma cláusula de Horn consiste em uma regra cuja cabeça contém um único predicado e um corpo com zero, um ou mais predicados. O seguinte exemplo, na sintaxe proposta por Kowalsky (1979) para a linguagem de programação lógica PROLOG descreve que uma pessoa X é irmão da pessoa Y se X é homem e ambos X e Y possuem o mesmo pai Z.

$$\mathsf{irmão}(X,Y) \ \textbf{:-}\ \mathsf{homem}(X),\ \mathsf{pai}(Z,X),\ \mathsf{pai}(Z,Y).$$

A parte à esquerda do símbolo **:-** é a cabeça e a parte à direita do símbolo é o corpo (ou cauda) da cláusula. O símbolo **:-** é equivalente à implicação lógica $\leftarrow$ e é denominado *neck*[2]. As vírgulas

[2] q **:-** $p \equiv q \leftarrow p \equiv p \rightarrow q$

separando cada predicado significam conjunções lógicas. Além disso, todas as variáveis são sempre universalmente quantificadas. As variáveis entre parênteses são chamadas de *argumentos*.

Nota-se que se todos os predicados não possuem argumentos, a linguagem se reduz à lógica de ordem zero e se todos os predicados possuem um único argumento constante (sem variáveis envolvidas), a linguagem se reduz à lógica de atributos.

2.2.4 Lógica de Segunda Ordem

A lógica de segunda ordem é uma extensão da lógica de primeira ordem, permitindo que os predicados possam ser considerados como variáveis. Por exemplo, suponha o esquema:

$$P_1(X, Y) \text{ :- } P_2(X),\ P_3(X, Z),\ P_4(Y, Z).$$

onde P_1, P_2, P_3, P_4 são variáveis que representam predicados e X, Y, Z são variáveis que representam objetos. Uma possível instanciação poderia ser

$$\mathsf{irmão}(X,Y) \text{ :- } \mathsf{homem}(X),\ \mathsf{pai}(Z,X),\ \mathsf{pai}(Z,Y).$$

Conseqüentemente, o esquema permanece intacto e apenas os nomes dos predicados podem variar. É conveniente salientar que esta linguagem de representação é tão rica e flexível que seu uso é, em muitos casos, computacionalmente inviável. Algumas vezes, na prática, se introduz restrições, tais como limitar o número de predicados na cláusula, excluir definições recursivas ou mesmo limitar o número de argumentos do predicado (Morik, Wrobel, Jörg-Uwe & Emde 1993).

2.2.5 Funções Matemáticas

Funções matemáticas podem ser usadas para descrever hipóteses, por exemplo para redes neurais. Neste caso, o espaço de representação é dividido em regiões complexas através da combinação de várias funções matemáticas.

Na Tabela 2.9 na próxima página são consideradas as linguagens de representação de alguns indutores, freqüentemente encontrados na literatura, tais como CART (Breiman, Friedman, Olshen & Stone 1984), C4.5 (Quinlan 1993), RIPPER (Cohen 1995), CN2 (Clark & Niblett 1987; Clark & Niblett 1989; Clark & Boswell 1991), FOIL (Quinlan 1990), PROGOL (Muggleton & Firth 1999) e rede neural. É interessante notar que apenas RIPPER, FOIL e PROGOL são capazes de processar conhecimento do domínio.

Indutor	LRE	LRH	LRC
C4.5	atributos	atributos	
CART	atributos	atributos	
CN2	atributos	atributos	
RIPPER	atributos	atributos	atributos
FOIL	atributos	primeira ordem	primeira ordem
PROGOL	atributos	primeira ordem	primeira ordem
rede neural	atributos	funções matemáticas	

Tabela 2.9: Linguagens de representação de alguns indutores

2.3 Considerações Finais

> Iron rusts from disuse; stagnant water loses its purity and in cold weather becomes frozen; even so does inaction sap the vigor of the mind.
>
> —*Leonardo da Vinci, The Notebooks*

Neste capítulo foram apresentados conceitos e definições de alguns termos amplamente usados neste trabalho, bem como em Aprendizado de Máquina, além de uma descrição sobre as principais linguagens de representação.

A investigação de estruturas diferentes, que podem ser apropriadas para diferentes contextos, bem como o entendimento do seu poder e limitação são necessários para o uso com êxito de Aprendizado de Máquina. Quanto maior a compreensão sobre as estruturas fundamentais usadas por classificadores, mais adequadamente pode-se aplicar ou alterá-las com base no conhecimento do domínio.

Além da compreensão dos algoritmos de AM, é igualmente importante poder avaliar seu desempenho. O próximo capítulo introduz a metodologia de avaliação usada para comparar algoritmos nos experimentos realizados, descritos nos Capítulos 5, 6 e 8.

Capítulo 3

Metodologia de Avaliação de Algoritmos

> If you want to show that one algorithm is *always* more accurate, then forget it: this simply cannot be proven.
>
> —*Steven L. Salzberg (Salzberg 1995b)*

Aprendizado de Máquina é uma ferramenta poderosa, mas não existe um único algoritmo que apresente o melhor desempenho para todos os problemas (Dietterich 1997a; Kohavi, Sommerfield & Dougherty 1996; Schaffer 1994). Dessa forma, é importante compreender o poder e a limitação dos diferentes algoritmos. Para tanto, neste capítulo é descrita a metodologia de avaliação, por nós adotada, para comparar dois algoritmos, a qual se baseia na idéia de amostragem (*resampling*), explicada na próxima seção. Além disso, são apresentados os conjuntos de exemplos utilizados nos experimentos reportados em capítulos subseqüentes, incluindo uma breve descrição e informações básicas sobre eles, tais como número de exemplos e atributos. Por último, os indutores utilizados nos experimentos, além de algumas ferramentas, são também descritos.

3.1 Métodos de Amostragem

> As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.
>
> —*Albert Einstein*

Dados um conjunto de exemplos de tamanho finito e um indutor, é importante *estimar* o de-

sempenho futuro do classificador induzido utilizando o conjunto de exemplos. Todos os métodos não paramétricos descritos a seguir, exceto pelo método de resubstituição, estão baseados na idéia de *amostragem*, ilustrada na Figura 3.1. O mundo real apresenta uma distribuição de exemplos D, a qual é desconhecida. Ao extrair exemplos do mundo real, formando assim um conjunto de exemplos, obtém-se uma distribuição de exemplos D', a qual é supostamente similar à distribuição D. De modo a estimar uma medida, geralmente a precisão ou o erro, de indutores treinados com base na distribuição D', extraem-se amostras a partir de D', treina-se um indutor com essas amostras e testa-se seu desempenho em exemplos de D' (normalmente com exemplos fora da amostra utilizada para treinamento). Desta forma, simula-se o processo de amostragem que ocorre no mundo real, assumindo que D' representa o mundo real.

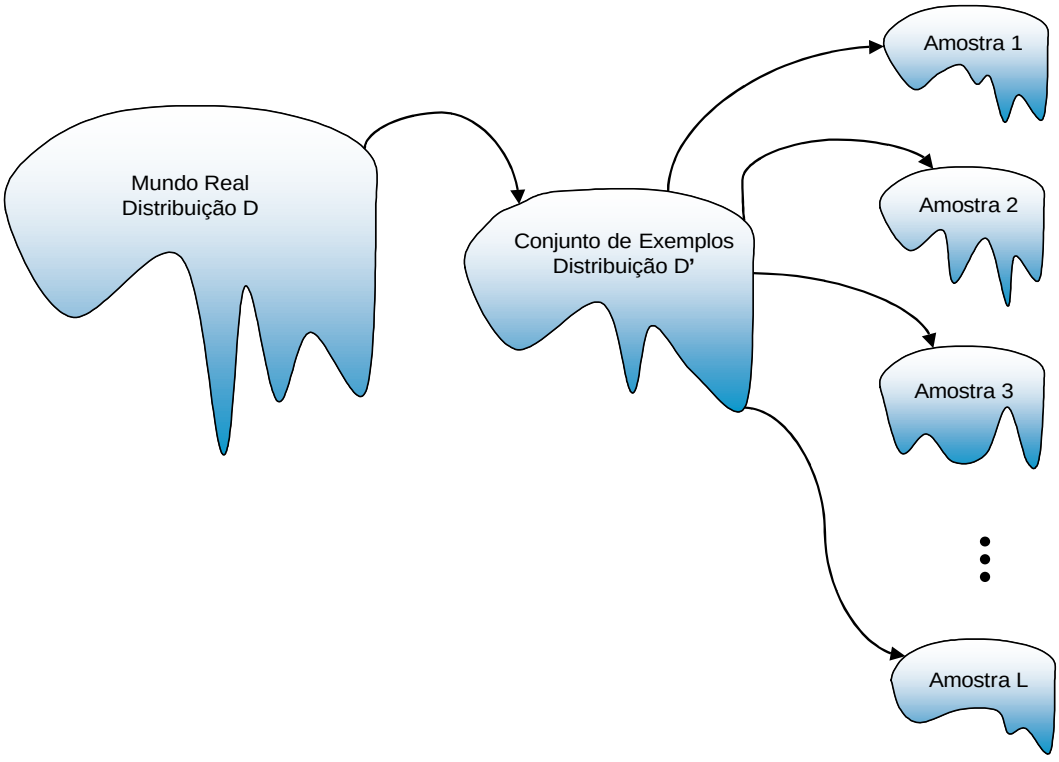


Figura 3.1: Técnicas de estimativas baseadas na idéia de amostragem

É importante, ao estimar uma medida verdadeira, que a amostra seja *aleatória*, isto é, os exemplos não devem ser pré-selecionados. Para problemas reais, normalmente é tomada uma amostra de tamanho n e o objetivo consiste em estimar uma medida para aquela população em particular (não para todas as populações). Existem vários métodos para estimar uma medida verdadeira que são descritos a seguir e resumidos na Tabela 3.1 na próxima página.

	holdout	aleatória	*leave-one-out*	*r-fold cv*	*r-fold strat cv*	*bootstrap*
Treinamento	pn	t	$n-1$	$n(r-1)/r$	$n(r-1)/r$	n
Teste	$(1-p)n$	$n-t$	1	n/r	n/r	$n-t$
Iterações	1	$L \ll n$	n	r	r	$\simeq 200$
Reposição	não	não	não	não	não	sim
Prevalência de Classe	não	não	não	não	sim	sim/não

Tabela 3.1: Alguns parâmetros típicos de estimadores, onde n representa o número de exemplos, r o número de *folds* (partições), p um número tal que $0 < p < 1$, t um número tal que $0 < t < n$ e L o número de hipóteses induzidas

3.1.1 Resubstituição

Como mencionado anteriormente, um conjunto de exemplos geralmente é dividido em dois subconjuntos distintos: o conjunto de treinamento e o conjunto de teste. Por exemplo, no cálculo do erro realizado utilizando-se o conjunto de teste, ele pode ser considerado o erro verdadeiro, uma vez que se aproxima ao erro da população se o conjunto de teste é grande o suficiente. Embora o erro seja a medida mais comumente utilizada, quaisquer outras medidas podem ser realizadas, tais como tempo de aprendizado, quantidade de memória utilizada, tamanho ou complexidade da hipótese induzida. Quando uma amostra de teste atinge 1.000 exemplos, as medidas estimadas são extremamente precisas e com 5.000 exemplos, as medidas estimadas são quase idênticas às medidas verdadeiras da população (Weiss & Kulikowski 1991).

O método de resubstituição consiste em construir o classificador e testar seu desempenho no mesmo conjunto de exemplos, ou seja, o conjunto de teste é idêntico ao conjunto de treinamento. Como já mencionado, este estimador fornece uma *medida aparente*, possuindo uma estimativa altamente otimista da precisão, devido ao fato de que o processo de classificação tenta maximizá-la. Para muitos algoritmos de indução que classificam corretamente todos os exemplos, tais como 1-NN ou árvores de decisão sem poda, esta estimativa é muito otimista: se não houver exemplos conflitantes, a estimativa de precisão atinge 100%. Assim sendo, o desempenho calculado com este método possui um *bias* otimista: o bom desempenho no conjunto de treinamento em geral não se estende a conjuntos independentes de teste.

Quando o *bias* do estimador de resubstituição foi descoberto, diversos métodos de *cross-validation* foram propostos, os quais são descritos a seguir. Todos eles estão baseados no mesmo princípio: não deve haver exemplos em comum entre o conjunto de treinamento (ou aprendizado) e o conjunto de teste.

3.1.2 Holdout

O estimador *holdout* divide os exemplos em uma porcentagem fixa de exemplos p para treinamento e $(1-p)$ para teste, considerando normalmente $p > 1/2$. Valores típicos são $p = 2/3$ e $(1-p) = 1/3$, embora não existam fundamentos teóricos sobre estes valores.

De forma a tornar o resultado menos dependente da forma de divisão dos exemplos, pode-se calcular a média de vários resultados de *holdout* através da construção de várias partições obtendo-se, assim, uma estimativa média do *holdout*. Uma vez que uma hipótese construída utilizando todos os exemplos, em média, apresenta desempenho melhor que uma hipótese construída utilizando apenas uma parte dos exemplos, este método tem a tendência de super estimar o erro verdadeiro.

3.1.3 Amostragem Aleatória

Na amostragem aleatória, L hipóteses, $L \ll n$, são induzidas a partir de cada conjunto de treinamento e o erro final é calculando como sendo a média dos erros de todas as hipóteses induzidas de conjuntos de teste independentes e extraídos aleatoriamente. Amostragem aleatória pode produzir melhores estimativas de erro que o estimador *holdout*, explicado na Seção 3.1.6.

3.1.4 Cross-Validation

Este estimador é um meio termo entre os estimadores *holdout* e *leave-one-out* (descrito na Seção 3.1.6). Em *r-fold cross-validation* — CV — os exemplos são aleatoriamente divididos em r partições mutuamente exclusivas (*folds*) de tamanho aproximadamente igual a n/r exemplos. Os exemplos nos $(r-1)$ *folds* são usados para treinamento e a hipótese induzida é testada no *fold* remanescente. Este processo é repetido r vezes, cada vez considerando um *fold* diferente para teste. O erro na *cross-validation* é a média dos erros calculados em cada um dos r *folds*.

Este procedimento de rotação reduz tanto o *bias* inerente ao método de *holdout* quanto o custo computacional do método *leave-one-out*. Entretanto, deve-se observar, por exemplo, que em 10-*fold cross-validation*, cada par de conjuntos de treinamento compartilha 80% de exemplos. É fácil generalizar que a porcentagem de exemplos compartilhados na *r-fold cross-validation* é dada por $(1-2/r)$ para $r \geq 2$ *folds*, como é mostrado pelo gráfico na Figura 3.2 na página oposta. À medida que o número de *folds* aumenta, esta sobreposição pode evitar que os testes estatísticos obtenham uma boa estimativa da quantidade de variação que seria observada se cada conjunto de treinamento fosse independente dos demais.

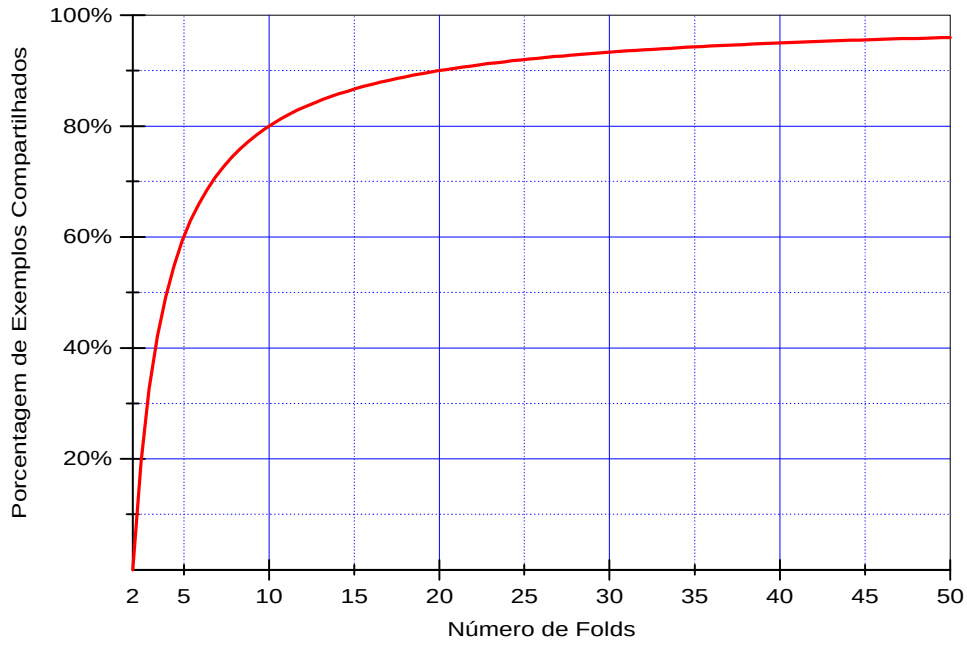


Figura 3.2: Número de *folds versus* porcentagem de exemplos compartilhados em *cross-validation*

3.1.5 Stratified Cross-Validation

O estimador *stratified cross-validation* é similar à *cross-validation*, mas ao gerar os *folds* mutuamente exclusivos, a distribuição de classe — a proporção de exemplos em cada uma das classes — é considerada durante a amostragem. Isto significa, por exemplo, que se o conjunto original de exemplos possui duas classes com distribuição de 20% e 80%, então cada *fold* também terá esta mesma proporção de classes.

3.1.6 Leave-one-out

O estimador *leave-one-out* é um caso especial de *cross-validation*. É computacionalmente dispendioso e freqüentemente é usado em amostras pequenas. Para uma amostra de tamanho n uma hipótese é induzida utilizando $(n-1)$ exemplos; a hipótese é então testada no único exemplo remanescente. Este processo é repetido n vezes, cada vez induzindo uma hipótese deixando de considerar um único exemplo. O erro é a soma dos erros em cada teste dividido por n.

3.1.7 Bootstrap

No estimador *bootstrap*, a idéia básica consiste em repetir o processo de classificação um grande número de vezes (Efron & Tibshirani 1993). Estima-se então valores, tais como o erro ou *bias*, a partir dos experimentos replicados, cada experimento sendo conduzido com base em um novo conjunto de treinamento obtido por amostragem com reposição do conjunto original de exemplos.

Há muitos estimadores *bootstrap*. O estimador mais comum é o *bootstrap e0*. Um conjunto de treinamento *bootstrap* consiste de n exemplos (assim como o conjunto original de exemplos) amostrados *com reposição* a partir do conjunto original de exemplos. Isto significa que alguns exemplos T_i podem não aparecer no conjunto de treinamento *bootstrap* e alguns T_i podem aparecer mais de uma vez. Os exemplos remanescentes (aqueles que não aparecem no conjunto de treinamento *bootstrap*) são usados como o conjunto de teste. Neste trabalho o termo '*bootstrap*' sempre se refere ao estimador *bootstrap e0*.

Para uma dada amostra *bootstrap*, um exemplo de treinamento tem probabilidade $1 - (1 - 1/n)^n$ de ser selecionado pelo menos uma vez em cada uma das n vezes nas quais os exemplos são aleatoriamente selecionados a partir do conjunto original de exemplos. Para n grande, isto é aproximadamente $1 - 1/e = 0,632$.

Portanto, para esta técnica, a fração média de exemplos não repetidos no conjunto de treinamento é $63,2\%$ e a fração média de exemplos no conjunto de teste é de $36,8\%$. O erro estimado é a média dos erros sobre o número de iterações. Cerca de 200 iterações para estimativas *bootstrap* são consideradas necessárias para obter-se uma boa medida, no caso do conjunto original de exemplos ser pequeno. A estimativa do erro para este método é estatisticamente equivalente ao estimador *leave-one-out*.

3.2 Desempenho de Algoritmos

> Experimental machine learning research needs to scrutinize its approach to experimental design. If not done carefully, comparative studies of classification algorithms can easily result in statistically invalid conclusions.
>
> —*Steven L. Salzberg (Salzberg 1995b)*

Esta seção descreve uma metodologia para a avaliação de algoritmos que é comumente utilizada em AM e que foi utilizada nos experimentos, por nós realizados, reportados nos Capítulos 5, 6 e 8. Existem muitos outros testes estatísticos para medir se a diferença entre quaisquer dois

algoritmos é significativa ou não, além do descrito aqui (Freedman, Pisani & Purves 1998). Uma boa revisão sobre a comparação de algoritmos pode ser encontrada em (Salzberg 1995b; Dietterich 1997c).

3.2.1 Calculando Média e Desvio Padrão Utilizando Amostragem

Antes de comparar dois algoritmos, algumas definições adicionais são necessárias. Para tanto, assume-se o emprego de *cross-validation*, uma vez que é um método comumente utilizado pela comunidade de AM. Entretanto, qualquer outro método de amostragem (exceto resubstituição) pode ser utilizada no lugar de *cross-validation* para calcular a média e desvio padrão de um algoritmo.

Dado um algoritmo A e um conjunto de exemplos T, assume-se que T seja dividido em r partições. Para cada partição i, é induzida a hipótese h_i e o erro denotado por $\mathsf{err}(h_i)$, $i = 1, 2, \ldots, r$, é calculado. A seguir, a média, variância e desvio padrão para todas as partições são calculados utilizando-se (3.1), (3.2) e (3.3), respectivamente.

$$\mathsf{mean}(A) = \frac{1}{r}\sum_{i=1}^{r}\mathsf{err}(h_i) \tag{3.1}$$

$$\mathsf{var}(A) = \frac{1}{r}\left[\frac{1}{r-1}\sum_{i=1}^{r}\left(\mathsf{err}(h_i) - \mathsf{mean}(A)\right)^2\right] \tag{3.2}$$

$$\mathsf{sd}(A) = \sqrt{\mathsf{var}(A)} \tag{3.3}$$

Assumindo $\mathsf{mean}(A)$ como tendo uma distribuição normal (é importante lembrar que cada $\mathsf{err}(h_i)$ é, por si só, uma média), observa-se que o termo $1/(r-1)$ em (3.2) tem origem na definição do estimador não viciado para variância, enquanto o termo $1/r$ é originário do Teorema do Limite Central para estimar a variância de médias (Moses 1986, Capítulo 4).

É possível denotar $\mathsf{mean}(A)$ como $\mathsf{mean}(A, T)$, quando a intenção é tornar evidente o fato que o erro médio do algoritmo A foi calculado sobre o conjunto de exemplos T. Alternativamente, $\mathsf{mean}(A)$ pode ser denotado por $\mathsf{mean}(T)$, quando deseja-se evidenciar o fato que o erro médio foi calculado utilizando o conjunto de exemplos T, assumindo o algoritmo A fixo para um dado experimento. Analogamente, essa notação se estende para $\mathsf{var}(A)$, $\mathsf{sd}(A)$ ou outros valores que possam ser derivados a partir destes (por exemplo, (3.6) definida na página 51).

Para exemplificar o cálculo da média e desvio padrão de um algoritmo A utilizando um

conjunto de exemplos T, considere 10-*fold cross-validation*, isto é, $r = 10$, para um algoritmo A com os seguintes erros em cada *fold* $(5,5,\ 11,4,\ 12,7,\ 5,2,\ 5,9,\ 11,3,\ 10,9,\ 11,2,\ 4,9,\ 11,0)$. Então, os valores de (3.1) e (3.3) são:

$$\mathsf{mean}(A) = \frac{90,00}{10} = 9,00$$

$$\mathsf{sd}(A) = \sqrt{\frac{1}{10 \times (9)} 90,30} = 1,00$$

Em geral, o erro é representado por sua média seguida pelo símbolo "±" e seu desvio padrão; no exemplo, o erro é $9,00 \pm 1,00$.

É importante notar que a maioria dos programas de AM que realizam *cross-validation* efetuam esses cálculos, por exemplo, a biblioteca $\mathcal{MLC}$++ (Kohavi, Sommerfield & Dougherty 1994; Kohavi, Sommerfield & Dougherty 1996; Felix, Rezende, Doi, de Paula & Romanato 1998) ou a ferramenta MineSet™ (Rathjens 1996).

3.2.2 Comparando Algoritmos

Ao tentar comparar dois algoritmos observando apenas valores, por exemplo, a taxa de erro em problemas de classificação ou o erro em problemas de regressão, não é fácil perceber se um algoritmo é melhor do que o outro. Em várias situações, para comparar o erro (média e desvio padrão) obtido, *r-fold stratified cross-validation* é usualmente utilizada (para manter a distribuição de classes). De fato, a maioria dos trabalhos na área reportam erros utilizando 10-*fold cross-validation* ou *stratified cross-validation*.

Ao comparar dois indutores no mesmo domínio T, o desvio padrão pode ser visto como uma imagem da robustez do algoritmo: se os erros, calculados sobre diferentes conjuntos de teste, provenientes de hipóteses induzidas utilizando diferentes conjuntos de treinamento, são muito diferentes de um experimento para outro, então o indutor não é robusto a mudanças no conjunto de treinamento, proveniente de uma mesma distribuição.

Por outro lado, suponha por exemplo, que deseja-se comparar dois algoritmos com taxas de erro iguais a $9,00 \pm 1,00$ e $7,50 \pm 0,80$. Para decidir qual deles é melhor que o outro (com grau de confiança de 95%) basta assumir o caso geral para determinar se a diferença entre dois algoritmos — A_S e A_P — é significante ou não, assumindo uma distribuição normal (Weiss & Indurkhya 1998). Em geral, a comparação é feita de forma que A_P é o algoritmo proposto e A_S o algoritmo

padrão. Para isso, a média e desvio padrão combinados são calculados de acordo com (3.4) e (3.5), respectivamente. A diferença absoluta, em desvios padrões, é calculada utilizando (3.6).

$$\mathsf{mean}(A_S - A_P) = \mathsf{mean}(A_S) - \mathsf{mean}(A_P) \tag{3.4}$$

$$\mathsf{sd}(A_S - A_P) = \sqrt{\frac{\mathsf{sd}(A_S)^2 + \mathsf{sd}(A_P)^2}{2}} \tag{3.5}$$

$$\mathsf{ad}(A_S - A_P) = \frac{\mathsf{mean}(A_S - A_P)}{\mathsf{sd}(A_S - A_P)} \tag{3.6}$$

Se $\mathsf{ad}(A_S - A_P) > 0$ então A_P supera A_S. Porém, se $\mathsf{ad}(A_S - A_P) \geq 2$ desvios padrões então A_P supera A_S com grau de confiança de 95%. Por outro lado, se $\mathsf{ad}(A_S - A_P) \leq 0$ então A_S supera A_P e se $\mathsf{ad}(A_S - A_P) \leq -2$ então A_S supera A_P com grau de confiança de 95%.

Retornando ao exemplo descrito anteriormente, assuma que $A_S = 9,00 \pm 1,00$, para o algoritmo padrão, e $A_P = 7,50 \pm 0,80$, para o algoritmo proposto. Apenas observando os valores, há uma tendência em se achar que A_P é melhor que A_S, entretanto utilizando (3.4), (3.5) e (3.6), obtém-se:

$$\begin{aligned}
\mathsf{mean}(A_S - A_P) &= 9,00 - 7,50 &&= 1,50 \\
\mathsf{sd}(A_S - A_P) &= \sqrt{\frac{1,00^2 + 0,80^2}{2}} &&= 0,91 \\
\mathsf{ad}(A_S - A_P) &= \frac{1,50}{0,91} &&= 1,65
\end{aligned}$$

Conseqüentemente, como $\mathsf{ad}(A_S - A_P) < 2$, A_P supera A_S, porém A_P **não** supera A_S significativamente (com grau de confiança de 95%).

3.3 Conjuntos de Exemplos

> The important thing is never to stop questioning. Curiosity has its own reason for existing.
>
> —*Albert Einstein*

Os experimentos, reportados em capítulos subseqüentes, foram conduzidos em conjuntos de exemplos provenientes de diversos domínios do mundo real. A maioria dos conjuntos de exemplos

é do repositório UCI Irvine (Blake & Merz 1998). Os conjuntos de exemplos dna e genetics são do projeto StatLog (Michie, Spiegelhalter & Taylor 1994). O conjunto de exemplos smoke foi obtido da URL `http://lib.stat.cmu.edu/datasets/csb/`. Além de serem de diferentes domínios, os conjuntos de exemplos possuem dimensões diferentes: o número de exemplos e o número de atributos variam sobre um grande intervalo. Além disso, os conjuntos de exemplos possuem atributos nominais, contínuos ou combinação de ambos, além de valores desconhecidos. A seguir é fornecida uma descrição, quando disponível, sobre conjuntos de exemplos utilizados neste trabalho, bem como um resumo das suas características.

anneal Este conjunto de exemplos foi doado por David Sterling e Wray Buntine.

breast-cancer Este conjunto de exemplos foi obtido dos hospitais da Universidade de Wisconsin, Madison pelo Dr. William H. Wolberg. O problema consiste em predizer se uma amostra de tecido retirado da mama de um paciente é um câncer benigno ou maligno. A cada amostra foi atribuído um vetor 9-dimensional. Cada componente encontra-se no intervalo de 1 a 10, com 1 significando estado normal e 10 o estado mais anormal. O grau de quão maligno o tecido foi determinado através da retirada de uma amostra de tecido da mama do paciente e realizando uma biópsia nele. Um diagnóstico benigno é confirmado por biópsia ou por exames periódicos, dependendo da escolha do paciente.

bupa Este conjunto de exemplos foi uma contribuição de R. S. Forsyth ao repositório UCI. O problema consiste em predizer se um paciente do sexo masculino tem desordens hepáticas com base em vários testes de sangue e na quantidade de álcool consumida.

cmc Este conjunto de exemplos consiste em um subconjunto do estudo de eficácia contraceptiva da Indonésia, realizado em 1987. As amostras são de mulheres casadas que não estavam grávidas ou não sabiam se estavam grávidas na ocasião da entrevista. O problema consiste em predizer a escolha do método contraceptivo de uma mulher (nenhum, método de curta duração, método de longa duração) com base nas características demográficas e sócio-econômicas.

crx Este conjunto de exemplos está relacionado com aplicações de cartões de crédito. Todos os nomes de atributos e valores foram alterados para símbolos sem significado para proteger a confidencialidade dos dados.

dna O domínio deste conjunto de exemplos (Michie, Spiegelhalter & Taylor 1994) é referente à área de biologia molecular. Juntas de união são pontos na seqüência de DNA nas

quais o DNA *supérfulo* é removido durante a criação da proteína. O problema consiste em reconhecer fronteiras exon/intron, denotadas por sítios **EI**; fronteiras intron/exon, denotadas por sítios **IE** ou nenhuma delas. As fronteiras **IE** são denominadas de *receptoras* e as fronteiras **EI** de *doadoras*. Os exemplos foram retirados do GenBanck 64.1 (`genbank.bio.net`). Os atributos fornecem uma janela de 60 nucleotídeos, cada um representado por 3 atributos binários que representam os valores A, C, G, ou T, fornecendo, dessa forma, 180 atributos binários. A classificação é o ponto central da janela, possuindo 30 nucleotídeos de cada lado da junta.

genetics Este conjunto de exemplos é uma codificação alternativa para o conjunto de exemplos dna. Os atributos fornecem uma janela de 60 nucleotídeos, cada um representado pelos valores A, C, G, ou T.

hepatitis Este conjunto de exemplos tem como objetivo predizer a expectativa de vida de pacientes com hepatitis.

hungaria Este conjunto de exemplos tem como objetivo o diagnóstico de doenças cardíacas.

letter O objetivo deste conjunto de exemplos é de identificar um grande número de pontos preto e branco formando uma grade retangular como uma das 26 letras do alfabeto. As imagens das letras são baseadas em 20 diferentes fontes e cada letra dentro dessas 20 fontes foram aleatoriamente distorcidas para produzir um arquivo de 20000 estímulos. Cada estímulo foi convertido em 16 atributos numéricos primitivos, tais como momentos estatísticos ou número de arestas, normalizados para um intervalo de valores inteiros entre 0 e 15.

pima Este conjunto de exemplos foi doado por V. Sigillito do Laboratório de Física Aplicada, Univesidade Johns Hopkins. É um subconjunto de uma base de dados maior que é mantida pelo Instituto Nacional de Diabetes e Doenças Digestivas e Renais nos Estados Unidos.

Todas as pacientes são mulheres com pelo menos 21 anos de idade de descendência indígena pima vivendo próximas a Phoenix, Arizona, EUA. O problema consiste em predizer se uma paciente apresentará um resultado positivo para diabetes de acordo com os critérios da Organização Mundial da Saúde a partir de medidas fisiológicas e resultados de testes médicos.

smoke Este conjunto de exemplos tem como objetivo predizer atitudes baseadas nas restrições de fumar no local de trabalho (proibido, restrito ou irrestrito) com base em co-variantes legais, relacionadas ao fumo e sócio-demográficas (Bull 1994).

sonar Este conjunto de exemplos foi usado por Gorman e Sejnowski no estudo de classificação de sinais de sonar utilizando uma rede neural (Gorman & Sejnowski 1988). O problema consiste em discriminar entre sinais de sonar que representam um cilindro de metal daqueles que representam uma rocha ligeiramente cilíndrica.

O conjunto de exemplos contém 111 exemplos obtidos por varredura de sonar de um cilindro de metal em vários ângulos e sob várias condições; contém também 97 exemplos obtidos por varredura de rochas sob as mesmas condições. Cada exemplo é um conjunto de 60 números reais entre 0 e 1. Cada número representa a energia em uma banda de freqüência particular integrada sobre um certo período de tempo. A classe associada com cada exemplo contém a letra **R** se o objeto é uma rocha e **M** se ele é uma mina (cilindro de metal).

Na Tabela 3.2 são resumidas algumas características dos conjuntos de exemplos descritos. Para cada conjunto de exemplos, são mostrados o número de exemplos (#Exemplos), duplicados (que aparecem mais de uma vez) ou conflitantes (mesmo atributo-valor mas com classes diferentes), número de atributos (#Atributos) contínuos ou nominais, número de classes (#Classes), o erro majoritário e se o conjunto de exemplos possui ao menos um valor desconhecido. Os conjuntos de exemplos são apresentados em ordem crescente do número de atributos.

Na Figura 3.3 na página oposta são mostradas duas dimensões dos conjunto de exemplos, ou seja, o número de atributos e número de exemplos. Devido ao grande intervalo de variação, o número de exemplos é representado como $\log_{10}$(#Exemplos).

Conjunto de Exemplos	#Exemplos	Duplicados ou conflitantes	#Atributos (cont.,nom.)	#Classes	Erro Majoritário	Valor Desconhecido
bupa	345	4	6 (6,0)	2	42,03%	N
pima	769	1	8 (8,0)	2	34,98%	N
cmc	1.473	115	9 (2, 7)	3	57,30%	N
breast-cancer	699	8	10 (10,0)	2	34,48%	S
hungaria	294	1	13 (13,0)	2	36,05%	S
smoke	2.855	29	13 (2, 11)	3	30,47%	N
crx	690	0	15 (6,9)	2	44,49%	S
letter	15.000	846	16 (16,0)	26	95,92%	S
hepatitis	155	0	19 (6,13)	2	20,65%	S
anneal	898	12	38 (6,32)	5	23,83%	S
sonar	208	0	60 (60,0)	2	46,63%	N
genetics	3.190	185	60 (0,60)	3	48,12%	N
dna	3.186	185	180 (0,180)	3	48,09%	N

Tabela 3.2: Características dos conjuntos de exemplos

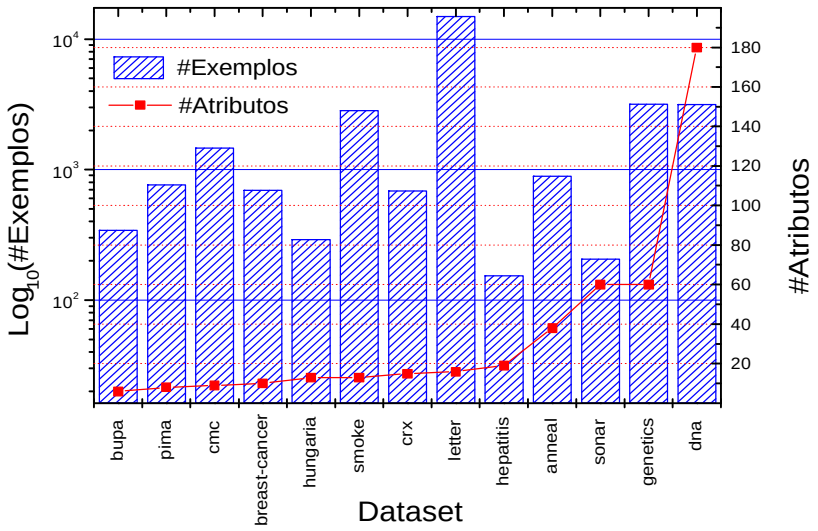


Figura 3.3: Número de exemplos (escala $\log_{10}$) e número de atributos dos conjuntos de exemplos

3.4 Indutores

> Experimentation with variants of current learning algorithms, specially when it is directed towards probing their weakness, should lead to a better understanding of how methods relate to families of tasks. This is essential if we are to design a new generation of more robust and effective learning systems.
>
> —*John Ross Quinlan, (Quinlan 1993)*

Nos experimentos efetuados neste trabalho, foram utilizados vários indutores que podem ser encontrados na biblioteca $\mathcal{MLC}$++, descrita na próxima seção (Kohavi, Sommerfield & Dougherty 1996). Dentre os indutores, tem-se C4.5, C4.5RULES, CN2, IB (*Instance-Based*) e NB (*naïve* Bayes), entre outros.

Esses indutores são bem difundidos na comunidade de AM, e mais recentemente na área de Mineração de Dados (descrita no próximo capítulo), e representam três diferentes abordagens de aprendizado. NB é um indutor estatístico muito simples, IB é um indutor preguiçoso (*lazy*) e C4.5, C4.5RULES e CN2 são indutores ávidos (*eager*).

Algoritmos puramente preguiçosos armazenam todos os dados de treinamento e respondem a uma requisição de informação através da combinação dos seus dados armazenados, descartando a resposta obtida e qualquer resultado intermediário. Em contraste, algoritmos ávidos compilam os dados de treinamento de maneira 'gulosa' em uma descrição de conceito intencional, tais como

um conjunto de regras ou uma árvore de decisão, descartando os dados após este processo (Aha 1997).

Esta distinção entre algoritmos preguiçosos/ávidos exibe alguns aspectos interessantes. Por exemplo, ainda que algoritmos preguiçosos apresentem, durante a fase de aprendizado, menor custo computacional que os algoritmos ávidos, eles tipicamente têm requisitos de armazenamento muito grande e freqüentemente possuem um grande custo computacional quando respondem a alguma requisição de classificação de um novo exemplo.

Segundo Aha (1998), métodos ávidos assumem que seu *bias* é apropriado para a tarefa a ser realizada. Quando essa suposição está correta têm-se vantagens, tais como o aumento da velocidade do tempo de resposta à consulta. Porém, existe o risco dessa suposição estar errada e uma informação importante, para a geração de respostas mais precisas, ser perdida durante a abstração do conceito.

Uma outra vantagem computacional, que pode ser citada para métodos preguiçosos, é o seu pequeno custo de treinamento, por exemplo, armazenar um exemplo e atualizar alguns índices. Entretanto, isso é freqüentemente contra-balanceado com custos mais altos para classificar um novo exemplo, exceto se uma estrutura eficiente de indexação for implementada.

Dessa forma, pode-se notar que a principal diferença entre os métodos preguiçosos e ávidos é que o primeiro permite a opção da seleção de uma hipótese diferente a cada novo exemplo a ser classificado, ou permite a seleção de uma aproximação local para a classe de cada novo exemplo a ser classificado. O segundo está comprometido com uma única hipótese que cobre todo o espaço de exemplos (Mitchell 1998).

A seguir são descritos os indutores utilizados nos experimentos realizados, além de outros algoritmos que são citados no decorrer desta tese. Uma descrição mais detalhada sobre indutores pode ser encontrada em (Baranauskas & Monard 2000d).

ID3 O indutor ID3 é membro de uma família mais ampla de algoritmos de AM indutivo conhecida como *Top Down Induction of Decision Trees* – TDIDT (Quinlan 1986). É um algoritmo básico para a construção de árvores de decisão sem poda, na qual é conduzida uma busca gulosa (*greedy*), ou seja, o algoritmo não reconsidera escolhas anteriores.

A construção de uma árvore de decisão realiza-se da seguinte forma (Breiman, Friedman, Olshen & Stone 1984; Quinlan 1986): utilizando o conjunto de treinamento, um atributo é escolhido de forma a particionar os exemplos em subconjuntos, de acordo com valores deste atributo. Para cada subconjunto, outro atributo é escolhido para particionar novamente cada um deles. Este processo prossegue, enquanto um dos subconjuntos contenha uma mistura de exemplos pertencendo a classes diferentes. Uma vez obtido um subconjunto

uniforme — todos os exemplos naquele subconjunto pertencem à mesma classe — um nó folha é criado e rotulado com o mesmo nome da respectiva classe.

Quando um novo exemplo deve ser classificado, começando pela raiz da árvore induzida, ID3 testa e desvia para cada nó com o respectivo atributo até que atinja uma folha. A classe deste nó folha será atribuída ao novo exemplo. Se nenhuma condição da árvore for satisfeita, então existe uma regra padrão que atribui a classe mais comum (classe majoritária) ao novo exemplo.

A versão original do ID3 usa o critério de ganho de informação para escolher os nós de decisão. O critério de ganho é calculado baseado na entropia. Na $\mathcal{MLC}$++, o critério *default* utilizado é chamado de *Normalized-Mutual-Info*, o qual é muito semelhante ao critério de ganho.

C4.5 Este indutor é um dos sucessores do ID3 (Quinlan 1993). Muitas extensões foram acrescentadas ao algoritmo básico do ID3, tais como a melhora da eficiência computacional, tratamento de valores desconhecidos, tratamento de atributos contínuos, uso de *windowing* (vide Seção 7.5 na página 122) e o uso do critério de razão de ganho de informação, em substituição ao critério de ganho utilizado na versão original do ID3. A construção da árvore e classificação de novos exemplos é feita de forma similar ao ID3.

C4.5RULES O algoritmo C4.5RULES examina a árvore de decisão original produzida pelo C4.5 e então deriva um conjunto de regras da forma $L \rightarrow R$ (Quinlan 1993). O lado esquerdo L é uma conjunção de testes baseados nos atributos do conjunto de exemplos e o lado direito R é uma classe. Uma das classes é também designada como sendo a classe *default*.

Para classificar um novo exemplo, C4.5RULES examina a lista ordenada de regras para encontrar a primeira regra que cobre o exemplo. Então, a classe predita é aquela determinada pelo lado direito dessa regra. Se o exemplo não for coberto por nenhuma regra, ou seja, nenhum lado esquerdo das regras é satisfeito, o exemplo é classificado como pertencente à classe *default*. As regras são agrupadas por classes e não são ordenadas dentro de uma mesma classe (ou seja, é possível trocar a ordem entre regras prevendo uma mesma classe), mas são ordenadas inter-classes.

É importante notar que C4.5RULES não reescreve simplesmente a árvore de decisão para um conjunto de regras. Na verdade, ele generaliza as regras desconsiderando condições supérfluas, ou seja, condições irrelevantes que não afetam a conclusão, sem afetar a precisão e mantendo as regras mais importantes.

CN2 Este é um algoritmo de AM que induz regras da forma $L \rightarrow R$ em domínios em que pode haver ruído (Clark & Niblett 1987; Clark & Niblett 1989; Clark & Boswell 1991). Para valores desconhecidos, CN2 usa o método simples de substituir esses valores pelo valor mais comum se o atributo for nominal. Para atributos contínuos, o valor médio do intervalo mais comum substitui o valor desconhecido.

Este indutor pode gerar regras *não ordenadas* (*default* CN2) ou *ordenadas*. Para classificar um novo exemplo utilizando regras não ordenadas, todas as regras são avaliadas e aquelas que disparam, ou seja, cujas condições são verdadeiras, são coletadas. Se mais de uma classe é indicada pelas regras coletadas, CN2 associa, a cada regra, a distribuição de exemplos cobertos entre classes e totaliza essas distribuições para encontrar a classe mais provável.

Por exemplo, considere as três regras seguintes que classificam um robô nas duas classes [inimigo, amigo] nas quais [15,1] denota que a regra cobre 15 exemplos de treinamento da classe inimigo e 1 da classe amigo.

if	cabeça=quadrada	**and**	carrega=arma	**then**	classe=inimigo	cobre	[15,1]
if	altura=baixo	**and**	voa=não	**then**	classe=amigo	cobre	[1,10]
if	aparência=nervosa			**then**	classe=inimigo	cobre	[20,0]

Dado um novo exemplo de um robô que possui cabeça quadrada, carrega uma arma, é baixo, não voa e tem aparência nervosa, todas essas regras disparam. CN2 resolve isto somando os exemplos cobertos [36,11] e classificando o novo exemplo através da classe mais comum nessa soma — inimigo.

Para regras ordenadas, CN2 tenta classificar um novo exemplo pela primeira regra que dispara, desconsiderando todas as demais. Em ambos os casos (regras ordenadas ou não), existe a regra *default* que é utilizada caso nenhuma seja regra disparada.

Instance Based É um indutor preguiçoso, também denominado IB (*Instance-Based*). Ele é também conhecido como algoritmo K vizinhos mais próximos (*K-Nearest Neighbors* ou K-NN) (Mitchell 1998). A idéia geral é postergar a compilação do conjunto de treinamento, armazenando os exemplos. A classificação de um novo exemplo é efetuada com base no voto dos K exemplos mais próximos utilizando uma métrica de distância (Aha 1992; Aha 1997).

Naïve Bayes Este indutor, também denominado NB, usa a regra de Bayes para calcular a

probabilidade de cada classe dado um exemplo, assumindo que os atributos são independentes (Langley, Iba & Thompson 1992; Heckerman 1996). Formalmente,

$$
\begin{aligned}
p(y|\vec{x}) &= \frac{p(\vec{x}|y) \cdot p(y)}{p(\vec{x})} && \text{regra de Bayes} \\
&\propto p(\vec{x}|y) \cdot p(y) && p(\vec{x}) \text{ é igual para todas as classes} \\
&= p(x_1, x_2, \ldots, x_m|y) \cdot p(y) \\
&= p(x_1|y) \cdot p(x_2|y) \cdot \ldots \cdot p(x_m|y) \cdot p(y) && \text{por independência} \\
&= \prod_{j=1}^{m} p(x_j|y) \cdot p(y)
\end{aligned}
$$

Mesmo que, em domínios reais, os atributos não sejam independentes, o algoritmo é bem robusto a violações da condição de independência. As probabilidades para atributos nominais são estimadas através de contagem. A probabilidade para uma contagem de zero é assumida, pela biblioteca $\mathcal{MLC}$++, como sendo $1/2n$ para n exemplos. As probabilidades para atributos contínuos são estimadas assumindo uma distribuição normal e calculando a média e o desvio padrão a partir dos dados. Valores desconhecidos são ignorados, ou seja, eles não participam do produtório.

Além desses, algumas vezes são mencionados neste trabalho outros algoritmos. O algoritmo $\mathcal{CI}$ — MineSet™ *Column Importance* — não fornece um classificador, mas seleciona atributos relevantes de um conjunto de exemplos (Rathjens 1996, Capítulo 17). O algoritmo $\mathcal{CI}$ particiona os dados de forma similar a árvores de decisão. Uma medida denominada *pureza*, que varia de 0 a 100, informa quão bem os atributos discriminam as diferentes classes. Cada conjunto na partição tem sua própria medida de pureza e a medida de pureza dentro da partição é uma combinação dessas medidas individuais. Como em árvores de decisão, a relevância de um atributo depende daqueles previamente considerados. Uma diferença entre $\mathcal{CI}$ e árvores de decisão refere-se ao processo de discretização. O algoritmo $\mathcal{CI}$ usa um processo de discretização global, ou seja, ele discretiza todos os atributos contínuos antes de particionar os dados. Os algoritmos de árvore de decisão não pré-discretizam os atributos; eles encontram os valores de teste à medida que a árvore é construída. O tipo de discretização usado em experimento com o algoritmo $\mathcal{CI}$ foi a entropia, o *default* na ferramenta MineSet™.

O algoritmo CART induz árvores de decisão para classificação ou regressão (Breiman, Friedman, Olshen & Stone 1984). O classificador oblíquo OC1 é um algoritmo de indução projetado

para aplicações nas quais os atributos são numéricos (Murthy, Kasif & Salzberg 1994), tais como astronomia (Salzberg, Chandar, Ford, Murthy & White 1995) e análise de seqüências de DNA (Salzberg 1995a). OC1 constrói árvores de decisão que contém uma combinação linear de atributos em cada nó interno. Essas árvores, portanto, particionam o espaço de descrição com hiperplanos paralelos aos eixos, bem como hiperplanos oblíquos.

Um outro classificador citado neste trabalho é o perceptron, algumas vezes denominado neurônio, associado à regra de correção de erro perceptron (Rosenblatt 1958; Minsky & Papert 1988). No contexto deste trabalho redes neurais são redes de perceptrons de várias camadas, usualmente associadas com o método de aprendizado *backpropagation* (Rumelhart, Hinton & Williams 1986), embora existam diversos outros tipos de redes neurais.

Finalmente, um outro elemento citado é a regra ou classificador de Bayes B^*, que prevê a classe mais comum para um dado exemplo, com base na distribuição D (que é desconhecida). A precisão da regra de Bayes é a maior possível e é de interesse mais teórico já que na prática a distribuição D é desconhecida.

3.5 Ferramentas

In the middle of difficulty lies opportunity.

—*Albert Einstein*

Grandes esforços têm sido feitos para definição e integração de algoritmos de Aprendizado de Máquina. A seguir são citadas algumas ferramentas existentes.

1. AC2 é um conjunto de biblioteca C/$\mathcal{C}$++ que permite embutir funcionalidades de Mineração de Dados em outros softwares. Encontra-se disponível em `http://www.isoft.fr/html/prod_ac2.htm`. AC2 abrange todo o processo de Mineração de Dados, desde modelagem de dados, orientada a objetos, até avaliação da precisão obtida.

2. $\mathcal{MLC}$++ é um projeto de Aprendizado de Máquina em $\mathcal{C}$++ (Machine Learning in $\mathcal{C}$++) e foi desenvolvido em 1993 na Universidade de Stanford (Kohavi, Sommerfield & Dougherty 1994; Kohavi, Sommerfield & Dougherty 1996) e em 1995 passou a estar sob a responsabilidade da Silicon Graphics. O projeto tem como objetivo facilitar o uso dos algoritmos de AM bem como auxiliar pesquisadores da área em experimentos com novos algoritmos ou com modificações nos algoritmos existentes. $\mathcal{MLC}$++ disponibiliza uma biblioteca de classes e funções em $\mathcal{C}$++ implementando os algoritmos principais. Essa biblioteca fornece diferentes implementações dos algoritmos como também integra-os na biblioteca $\mathcal{C}$++

com uma metodologia de codificação unificada e separação dos componentes em classes de $\mathcal{C}$++ (Felix, Rezende, Doi, de Paula & Romanato 1998).

Esta biblioteca foi utilizada para executar os experimentos reportados no Capítulos 5 e 6, sendo de domínio público, incluindo o código fonte, que pode ser obtido em `http://www.sgi.com/tech/mlc/`. A biblioteca contém implementados os principais algoritmos de indução, tais como ID3, K-NN, *naïve* Bayes, etc. $\mathcal{MLC}$++ contém interfaces para os principais algoritmos C4.5, IB 1-4, OC1 e CN2. $\mathcal{MLC}$++ também conta com *wrappers* que utilizam os algoritmos como caixas-preta para estimativa de precisão de classificação utilizando amostragem, seleção de atributos, filtros de discretização de valores, etc. Dentre algumas facilidades, $\mathcal{MLC}$++ fornece um formato padrão de entrada de dados, similar a Tabela 2.1 na página 21, para todos os algoritmos; obtenção de estatísticas de desempenho, tais como precisão, taxa de aprendizado e matriz de confusão; e visualização gráfica das estruturas aprendidas, por exemplo, árvores de decisão. Alguns dos algoritmos suportam visualização dos classificadores e podem gerar saídas para o software MineSet™. Entretanto, o código disponível data de 1997 e sua compilação não é trivial de ser efetuada.

3. MineSet™ é um produto da Silicon Graphics para análise exploratória de dados `http://www.sgi.com/Products/software/MineSet/`. Combina várias ferramentas integradas e interativas para acesso e transformação de dados, Mineração de Dados e visualização. Este software usa $\mathcal{MLC}$++ como base para os algoritmos de indução. Entretanto, por se tratar de um produto comercial, alguns detalhes, sobre a implementação e funcionamento interno de suas ferramentas são omitidos, o que dificulta sua análise por pesquisadores da área de AM.

   Os recursos desta ferramenta são compostos por alguns dos principais algoritmos de AM e um conjunto de poderosas ferramentas de visualização, as quais permitem, além da visualização dos dados, a visualização de estatísticas geradas sobre esses dados e de resultados obtidos a partir da aplicação das ferramentas de Mineração de Dados (Rezende & Pugliesi 1998). Os algoritmos de AM utilizados pelo MineSet™, versão 2.6, encontram-se implementados na biblioteca $\mathcal{MLC}$++. O algoritmo $\mathcal{CI}$ desta ferramenta foi utilizado nos experimentos reportados no Capítulo 5.

4. Mobal é uma ferramenta que integra técnicas manuais de aquisição de conhecimento com vários algoritmos computacionais envolvendo Lógica de Primeira Ordem. O sistema é baseado em regras relacionais e o conhecimento do domínio é armazenado em uma sintaxe similar à lógica de predicados (Morik, Wrobel, Jörg-Uwe & Emde 1993).

5. Weka é um conjunto de algoritmos de AM, escritos em Java e de domínio público, podendo ser obtido em `http://www.cs.waikato.ac.nz/~ml/weka/`. Diferentemente da biblioteca $\mathcal{MLC}$++, que possui interfaces para indutores já existentes, Weka adota uma outra perspectiva. Todos os algoritmos são implementados em Java, tanto novos como aqueles pré-existentes. Por exemplo, o indutor C4.5, originalmente escrito em C foi recodificado em Java.

   Ainda que esse processo de recodificar algoritmos padroniza as interfaces e produza código uniforme, as novas versões dos algoritmos originais podem não ser disponibilizadas na Weka, pois exigem a conversão em código Java. Além disso, a recodificação de algoritmos sempre está sujeita a falhas, as quais podem causar um comportamento anômalo do algoritmo em Java que não ocorre no código fonte original.

3.6 Considerações Finais

> It would have been especially interesting to read what Quinlan thinks of the numerous experiments comparing C4.5 and its predecessors to the other classification algorithms that are commonly used in the research community.
>
> —*Steven L. Salzberg, Book Review of (Quinlan 1993)*

É comum, em Aprendizado de Máquina supervisionado, a necessidade de encontrar-se um classificador que possa ser usado para predizer, com a melhor precisão possível, as classes de novos exemplos. Com esse objetivo, durante os últimos anos, muitos algoritmos de AM supervisionado foram desenvolvidos.

Neste capítulo foram mostradas diversas técnicas de amostragem que, exceto pelo método de resubstituição, são normalmente utilizadas para estimar o desempenho de um algoritmo. Com base nisso, foi escolhida a metodologia de comparação entre dois algoritmos, a qual é utilizada nos experimentos por nós realizados e descritos nesta tese em capítulos subseqüentes. Finalmente, foi fornecida uma descrição dos conjuntos de exemplos, indutores e ferramentas que foram usados nos experimentos reportados nos próximos capítulos ou apenas citados no decorrer desta tese.

O capítulo seguinte descreve uma área de pesquisa em Inteligência Artificial que vem recebendo grande interesse pela comunidade científica, e que utiliza técnicas de Aprendizado de Máquina: a Extração de Conhecimento em grande volume de dados.

Capítulo 4

Extração de Conhecimento & Mineração de Dados

> Discovery consists of seeing what everybody has seen and thinking what nobody has thought.
>
> —*Albert von Szent-Gyoergyi*

Tradicionalmente, a transformação de dados em informação útil (conhecimento) baseia-se em análises e interpretações manuais. Por exemplo, em uma companhia de seguros de saúde, é comum os especialistas periodicamente analisarem as tendências e alterações nos dados de saúde de seus clientes; a partir daí, os especialistas então geram um relatório que será usado para decisões futuras quanto à forma e custos de atendimento. Em um outro extremo, geologistas planetários procuram imagens de planetas e asteróides remotos em chapas fotográficas, cuidadosamente localizando e catalogando objetos de interesse, tais como vulcões e crateras de impacto.

Em geral, o método clássico de análise de dados reside em um ou mais analistas humanos tornando-se intimamente familiar com os dados e atuando como uma interface entre dados e usuários. Nota-se, facilmente, que a forma manual de tratamento de dados é lenta, cara e altamente subjetiva. É comum que tais dados residam em uma base de dados computadorizada, facilitando a extração de muitos relatórios ou consultas. Mas, ainda assim, o analista humano deve comparar relatórios, cruzar informações e utilizar seu conhecimento prévio sobre a área de atuação para tentar extrair *algo novo* dos dados existentes (Monard, Caulkins, Baranauskas, Oliveira & Rezende 1999).

Na medida em que as bases de dados atuais crescem cada vez mais, essa abordagem torna-se impraticável em vários domínios, principalmente na descoberta de informação útil aos usuários (Yo-

on & Kerschberg 1993). Isso pode ser facilmente explicado sabendo-se que uma base de dados possui três dimensões principais: número de exemplos n, número de atributos m e a quantidade de valores de um atributo. Bases de dados contendo registros da ordem de $n = 10^9$ exemplos tornam-se cada vez mais comuns, por exemplo, em Astronomia. De forma similar, o número de campos pode facilmente atingir a ordem de $m = 10^2$ ou mesmo $m = 10^3$ em aplicações, por exemplo, de diagnóstico médico (Teller & Veloso 1995). É muito improvável que um ser humano consiga analisar e inferir algo importante quando se possui milhões de registros, cada um contendo centenas ou milhares de campos.

Neste capítulo é proposta uma metodologia para extrair conhecimento de grandes conjuntos de exemplos, detalhando-se suas diversas etapas nas seções que se seguem. O termo *Extração de Conhecimento* de bases de dados é comumente conhecido como *Knowledge Discovery in Databases* — KDD. Ainda neste capítulo é mostrada que a Mineração de Dados, também conhecida como *Data Mining* — DM — é uma das etapas do processo global de KDD. Finalmente, alguns trabalhos relacionados são citados bem como problemas e desafios na área de KDD.

4.1 Etapas do Processo de Extração de Conhecimento

> I do not feel obliged to believe that the same God who has endowed us with sense, reason, and intellect has intended us to forgot their use.
>
> —*Galileo Galilei*

Quando o tamanho de uma base de dados é muito grande, tentativas de extrair conhecimento manualmente da base de dados são, geralmente, improdutivas. Normalmente, recorre-se a uma metodologia computacional que automatize o processo de manipulação de dados, visando a extração de informação útil. Inicialmente, foi proposta uma metodologia por Fayyad, Piatetsky-Shapiro & Smyth (1996a), composta por nove etapas, e mais tarde outra por Weiss & Indurkhya (1998), composta por quatro etapas. Neste trabalho, efetuou-se uma combinação e compactação dessas duas metodologias, que favorece a compreensão geral do processo de KDD, resultando nas três etapas descritas a seguir e esquematizadas na Figura 4.1 na próxima página:

1. Pré-processamento:
   1.1 Preparação de Dados;
   1.2 Redução de Dados;
2. Mineração de Dados;

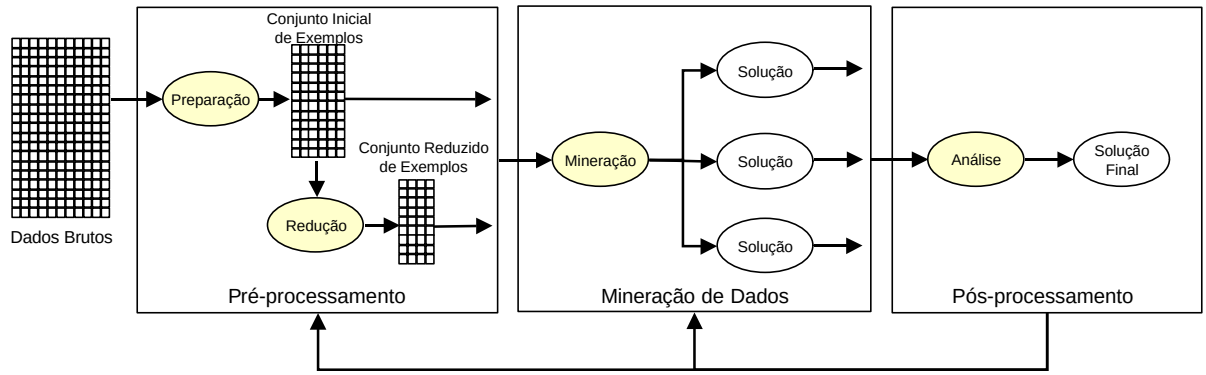


Figura 4.1: Etapas do processo de Extração de Conhecimento

3. Pós-processamento ou Análise da Solução.

De forma resumida, a etapa de pré-processamento, na qual se preparam os dados para o processo de mineração, pode ser entendida como duas sub-etapas. Na sub-etapa de preparação, os dados são coletados e transformados para início do processo de mineração. A sub-etapa de redução, opcional quando a quantidade de dados é moderada, diminui a quantidade de dados de forma a viabilizar a aplicação da etapa de mineração em grandes bases utilizando algoritmos de AM simbólicos, objeto de estudo neste trabalho. A etapa de mineração procura por soluções que podem ter diferentes objetivos e complexidade. Por último, a etapa de pós-processamento ou análise das soluções obtidas é efetuada, consolidando os resultados obtidos em uma solução final que será apresentada ao usuário. É possível retornar de uma etapa para etapas anteriores, visando uma melhoria dos resultados que somente tenha sido percebida como possível em uma etapa mais adiantada. Uma descrição um pouco mais detalhada de cada etapa é fornecida nas seções seguintes.

4.1.1 Pré-Processamento

Preparação de Dados

Esta é uma etapa crucial em todo o processo de KDD, freqüentemente recebendo pouca atenção na literatura pelo fato de ser considerada muito específica pois, na prática, requer realizar um *reconhecimento do domínio da aplicação* considerando-se o conhecimento prévio relevante e os objetivos da aplicação.

Na Figura 4.2 na página seguinte são descritos os relacionamentos principais para a preparação de dados, etapa na qual os dados brutos são transformados no formato atributo-valor definido na Tabela 2.1 na página 21, ou seja, no formato padrão. Embora outros formatos pos-

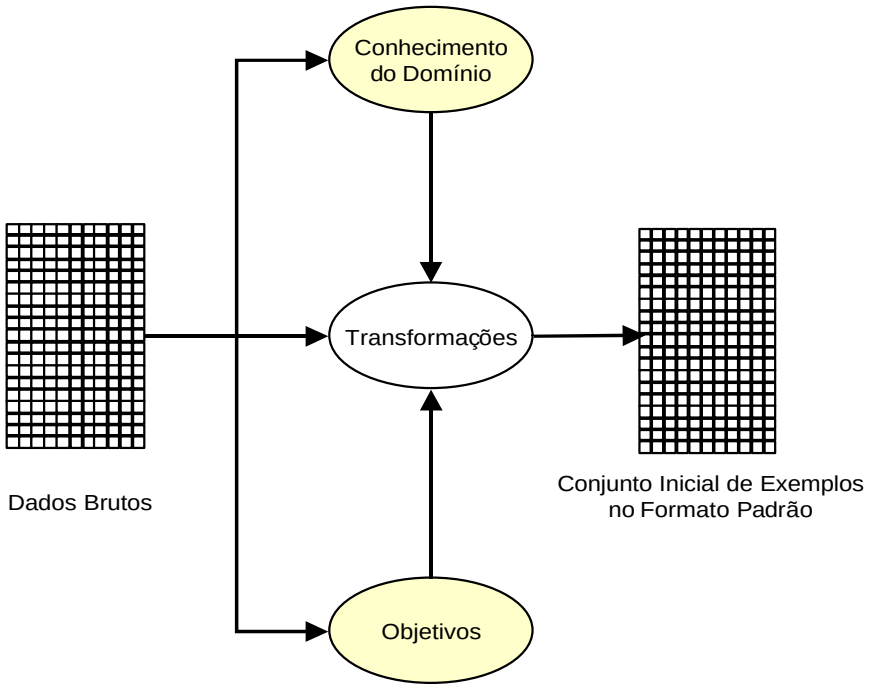


Figura 4.2: Preparação de dados

sam ser definidos, o formato padrão adotado coloca os dados originais em uma representação simples e uniforme que é universalmente aceitável para várias técnicas de Mineração de Dados.

Os dados brutos nem sempre são adequados para Mineração de Dados. Várias *transformações* podem ser necessárias para produzir atributos e exemplos com um bom poder de predição, tais como:

Definição de atributos Com base nos dados brutos e no conhecimento prévio do domínio, é necessário definir quais atributos são importantes para atingir a meta do processo de KDD. A definição dos atributos inicialmente é efetuada de forma manual, quando o especialista humano seleciona um subconjunto do total de atributos disponíveis nos dados brutos. Como isso implica que muitas decisões de um ser humano estão envolvidas, em caso de dúvida, deve-se incluir atributos extras. Isso deve-se ao fato que os algoritmos de aprendizado têm facilidade de lidar com atributos extras, mas possuem dificuldades no processo de compor novos atributos com maior capacidade de predição.

Extração e integração Os dados brutos podem se encontrar sob diferentes formas de armazenamento, tais como arquivos, base de dados ou *dataware house*. Assim, é necessário realizar a extração e integração dos dados provenientes de diferentes fontes em diferentes formatos, para o formato padrão. No caso de dados relacionais, isso pode requerer a junção

ou projeção de várias tabelas com relações de diferentes cardinalidades (um-para-muitos ou muitos-para-muitos) em uma única tabela (Monard, Caulkins, Baranauskas, Oliveira & Rezende 1999).

Transformação de dados Após a extração e integração, os dados brutos devem ser transformados em dados mais apropriados para extrair conhecimento. Algumas destas transformações podem ter sido incluídas nos dados brutos; outras poderão ser encontradas ou definidas durante essa etapa; outras poderão somente ser detectadas como necessárias quando a etapa de Mineração de Dados ocorrer.

Dentre as transformações usuais tem-se o resumo de dados (por exemplo, dados sobre vendas individuais podem ter sido armazenados, mas um resumo diário talvez seja mais indicado para a tarefa em questão), a transformação de tipos de dados (por exemplo, um algoritmo de aprendizado pode não ser capaz de lidar com atributos do tipo data, o que pode requerer que este atributo seja transformado no número inteiro de segundos a partir de uma determinada data inicial ou em períodos, tais como semanas, meses ou anos), e a normalização de valores. Embora os dados no formato padrão possam ser usados por uma variedade de algoritmos, alguns deles podem requerer dados normalizados de forma a obter melhores resultados; neste caso, os dados são colocados em um intervalo específico de valores, normalmente entre -1 e $+1$. Esses algoritmos também podem requerer a verificação de informação temporal, mapeamento de valores desconhecidos ou ausentes, ou ainda encontrar representações invariantes para alguns dados.

Limpeza O principal objetivo do pré-processamento de dados consiste em extrair conhecimento que será usado para resolver problemas ou tomar decisões. Entretanto, problemas com os dados podem evitar que isso seja efetuado com sucesso. Os dados são adquiridos na forma de atributos simbólicos ou numéricos de uma variedade de fontes. Estas fontes podem variar desde seres humanos até sensores com diferentes graus de qualidade (Famili, Shen, Weber & Simoudis 1997).

A qualidade dos dados é um ponto central em KDD, uma vez que muitos indutores não são capazes de considerar informações adicionais durante a fase de aprendizado. De forma a assegurar a qualidade dos dados, técnicas de limpeza devem ser aplicadas aos dados antes de fornecê-los como entrada para os algoritmos de aprendizados.

A limpeza semi-automática é uma tarefa demorada, principalmente em grande volume de dados provenientes de sistemas automáticos de aquisição de dados. Portanto, técnicas

automáticas de limpeza são importantes. De forma geral, elas podem ser divididas em dois grupos de tarefas:

1. tarefas específicas do domínio, que são solucionadas por técnicas que utilizam o conhecimento do domínio, tais como verificação de consistência dos atributos e granularidade dos dados; os exemplos errôneos devem ser eliminados ou corrigidos;
2. tarefas independentes do domínio, que podem ser automatizadas, tais como a decisão da estratégia para o tratamento de atributos incompletos, remoção de ruído, tratamento de conjuntos de exemplos não balanceados, seleção de um subconjunto de atributos, construção de atributos e seleção de exemplos (Batista, Carvalho & Monard 2000).

Um dos problemas mais importantes na limpeza de dados consiste em como saber se uma informação precisa, existente nos dados brutos, não está sendo destruída. Dessa forma, as técnicas de limpeza devem ser cuidadosamente escolhidas, de forma a introduzir a menor distorção possível.

Por exemplo, considerando o tratamento de valores desconhecidos, muitos indutores são capazes de tratá-los de forma automática. Contudo, esse tratamento é efetuado através de técnicas simples, por exemplo, a substituição de todos os valores desconhecidos de um atributo pela sua média ou pelo valor mais freqüente. Este método de substituição pode introduzir um *bias* nos dados, uma vez que o relacionamento entre os atributos não é considerado.

Uma técnica mais eficiente para tratar valores desconhecidos consiste em testar como eles estão distribuídos antes de qualquer manipulação de dados. Valores desconhecidos distribuídos aleatoriamente são considerados menos perigosos e podem ser manipulados por técnicas simples, tais como a remoção de todos os exemplos com valores desconhecidos. Entretanto, valores desconhecidos distribuídos não aleatoriamente devem ser tratados com maior cuidado, através de métodos mais robustos. Um método robusto para o tratamento de valores desconhecidos consiste na criação de um modelo, utilizando um sistema de aprendizado para predizer os valores desconhecidos. Vários sistemas de aprendizado podem ser utilizados para criar esses modelos, tais como redes neurais e árvores de decisão. Porém, essa solução pode consumir uma quantidade considerável de tempo e alguns modelos mais simples, tais como K-NN podem fornecer soluções de forma mais rápida para um grande volume de dados (Batista 2000).

Composição de atributos Em alguns casos, existem transformações adicionais que podem apresentar um impacto muito grande nos resultados. Neste sentido, a composição de atributos é um fator determinante na qualidade dos resultados, muito maior do que o próprio método de mineração adotado para produzir os resultados. Em muitos casos, a composição de atributos é dependente do domínio da aplicação. A composição de atributos é tratada no Capítulo 6, bem como o trabalho por nós desenvolvido sobre este tema.

Redução de Dados

Considerando a etapa de preparação de dados, é possível que uma grande quantidade de dados brutos resulte em um conjunto de exemplos, no formato padrão, de tamanho relativamente moderado. Neste caso, é possível aplicar algoritmos de mineração diretamente. Entretanto, para grandes conjuntos de exemplos, é bem provável que a etapa de redução de dados seja necessária antes da utilização dos algoritmos de mineração, mostrada na Figura 4.3.

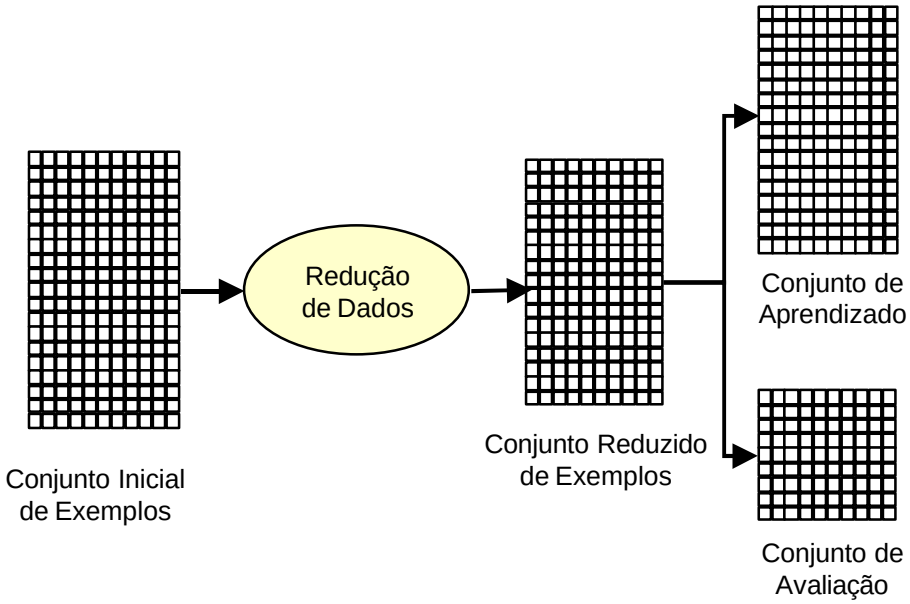


Figura 4.3: Redução de dados

Uma razão para reduzir a dimensão dos dados é que a quantidade de dados podem ser muito grande para ser minerada. Com dados atingindo a ordem de terabytes, é muito simples exceder a capacidade de processamento de um algoritmo de mineração ou mesmo de um computador. Outra razão refere-se ao tempo de processamento para encontrar uma solução, que pode ser muito longo, particularmente quando algumas variações são consideradas na fase de mineração.

A redução de dados é uma etapa mediadora entre a preparação de dados e a Mineração de

Dados. Entretanto, é possível considerar técnicas que removem atributos como métodos para preparação de dados. De forma similar, métodos que transformam dados em um novo conjunto de atributos podem também ser considerados como métodos de preparação de dados. Embora técnicas de limpeza de dados possam ser utilizadas para a redução de dados e vice-versa, a finalidade de cada uma é diferente. A redução procura tornar viável o processo de mineração na presença de um grande volume de dados, enquanto a limpeza procura garantir a qualidade dos dados.

Considerando as três dimensões de um conjunto de exemplos (número de exemplos, número de atributos e a quantidade de valores de um atributo), qualquer operação que altere esses valores afeta a dimensão dos dados. No caso de redução da dimensão, as operações são:

(i) remoção de um exemplo;

(ii) remoção de um atributo;

(iii) redução do número de valores de um atributo (suavizar, discretizar ou agrupar valores de um atributo).

Estas operações tentam preservar a característica dos dados originais pela eliminação daqueles não essenciais, suavizando ou discretizando algumas características. Existem outras operações que reduzem a dimensão dos dados, mas os novos dados produzidos por elas podem ser irreconhecíveis quando comparados com os originais. Por exemplo, indução construtiva pode ser empregada para gerar novos atributos derivados a partir dos originais, descartando-se os atributos originais usados na construção do atributo derivado.

Para a maioria das aplicações, o conjunto de exemplos no formato padrão possui muito mais exemplos do que atributos. A tendência, para um grande volume de dados, é que o número de exemplos cresça muito mais rapidamente do que o número de atributos. Por exemplo, uma rede de supermercado pode ter um aumento no número de vendas, mas o número de atributos armazenados por venda tende a não crescer muito. Dessa forma, a remoção de um atributo tem um impacto muito maior na redução de dados do que a remoção de um exemplo. Mesmo não sendo absolutamente necessária, a aplicação de técnicas de redução de dados pode ocorrer uma ou mais vezes no processo de KDD. A seleção de atributos é tratada no Capítulo 5, bem como o experimentos efetuados sobre esse tópico.

Em resumo, o conjunto de exemplos inicial no formato padrão é analisado e processado para produzir um subconjunto do conjunto original, mas com menos atributos. Isso deve ser efetuado de forma relativamente rápida (em relação ao prazo para término da tarefa em questão),

reduzindo o número de atributos ou a quantidade de valores de atributos de forma aceitável à etapa de mineração.

O objetivo das técnicas de redução é que elas sejam genéricas e independentes dos métodos de mineração. Mesmo assim, a utilidade delas será variável de acordo com a dimensão dos dados e dos métodos de mineração. Alguns algoritmos de mineração são muito mais rápidos do que outros. Alguns possuem técnicas de seleção de atributos embutidas e inseparáveis do próprio algoritmo. As técnicas de redução de dados são eficazes, mas não são perfeitas. Uma atenção cuidadosa deve ser tomada na avaliação de resultados experimentais intermediários de forma a definir as reduções mais adequadas dentre um conjunto de alternativas. Uma vez reduzido ou não, o conjunto de exemplos pode ser dividido no conjunto de treinamento e no conjunto de teste ao final da etapa de pré-processamento.

4.1.2 Mineração de Dados

Existem muitos métodos disponíveis para a Mineração de Dados, mostrada esquematicamente na Figura 4.4 na próxima página. Independente do método escolhido, a Mineração de Dados adequada pode exigir muita experimentação de forma a ajustar o método para o melhor desempenho ou outras características desejáveis, tais como simplicidade da solução. Os experimentos podem ser vistos como uma busca em um espaço de estados consistindo de combinações de parâmetros e opções. Os nós deste espaço, representando combinações, podem ser testados em seqüência e a variação que fornece o melhor resultado é selecionada. Embora possam existir muitos parâmetros a serem ajustados, a experiência sugere que poucos parâmetros resultam em um grande impacto sobre o resultado final. De fato, esta etapa pode ser vista como composta por três fases:

escolha da função ou tarefa envolve a decisão sobre o *objetivo* da solução obtida pelo algoritmo de mineração, por exemplo, resumo, classificação, regressão, categorização ou mesmo visualização (Agrawal, Mannila, Srikant, Toivonen & Verkamo 1996). Associada à escolha da função, encontra-se a decisão sobre *representação* da hipótese. As representações mais comuns incluem árvores de decisão, regras, modelos lineares, modelos não lineares (*e.g.*, redes neurais), modelos baseados em exemplos (*e.g.*, K-NN e raciocínio baseado em casos), modelos de dependência probabilística, tais como redes Bayesianas e modelos relacionais.

A representação da solução determina tanto a capacidade de representação do conceito embutido nos dados, como a facilidade de interpretação do modelo em termos humanos. Geralmente, quanto mais complexa é a solução, melhor ela se ajusta aos dados, mas

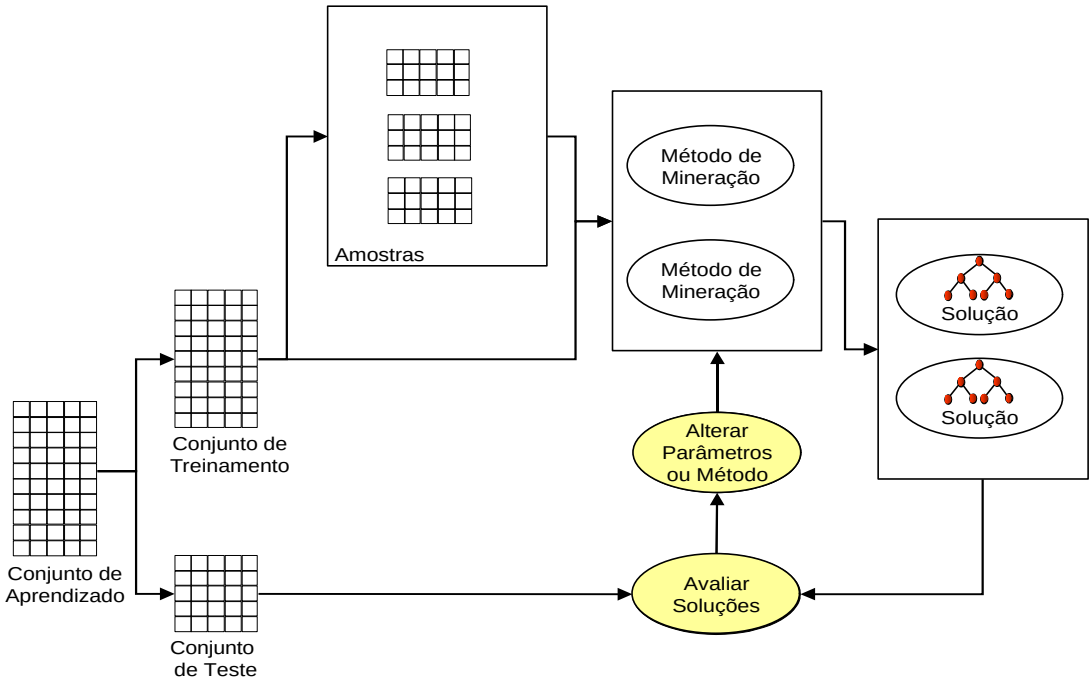


Figura 4.4: Mineração de Dados

também pode ser mais difícil de interpretar e de se provar confiável em novos exemplos.

Diferentemente de pesquisadores teóricos que usualmente preferem soluções complexas, pesquisadores envolvidos em aplicações reais, freqüentemente usam soluções mais simples devido a sua robustez e grau de interpretação (Fayyad & Uthurusamy 1995; Fayyad, Piatetsky-Shapiro, Smyth & Uthurusamy 1996);

escolha do algoritmo envolve a seleção dos algoritmos a serem usados, bem como a decisão de quais parâmetros podem ser apropriados, de forma a compatibilizar o método de mineração ao processo geral de Extração de Conhecimento (por exemplo, o usuário pode estar mais interessado em entender o modelo do que na sua capacidade de predição);

mineração busca por padrões de interesse em uma forma de representação particular ou em um conjunto de tais representações, incluindo árvores ou regras de classificação, regressão, categorização, visualização, entre outras (Han & Cercone 2000; Yang 2000).

Na literatura da área, encontrar padrões úteis nos dados é conhecido por diversas formas, tais como *extração de conhecimento*, *descoberta de informação*, *arqueologia de dados*, *mineração de dados*, entre outras. Entretanto, diferencia-se, no contexto deste trabalho, KDD e DM. O termo *Mineração de Dados* refere-se ao processo de resolução de problemas que encontra uma descrição lógica ou matemática, eventualmente de natureza complexa,

de padrões e regularidades em um conjunto de exemplos (Decker & Focardi 1995). De forma mais simples, DM é a aplicação de algoritmos que extraem informações úteis dos dados.

O termo *Mineração de Dados* foi e é muito usado mas com conotação diferente da adotada neste trabalho (que é a mesma da literatura dos últimos anos em Inteligência Artificial). Na década de 60, DM teve conotações negativas em Estatística quando a análise de dados baseada em computadores estava apenas se iniciando. O conceito é que, ao se realizar muitas buscas em qualquer conjunto de exemplos — mesmo aqueles gerados aleatoriamente — pode-se encontrar padrões que parecem ser estatisticamente significantes mas, que de fato, não são. Isso é de fundamental importância. É óbvio que a aplicação incorreta de métodos de mineração (altamente criticada como escavação de dados na literatura estatística) pode ser uma atividade catastrófica, levando à descoberta de padrões sem sentido. DM é uma atividade legítima desde que seja efetuada por alguém que entenda como fazê-la corretamente. No Capítulo 8 é proposto e desenvolvido o sistema XRULER, voltado para a Mineração de Dados.

4.1.3 Pós-Processamento

Após a etapa de mineração, tem início o pós-processamento, esquematicamente representado na Figura 4.5 na página seguinte. Nesta etapa, ocorre a *interpretação dos resultados*, que consiste na avaliação dos padrões descobertos, visualização dos padrões extraídos, remoção de padrões irrelevantes ou redundantes e tradução de padrões úteis em termos inteligíveis pelos usuários. A seguir, tem-se o *uso do conhecimento extraído*, que consiste na incorporação do conhecimento no desempenho do sistema, tomando ações baseadas no conhecimento ou simplesmente documentando e relatando para as partes interessadas o conhecimento obtido, bem como remoção de conflitos potenciais com conhecimento previamente tido como correto (ou extraído).

4.2 Considerações Finais

> All truths are easy to understand once they are discovered; the point is to discover them.
>
> —*Galileo Galilei*

Extração de Conhecimento é uma área recente de pesquisa em Inteligência Artificial. KDD é uma área interdisciplinar resultante da intersecção de pesquisas em várias outras — base de dados, estatística, Aprendizado de Máquina, lógica *fuzzy*, aquisição de conhecimento para sistemas

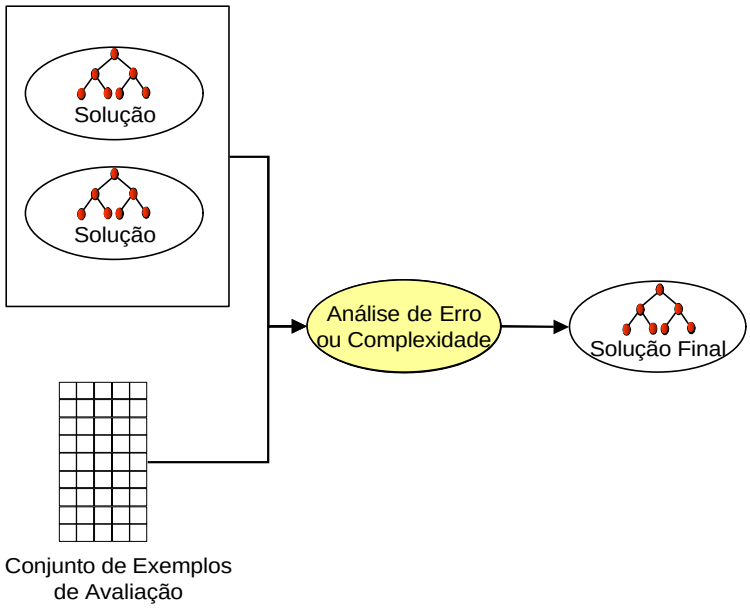


Figura 4.5: Análise da solução

baseados em conhecimento, visualização de dados, recuperação de informação e computação de alto desempenho, a qual incorpora teorias, algoritmos e métodos de todas essas áreas.

A área de KDD é ainda relativamente nova e ferramentas que suportem todo o processo de maneira simples de utilizar são raras. Entretanto, um dos tópicos que desafia pesquisadores é o uso de técnicas de Aprendizado de Máquina em grande volume de dados. A maioria das técnicas de AM aplicam-se a pequenos conjuntos de exemplos que podem caber na memória principal. Todavia, em aplicações de KDD essas técnicas devem ser empregadas com grandes conjuntos de exemplos.

Várias abordagens têm sido utilizadas para ultrapassar essa limitação. Entre elas tem-se o uso de paralelismo, desenvolvimento de novas técnicas orientadas a bases de dados, redução da dimensão dos dados e amostragem dos dados, entre outras. Todavia, muito esforço será necessário, antes que KDD possa ser aplicado com sucesso em grandes conjuntos de exemplos. Somente então o verdadeiro potencial de KDD poderá ser percebido.

Neste capítulo foi proposta uma abordagem para o processo de KDD que consiste em três etapas: pré-processamento, mineração e análise da solução. A etapa de pré-processamento normalmente é a que consome a maior quantidade de tempo do processo de KDD e é fundamental para a aplicação das duas etapas seguintes. Nos dois próximos capítulos são reportados experimentos, por nós efetuados, abordando dois aspectos da etapa de pré-processamento: seleção de atributos e composição de atributos.

Capítulo 5

Seleção de Atributos

> True knowledge exists in knowing that you know nothing. And in knowing that you know nothing, that makes you the smartest of all.
>
> —*Socrates*

O estágio atual da tecnologia permite a coleta e o armazenamento de um volume de dados cada vez maior. Com receio que algo seja perdido durante o processo de aprendizado, normalmente decide-se incluir todos os atributos, deixando para o algoritmo de Aprendizado de Máquina selecionar aqueles mais importantes. Por exemplo, o aprendizado de regras de diagnóstico para diferentes doenças, provenientes de vários registros médicos. Esses registros, freqüentemente, apresentam muito mais informação (atributos) da que é necessária para descrever cada doença. No caso específico para aprender o diagnóstico individual entre uma doença cardíaca e uma doença hormonal, sabe-se que os atributos relevantes para uma não são, necessariamente, os mesmos para a outra. Entretanto, em muitas situações, não é simples escolher ou realizar o processo de seleção de atributos, pois não se sabe exatamente quais são mais relevantes para o aprendizado de um determinado conceito.

A seleção de atributos úteis é, intrisicamente, um dos problemas principais em AM. Embora a maioria dos algoritmos de aprendizado tente selecionar atributos, ou então atribuir graus de importância a eles, análises teóricas e estudos experimentais indicam que muitos algoritmos apresentam um comportamento ruim com a presença de um grande número de atributos irrelevantes. Por exemplo, o número de exemplos necessário para o algoritmo K-NN atingir um dado nível de precisão parece crescer exponencialmente com o número de atributos irrelevantes, independente do conceito a ser aprendido. Mesmo os indutores de árvores de decisão, que explicitamente selecionam alguns atributos em detrimento de outros, exibem este comportamento

para alguns conceitos. Outros algoritmos, tais como *naïve* Bayes, são robustos com relação à presença de atributos irrelevantes, mas podem ser extremamente sensíveis em domínios contendo atributos fortemente correlacionados, mesmo se os atributos são relevantes. Isto pode ser explicado pelo fato que esses algoritmos assumem independência entre os atributos. Assim, isso sugere a necessidade de métodos adicionais para selecionar um subconjunto útil de atributos quando muitos deles encontram-se disponíveis (Langley 1996).

Na seção seguinte é introduzida a motivação para a seleção de atributos. Em seguida, são descritas três abordagens que podem ser utilizadas para FSS: embutida, filtro e *wrapper*. Utilizando as duas últimas abordagens são realizados experimentos que avaliam seu desempenho, reportados na Seção 5.4. Em seguinda é analisado, em experimentos adicionais, o impacto da seleção de atributos no classificador simbólico induzido por CN2.

5.1 Motivação

> For the things we have to learn before we can do them, we learn by doing them.
>
> —*Aristotle, Nichomachean Ethics*

Do ponto de vista conceitual, a tarefa de aprendizado de conceitos pode ser dividida em (i) decidir que atributos utilizar na descrição do conceito e (ii) decidir como combinar esses atributos. Sob esse aspecto, como mencionada, a seleção de atributos relevantes e a eliminação dos irrelevantes constitui um dos principais problemas a serem tratados em AM.

Do ponto de vista prático, é desejável que os algoritmos de indução trabalhem bem em domínios que contenham muitos atributos irrelevantes. Em outras palavras, um dos objetivos é que o número de exemplos de treinamento necessários para se atingir uma dada precisão cresça lentamente em relação ao número de atributos irrelevantes. Por exemplo, na tarefa de classificação de texto, é comum a representação de exemplos utilizando de 10^3 a 10^8 atributos, mas sabe-se que apenas uma pequena fração desses atributos é importante para o aprendizado do conceito. Nos últimos anos, uma quantidade crescente de pesquisas em AM, tanto teóricas quanto práticas, tem sido voltadas para o desenvolvimento de algoritmos que trabalhem bem na presença de muitos atributos irrelevantes.

Assim, intuitivamente, seria desejável que um indutor utilizasse apenas os atributos relevantes para o aprendizado do conceito. Existem diferentes definições na literatura para o significado da relevância de um atributo, dependendo dos objetivos a serem alcançados, tais como relevância em relação ao conceito meta e em relação a distribuição de probabilidade, entre outros. Diver-

sas definições sobre relevância de atributos podem ser encontradas em (Blum & Langley 1997; Kohavi & John 1997; Lee 2000).

Na maioria das aplicações, o conjunto de exemplos possui muito mais exemplos (linhas) do que atributos (colunas). Assim, remover uma coluna (atributo) tem um efeito mais dramático do que remover uma linha (exemplo). Embora não absolutamente necessária, a aplicação de métodos de seleção de um subconjunto de atributos (*Feature Subset Selection* — FSS) pode ser efetuada mais de uma vez, durante a fase de pré-processamento de dados. O conjunto original de exemplos é analisado e processado para produzir um subconjunto, com menos atributos. O tempo de processamento, para produzir o subconjunto de exemplos utilizando FSS, deve ser relativamente pequeno e pode resultar em uma redução dramática na dimensão dos atributos.

O processo de FSS pode ser descrito como uma busca pelo espaço de estados, onde cada nó (estado) representa um subconjunto de atributos; o valor de um nó é uma estimativa da precisão na classificação e os operadores são, geralmente, adicionar ou remover atributos.

Na próxima seção, a seleção de atributos é descrita como uma busca heurística, na qual cada estado no espaço de busca é composto por um subconjunto de atributos. São descritas, também, três abordagens para a FSS: *embutida*, *filtro* e *wrapper*.

5.2 Seleção de Atributos como Busca Heurística

Intellectuals solve problems; geniuses prevent them.

—Albert Einstein

Segundo Blum & Langley (1997), uma forma conveniente para representar as abordagens para a seleção de atributos — principalmente para aquelas que realizam seleção explícita — é a busca heurística, na qual cada estado no espaço de busca é composto por um subconjunto de possíveis atributos. Desse modo, qualquer método de seleção de atributos pode ser caracterizado por sua instanciação em relação a quatro questões básicas, as quais determinam a natureza do processo de busca heurística. São elas:

Ponto de partida A primeira questão que deve ser tratada é a determinação do ponto de partida (ou pontos de partida) no espaço de busca. Esta determinação, por sua vez, influencia a direção em que a busca será realizada e os operadores que serão utilizados para a geração dos estados sucessores. Na Figura 5.1 na próxima página é ilustrado o espaço de busca para quatro atributos, representados por uma seqüência de quatro círculos (Langley 1996). Pode-se observar que existe uma ordem parcial entre os estados,

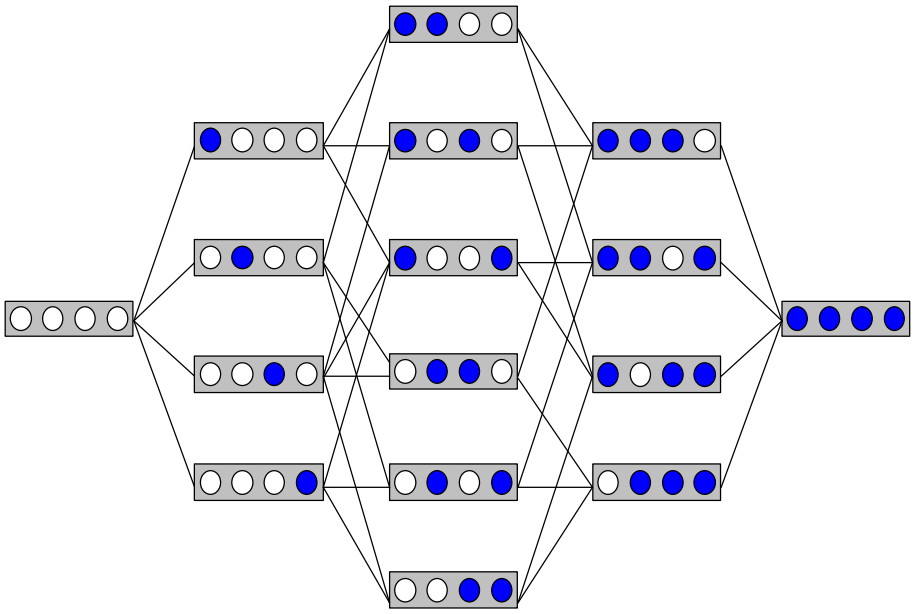


Figura 5.1: Um exemplo de espaço de estados de atributos

pois cada um deles possui um atributo a mais (círculos escuros) que o estado anterior, sendo o estado inicial, mais à esquerda, o conjunto vazio de atributos, representado pelos quatro círculos em branco. Essa abordagem é geralmente conhecida como seleção *forward*. Já a abordagem que inicia o ponto de partida com o subconjunto contendo todos os atributos e sucessivamente removendo-os, é denominada de eliminação *backward*. Também podem ser empregadas variações de ambas abordagens, selecionando-se um estado inicial em algum ponto do espaço de busca e movendo-se a partir desse ponto — seleção *outward*.

Organização da busca A cada ponto na busca, modificações locais no conjunto de atributos são consideradas; uma dessas é selecionada e uma nova iteração é realizada. Claramente, uma busca exaustiva em todo o espaço de estados é impraticável, já que para um número m de atributos existem 2^m possíveis estados. Uma abordagem mais prática é a utilização de um método guloso para a travessia do espaço de busca: em cada ponto da busca, consideram-se alterações locais sobre o conjunto corrente de atributos, seleciona-se um novo atributo e realiza-se uma nova iteração.

Por exemplo, a abordagem *hill-climbing*, conhecida como seleção ou eliminação *stepwise*, considera tanto a adição quanto a remoção de atributos em cada ponto de decisão, permitindo ainda que a recuperação de uma decisão anterior possa ser realizada. Dentre essas opções, pode-se considerar todos os estados gerados e selecionar o melhor, ou simplesmente escolher o primeiro estado que melhora a precisão sobre o conjunto corrente. Pode-se,

também, substituir o método guloso por métodos mais sofisticados, como, por exemplo, o método *best-first*, o qual, embora sendo mais custoso, é ainda computacionalmente viável para alguns domínios.

Estratégia de busca A terceira questão a ser tratada considera a estratégia utilizada na avaliação dos subconjuntos alternativos de atributos. Uma métrica normalmente empregada envolve a habilidade de um atributo discriminar as classes de um conjunto de treinamento. Diversos algoritmos de indução incorporam um critério baseado na Teoria da Informação, enquanto outros medem diretamente a precisão do conjunto de treinamento ou de um conjunto separado de avaliação.

Outro aspecto importante é como a estratégia de FSS interage com o algoritmo básico de indução. Essa interação pode ser subdividida em três abordagens (Kohavi & John 1997): (i) *embutida*, na qual FSS é embutida no algoritmo básico de indução; (ii) *filtro*, na qual FSS é utilizada para filtrar atributos de forma independente do algoritmo de indução a ser utilizado e (iii) *wrapper*, a qual emprega o próprio algoritmo de indução como uma caixa-preta. Essas abordagens são descritas na Seção 5.3.

Critério de parada Alguns critérios mais utilizados são:

- parar de remover ou adicionar atributos quando nenhuma das alternativas melhora o desempenho do classificador;
- continuar gerando subconjuntos de atributos até que um extremo do espaço de busca seja alcançado e escolher o melhor desses subconjuntos;
- ordenar os atributos segundo algum critério e utilizar um parâmetro para determinar o ponto de parada, por exemplo, o número de atributos desejado no subconjunto.

5.3 Abordagens para Seleção de Atributos

> Once you have eliminated the impossible, whatever remains, however improbable, must be the truth.
>
> —*Sir Arthur Conan Doyle, Sherlock Holmes*

Como mencionado anteriormente, as abordagens para a seleção de atributos podem ser agrupadas em: *embutida*, *filtro* e *wrapper* descritas a seguir (Baranauskas & Monard 1998a).

5.3.1 Embutida

Alguns indutores são capazes de realizar sua própria seleção de atributos de forma dinâmica, enquanto procuram por uma hipótese. De fato, FSS é uma parte integral desses indutores.

Em geral, a maioria dos algoritmos preguiçosos possuem uma abordagem embutida para a seleção de atributos. Por exemplo, métodos de particionamento recursivo, tais como árvores de decisão, efetuam uma busca gulosa através do espaço de árvores. A cada passo, eles usam uma função de avaliação para selecionar o atributo que tem a melhor capacidade de discriminar entre as classes. Eles particionam o conjunto de treinamento baseados nesse atributo e repetem o processo para cada subconjunto, estendendo a árvore até que nenhuma discriminação adicional seja possível. Este método é usado pelo indutor C4.5.

Métodos de separar-e-conquistar para indução de regras também possuem seleção embutida de atributos. Estes métodos usam uma função de avaliação para selecionar o atributo que ajuda a distinguir uma classe C das outras; então eles adicionam o teste resultante em uma única regra conjuntiva para essa classe C. Eles repetem esse processo até que a regra exclua todos os exemplos de outras classes e então removem os exemplos da classe C que a regra cobre, repetindo esse processo nos exemplos de treinamento remanescentes. Este método é empregado pelo indutor CN2.

5.3.2 Filtro

Essa abordagem de seleção de atributos introduz um processo separado, o qual ocorre antes da aplicação do algoritmo de indução propriamente dito, como é mostrado na Figura 5.2 na próxima página. A idéia é filtrar atributos irrelevantes, segundo algum critério, antes de iniciar a indução (John, Kohavi & Pfleger 1994). Esse passo de pré-processamento considera características gerais do conjunto de exemplos para selecionar alguns atributos e excluir outros. Sendo assim, métodos de filtros são independentes do algoritmo de indução que, simplesmente, receberá como entrada o conjunto de exemplos contendo apenas os atributos selecionados pelo filtro.

Segundo Blum & Langley (1997), um dos esquemas mais simples de filtragem é a avaliação de cada atributo individualmente, baseada na sua correlação com a classe, escolhendo os l atributos que fornecem o melhor valor. Esse método é comumente empregado em tarefas de categorização de textos, geralmente combinando filtro com algoritmos *naïve* Bayes (Mitchell 1998) ou K-NN, os quais têm mostrado bons resultados empíricos.

Um ponto principal dessa abordagem é que os filtros ignoram totalmente os efeitos do sub-

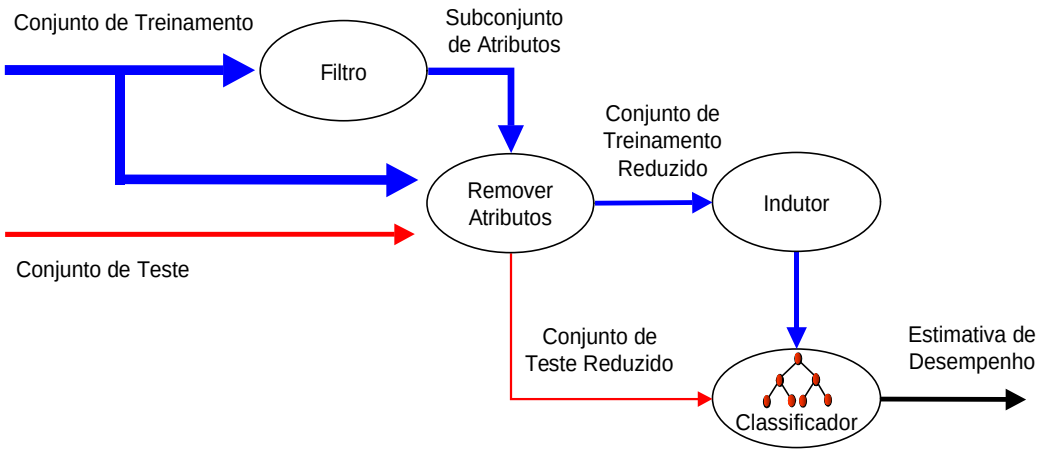


Figura 5.2: Abordagem filtro

conjunto de atributos no desempenho do indutor ao qual os dados reduzidos serão submetidos. Por esse motivo, Kohavi & John (1997) propõem que filtros sejam substituídos pela abordagem *wrapper*, a qual considera o próprio indutor no processo de seleção.

A seguir alguns algoritmos e técnicas que se situam dentro da abordagem de seleção de atributos por filtros são reportados.

Algoritmos de Filtragem

Qualquer algoritmo que efetue algum tipo de seleção pode ser usado para filtrar atributos. A saída do algoritmo de filtragem é o conjunto de atributos por ele selecionados. Os atributos restantes são removidos do conjunto de exemplos, reduzindo assim sua dimensão. Após isso, o conjunto de exemplos reduzido pode ser usado por qualquer indutor. Entretanto, atributos considerados como bons por um filtro não são, necessariamente, úteis para outras famílias de algoritmos que podem ter um *bias* de aprendizado diferente.

Por exemplo, um algoritmo de indução de árvores de decisão pode ser usado como um filtro de atributos. O conjunto de atributos selecionado pela árvore constitui a saída do processo de filtragem, sendo a árvore descartada. Outros algoritmos incluem a seleção utilizando K-NN (Cardie 1993), seleção assumindo atributos independentes, seleção baseada em distância entre outros (Weiss & Indurkhya 1998). Existem dois algoritmos, especificamente desenvolvidos para atuarem com filtros de atributos, FOCUS e RELIEF que são descritos a seguir.

FOCUS

Uma abordagem filtro utilizando o algoritmo FOCUS envolve um grau maior de busca no espaço de atributos (Almuallim & Dietterich 1991; Almuallim & Dietterich 1997). Esse algoritmo, inicialmente proposto para domínios booleanos sem ruído, procura exaustivamente pela combinação mínima de atributos que seja suficiente para descrever a classe de todos os exemplos de treinamento. Desse modo, esse método inicia a busca examinando cada atributo em separado, depois examina pares de atributos, triplas e assim por diante, até encontrar uma combinação que gera as melhores partições do conjunto de treinamento.

RELIEF

O algoritmo RELIEF incorpora uma função de avaliação de atributos mais complexa que o algoritmo FOCUS. RELIEF atribui uma relevância ponderada para cada atributo, que é definida para denotar a relevância do atributo em relação às classes (Kira & Rendell 1992a; Kira & Rendell 1992b; Kononenko 1994). RELIEF é um algoritmo que usa amostras aleatórias dos exemplos e atualiza os valores de relevância baseado na diferença entre o exemplo selecionado e os dois exemplos mais próximos da mesma classe e de outra classe.

A principal diferença entre RELIEF e FOCUS é que, enquanto FOCUS procura por um conjunto mínimo de atributos, o algoritmo RELIEF procura por todos os atributos relevantes.

5.3.3 Wrapper

Em contraste com filtros, a abordagem *wrapper* gera um subconjunto de atributos como candidato, executa o indutor com apenas esses atributos no conjunto de treinamento e usa a precisão do classificador extraído para avaliar o subconjunto de atributos em questão. Este processo é repetido para cada subconjunto candidato, até que o critério de parada seja satisfeito.

A idéia geral por trás da abordagem *wrapper* é mostrada na Figura 5.3 na página oposta. O algoritmo de FSS existe como um *wrapper* ao redor do indutor e é responsável por conduzir a busca por um bom subconjunto de atributos. A qualidade de um subconjunto candidato é avaliada utilizando o próprio indutor como uma caixa-preta. O objetivo da busca é encontrar o subconjunto (nó) com a melhor qualidade, utilizando uma função heurística para guiá-la.

No *wrapper* existente na biblioteca $\mathcal{MLC}$++, utilizado nos experimentos apresentados neste trabalho, a busca é conduzida no espaço do subconjunto de atributos, com os operadores adicionar ou remover, utilizando como busca o método *hill-climbing* ou o método *best-first* assim com direções *forward* e *backward* como direção da busca. Na $\mathcal{MLC}$++, *cross-validation* é o método

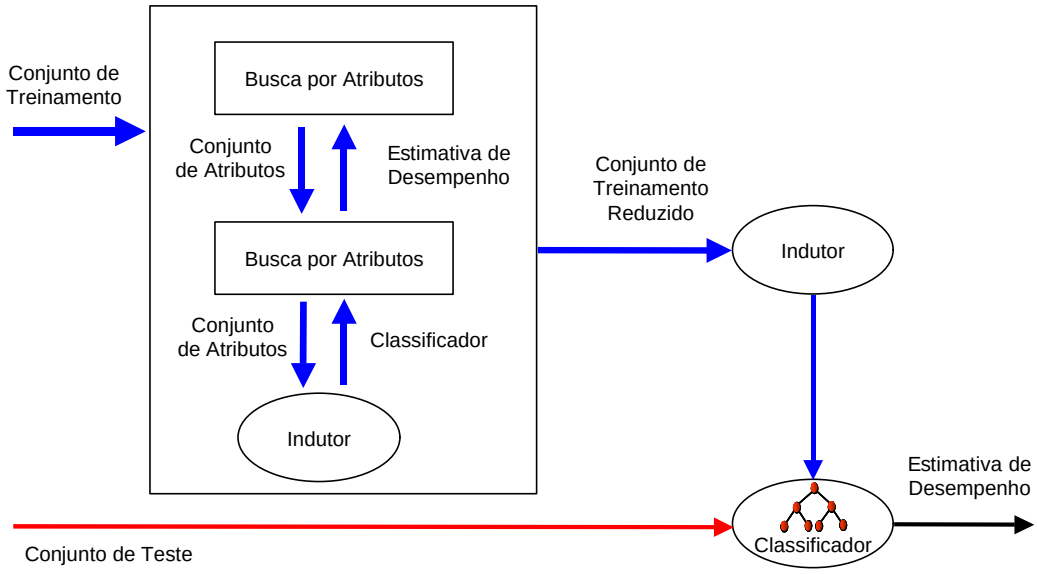


Figura 5.3: Abordagem *wrapper*

default usado para estimar a precisão de um nó (Kohavi & Sommerfield 1995).

Um argumento a favor da abordagem *wrapper* é que o mesmo algoritmo de indução que vai utilizar o subconjunto de atributos selecionado deve prover uma estimativa melhor de precisão que um outro algoritmo, o qual pode possuir um *bias* de indução totalmente diferente (Kohavi & John 1997). Por outro lado, como mostrado neste trabalho, essa abordagem pode ser computacionalmente dispendiosa, uma vez que o indutor deve ser executado para cada subconjunto de atributos considerado.

5.4 Metodologia Experimental

Vivere est cogitare (To think is to live).

—*Marcus Tullius Cicero*

De forma a avaliar filtros e *wrappers* para seleção de atributos, alguns experimentos foram por nós conduzidos, utilizando onze conjuntos de exemplos: bupa, pima, breast-cancer, hungaria, crx, letter, hepatitis, anneal, sonar, genetics e dna, já descritos na Seção 3.3 na página 51. O algoritmo de FSS *wrapper* utilizado encontra-se na biblioteca $\mathcal{MLC}$++ assim como o indutor ID3 usado como filtro. O outro filtro utilizado, $\mathcal{CI}$, encontra-se na ferramenta MineSet™. Os conjuntos de exemplos não foram pré-processados de qualquer maneira, por exemplo, pela remoção ou substituição de valores desconhecidos ou pela transformação de atributos. O nosso objetivo foi avaliar o desempenho dos quatro indutores com diferentes *biases* de aprendizado

C4.5, CN2, IB e NB utilizando seleção de atributos, comparando-os a si próprios sem utilizar seleção de atributos.

5.4.1 Filtros

Considerando cada conjunto de exemplos, todos os exemplos foram submetidos aos algoritmos ID3 (da biblioteca $\mathcal{MLC}$++) e $\mathcal{CI}$ usados como filtros para FSS. Ambos os algoritmos foram executados com seus parâmetros *defaults*, ou seja, sem ajustes.

A saída do indutor ID3 é simplesmente o subconjunto de atributos selecionados pela árvore; a própria árvore é descartada. Para propósitos de filtragem, ID3 foi usado na sua forma mais simples para obter a árvore de cobertura utilizando todos os exemplos, ou seja, sem poda e sem estimativa de erro em um conjunto de teste.

Assim como no filtro ID3, todos os exemplos foram utilizados por $\mathcal{CI}$ (*default* da ferramenta) para encontrar atributos relevantes. O algoritmo $\mathcal{CI}$ permite determinar o máximo de l ($l \leq m$) atributos relevantes na discriminação de classes. Nos experimentos efetuados usou-se $l = m$, o que significa que todos os atributos no conjunto de exemplos poderiam ser selecionados por $\mathcal{CI}$. Os atributos presentes na árvore induzida por ID3 bem como o conjunto de atributos selecionado por $\mathcal{CI}$ foram usados na etapa de estimativa de precisão.

Em ambos os filtros considerados, o ponto de partida é o conjunto vazio de atributos utilizando seleção *forward* com o operador de adicionar atributos. A precisão estimada de cada nó é o *bias* dos indutores ID3 e $\mathcal{CI}$ baseados em entropia. O método de busca é guloso. O critério de parada é quando nenhuma melhoria é possível com maior particionamento dos dados.

Para indicar o experimento ao qual está-se referindo, a notação FSS(f,*indutor*) é usada, onde 'f' indica seleção de atributos utilizando filtro e *indutor* $\in$ {ID3, $\mathcal{CI}$ } indica o indutor sendo usado como filtro.

5.4.2 Wrappers

Para cada conjunto de exemplos, quatro indutores — C4.5, CN2, IB e NB— foram aplicados através do *wrapper* como indutores caixa-preta. Esses indutores foram também executados com seus parâmetros *defaults*.

O *wrapper* $\mathcal{MLC}$++ foi executado utilizando busca *best-first* (uma vez que é um método mais robusto que *hill-climbing*), operadores compostos, assim como seleção *forward* e *backward*, obtendo para cada conjunto de exemplos e indutor os melhores atributos selecionados. Todos os demais parâmetros do *wrapper* $\mathcal{MLC}$++ foram usados com seus valores *defaults*, ou seja, a precisão estimada para cada nó é computada utilizando 10-*fold cross-validation*; o critério

de parada da busca é uma seqüência consecutiva de expansões de cinco nós que não geram um subconjunto de atributos apresentando uma melhoria de 0,1% na precisão em relação ao subconjunto anterior e, finalmente, um atributo é selecionado se ele não prejudica a precisão, ou seja, se a precisão é a mesma com ou sem aquele atributo. O conjunto de atributos selecionados pelos *wrappers* foi então utilizado na etapa de estimativa de precisão.

Novamente, de forma a indicar o experimento *wrapper* ao qual está-se referindo, a notação FSS(*método,indutor*) é usada, onde *método* ∈ {wf, wb}, indicando se *wrapper forward* (wf) ou *wrapper backward* (wb) foi usado e *indutor* ∈ {C4.5, CN2, IB, NB} indicando o indutor que foi usado como caixa-preta para o *wrapper*.

5.4.3 Estimativa de Precisão

De posse dos atributos selecionados pelos filtros e *wrappers*, cada um dos quatro indutores C4.5, CN2, IB e NB foi aplicado ao correspondente conjunto de exemplos reduzido e a precisão foi medida utilizando 10-*fold stratified cross-validation*. Para uma base de comparação, o algoritmo padrão (sem FSS) foi aplicado ao conjunto original de exemplos e a precisão também foi medida utilizando 10-*fold stratified cross-validation*.

Como a abordagem *wrapper* é muito lenta para os conjuntos de exemplos com um número grande de atributos, e devido às limitações dos recursos computacionais, o *wrapper* foi executado apenas uma única vez utilizando todos os exemplos, ou seja, sem a divisão em conjuntos de treinamento e teste. Analogamente, o mesmo processo foi efetuado com os filtros de forma que comparações entre *wrappers* e filtros fossem efetuadas sob as mesmas condições. Portanto, isto significa que os resultados da precisão para *wrappers* e filtros podem ser otimistas, mas não aqueles utilizando todos os atributos.

5.5 Resultados

> To be able to discern that what is true is true, and that what is false is false; this is the mark and character of intelligence.).
>
> —*Emanuel Swedenborg*

Nos resultados que se seguem não encontram-se disponíveis os valores para o conjunto de exemplos dna utilizando os indutores CN2 e IB com seleção *wrapper backward*, uma vez que após 40 dias em execução eles ainda não haviam terminado. Esses resultados são reportados como 'N/A' nas tabelas e não foram considerados no cálculo das médias envolvidas. Nesta seção, é

apresentado apenas um resumo dos resultados mais significantes. Um relatório detalhado dos resultados obtidos, incluindo os atributos selecionados em cada conjunto de exemplos, além de comparações adicionais encontra-se em (Baranauskas & Monard 1999).

5.5.1 Proporção de Atributos Selecionados

Na Tabela 5.1 é reportada a proporção de atributos selecionados para cada um dos 108 casos considerados, na qual #A indica o número de atributos no conjunto original de exemplos. É possível notar que o espaço de atributos foi reduzido, exceto para FSS(f,ID3) com os conjuntos de exemplos bupa, pima e letter, nos quais todos os atributos foram selecionados por esse filtro.

Dataset	#A	FSS									
		(wf,C4.5)	(wb,C4.5)	(wf,CN2)	(wb,CN2)	(wf,IB)	(wb,IB)	(wf,NB)	(wb,NB)	(f,ID3)	(f,$\mathcal{CI}$)
bupa	6	83,33	83,33	83,33	83,33	83,33	83,33	83,33	83,33	100,00	16,67
pima	8	62,50	62,50	87,50	87,50	37,50	62,50	62,50	62,50	100,00	75,00
breast cancer	10	70,00	70,00	50,00	90,00	60,00	60,00	50,00	70,00	80,00	90,00
hungaria	13	38,46	61,54	30,77	53,85	15,38	69,23	61,54	69,23	84,62	84,62
crx	15	46,67	73,33	33,33	46,67	33,33	86,67	40,00	46,67	80,00	86,67
letter	16	68,75	68,75	56,25	56,25	68,75	68,75	75,00	75,00	100,00	75,00
hepatitis	19	26,32	36,84	36,84	84,21	26,32	78,95	47,37	57,89	47,37	52,63
anneal	38	73,68	92,11	60,53	89,47	31,58	47,37	28,95	89,47	26,32	18,42
sonar	60	11,67	66,67	20,00	83,33	35,00	81,67	16,67	83,33	23,33	35,00
genetics	60	23,33	81,67	8,33	80,00	8,33	70,00	41,67	86,67	75,00	86,67
dna	180	13,89	59,44	8,33	N/A	5,56	N/A	22,22	92,22	50,56	47,22
Média	38,64	47,15	68,74	43,20	75,46	36,83	70,85	48,11	74,21	69,74	60,72

Tabela 5.1: Proporção de atributos selecionados

Considerando *wrappers*, em todos os casos, não apenas na média, o número de atributos selecionado utilizando seleção *backward* é sempre maior ou igual do que utilizando seleção *forward*, ou seja, #FSS(wb,*indutor*) $\geq$ #FSS(wf,*indutor*). Em média, a seleção *forward* escolheu 43,82% atributos contra 72,28% na seleção *backward*, um aumento de 64,93%. Este resultado parece confirmar a idéia de que, iniciando com o conjunto completo de atributos e removendo-os, favorece capturar a interação entre atributos (Kohavi & John 1997). Além disso, em média, os filtros selecionaram 69,74% e 60,72% dos atributos (para FSS(f,ID3) e FSS(f,$\mathcal{CI}$), respectivamente), que é um valor intermediário entre aqueles obtidos pelos *wrapper forward* e *wrapper backward*.

Deve ser notado que nos experimentos realizados, para alguns conjuntos de exemplos, um número de atributos maior do que o necessário para atingir o critério de parada foi selecionado pelo *wrapper*. Isto é devido ao fato que foi utilizado o valor *default* $\mathcal{MLC}$++ igual a zero para o parâmetro *penalidade de complexidade*. Este parâmetro permite penalizar subconjuntos de atributos com muitos atributos tais que, se a precisão de dois subconjuntos é a mesma, o subconjunto com o menor número de atributos é escolhido. Por exemplo, há sete atributos nominais assumindo um único valor no conjunto de exemplos anneal. Todos eles foram selecionados por

seleção *backward* e seis por seleção *forward* para o indutor C4.5, mesmo que eles não promovam qualquer melhoria na precisão. Os outros indutores também selecionaram mais de um desses atributos. Entretanto, no máximo, um desses atributos seria selecionado, se o parâmetro de penalidade de complexidade utilizado fosse maior que zero.

De fato, não somente para a abordagem *wrapper*, mas também para ambos filtros estudados (ID3 sem poda e permitindo $\mathcal{CI}$ selecionar até todos atributos presentes no conjunto de exemplos), optou-se por escolher subconjuntos maiores de atributos. A razão disso é que os indutores podem lidar com atributos extras, mas não podem compensar atributos que foram descartados.

5.5.2 Tempo de Execução

O tempo de execução, em segundos, para selecionar os atributos é mostrado na Tabela 5.2. Espera-se que a abordagem *wrapper forward* seja menos dispendiosa que *wrapper backward*, pois induzir classificadores quando há poucos atributos no conjunto de treinamento deve ser computacionalmente mais rápido. Obviamente, os filtros são muitíssimo mais rápidos que *wrappers*.

Dataset	FSS									
	(wf,C4.5)	(wb,C4.5)	(wf,CN2)	(wb,CN2)	(wf,IB)	(wb,IB)	(wf,NB)	(wb,NB)	(f,ID3)	(f,$\mathcal{CI}$)
bupa	28,7	23,7	189,7	164,1	80,1	47,7	8,8	6,5	0,9	0,1
pima	81,9	89,2	1292,1	790,7	335,1	357,0	27,2	46,4	2,1	0,4
breast cancer	135,8	116,7	697,7	564,3	537,1	566,7	33,2	37,7	1,1	0,5
hungaria	83,6	104,8	314,2	1242,9	112,2	185,0	23,1	20,7	0,9	0,2
crx	416,5	324,8	464,4	3628,7	353,8	544,7	46,4	67,4	1,8	0,7
letter	2003,5	1167,9	33446,1	68115,1	53490,4	74728,6	784,2	865,3	58,5	27,0
hepatitis	77,2	149,6	700,4	583,0	112,6	136,9	17,4	26,0	0,6	0,2
anneal	3721,0	3707,4	87607,7	71581,7	4138,5	6092,6	187,1	263,0	2,0	0,8
sonar	569,2	3968,4	5726,9	28153,0	2161,3	3751,7	126,7	768,7	2,5	1,6
genetics	8546,3	24841,0	42479,4	550329,7	36520,4	638657,0	2279,7	2026,6	2,5	51,8
dna	47896,5	275187,0	349135,5	N/A	225645,4	N/A	7966,5	18773,6	31,6	240,5
Média	1566,4	3449,4	17291,9	72515,3	9784,2	72506,8	353,4	412,8	7,3	8,3

Tabela 5.2: Tempo (em segundos) para selecionar atributos

Para *wrappers*, embora não para todos os casos individuais, o cenário geral indica que seleção *backward* é cerca de 5 vezes mais demorada que a seleção *forward*. Por outro lado, em média, os filtros são cerca de 10^3 vezes mais rápidos que *wrappers*.

Em todo caso, a abordagem *wrapper* é muito lenta. O tempo total necessário para executar todos os experimentos foi maior que 32 dias de processamento ininterrupto, para todos os indutores e conjuntos de exemplos (exceto FSS(wb,CN2) e FSS(wb,IB) no conjunto dna). Em média, *wrapper* utilizando o indutor NB foi o mais rápido seguido por C4.5, IB e CN2, respectivamente.

É também possível notar que, para a maioria dos conjuntos de exemplos, a seleção *backward* tomou mais tempo que a seleção *forward* para IB e NB mas não para C4.5 e CN2. Observa-se que apenas esses últimos indutores são capazes de selecionar atributos por si mesmos (FSS

embutida), além de realizar o tratamento de valores desconhecidos.

5.5.3 Comparação

A seguir, são mostradas comparações entre o indutor padrão (sem FSS) com sua correspondente seleção de atributos utilizando *wrapper forward*, *wrapper backward* e ambos os filtros ID3 e $\mathcal{CI}$. Na Tabela 5.3 na página oposta é mostrada a precisão (média e desvio padrão dos 10–*fold stratified cross-validation*) para cada indutor utilizando todos os atributos e utilizando apenas os atributos selecionados por ambos *wrappers* e ambos filtros. Para determinar se a diferença entre eles, ou seja, (FSS(wf,*indutor*)−*indutor*), (FSS(wb,*indutor*)−*indutor*), (FSS(f,ID3)−*indutor*) e (FSS(f,$\mathcal{CI}$)−*indutor*) é significante ou não, é mostrado na Figura 5.4 na página 90, para cada *indutor*, um gráfico com quatro barras consecutivas para cada conjunto de exemplos. Cada barra corresponde à média dividida pelo desvio padrão da precisão, calculada por (3.6) definida na página 51. Qualquer barra com altura maior que dois indica que o resultado é significativo, com nível de confiança de 95%. Quando a barra encontra-se acima de zero significa que o primeiro algoritmo supera o segundo; se a barra encontra-se abaixo então o segundo algoritmo supera o primeiro. Quando a altura da barra estiver acima (abaixo) de dois significa que o primeiro (segundo) algoritmo supera significativamente o segundo (primeiro).

Embora a qualidade dos resultados varie entre os conjuntos de exemplos, em geral a abordagem *wrapper* supera o indutor padrão. Na média geral, a precisão do indutor padrão de 81,40% melhorou para 85,61% utilizando seleção *forward* e para 84,35% utilizando seleção *backward*. Isto representa uma redução relativa na taxa de erro de 22,62% e 15,85%, respectivamente. Entretanto, é importante salientar que a média sobre conjuntos de exemplos diferentes não são muito significativas.

Considerando C4.5, a abordagem *wrapper* melhorou a precisão sensivelmente em relação ao indutor padrão, exceto para o conjunto de exemplos bupa.

Para IB, a abordagem *wrapper* supera o indutor padrão para todos os conjuntos de exemplos. A melhoria utilizando seleção *forward* para o conjunto de exemplos genetics foi de 78,75% para 90,88%, que corresponde a uma redução relativa na taxa de erro de 57,08%, utilizando apenas 8,33% do total de atributos.

Para ambos filtros, embora, em média, eles não superem os indutores padrões, eles não prejudicaram muito a precisão. Por exemplo, considerando a média geral, a precisão do indutor padrão diminuiu de 81,40% para 81,12% utilizando o filtro ID3 e para 80,60% utilizando o filtro $\mathcal{CI}$, representando um aumento relativo na taxa de erro de 1,52% e 4,30%, respectivamente, mostrando que, em geral, para os conjuntos de exemplos considerados, o filtro ID3 supera o filtro

Dataset	C4.5	FSS(wf,C4.5)	FSS(wb,C4.5)	FSS(f,ID3)	FSS(f,$\mathcal{CI}$)
bupa	68,71±1,73	66,97±2,76	66,97±2,76	68,71±1,73	60,92±2,10
pima	74,26±1,13	74,77±1,04	75,95±0,98	74,26±1,13	72,82±0,73
breast cancer	94,28±0,56	95,00±0,83	95,00±0,83	95,13±0,65	94,57±0,70
hungaria	77,52±4,20	82,97±3,27	82,97±3,27	77,16±3,56	79,57±3,57
crx	84,35±1,18	85,65±1,17	86,67±1,26	84,64±1,08	84,35±1,60
letter	86,59±0,34	86,93±0,31	86,93±0,31	86,59±0,34	86,87±0,35
hepatitis	79,38±2,27	84,50±2,00	88,38±1,62	79,88±2,89	80,58±2,39
anneal	92,54±1,18	92,55±1,24	94,21±0,86	89,09±0,79	90,99±1,43
sonar	69,74±1,97	83,19±2,30	83,74±3,75	80,24±2,56	74,95±5,01
genetics	94,17±0,39	94,67±0,30	94,17±0,39	94,39±0,34	93,17±0,29
dna	92,40±0,46	95,45±0,38	94,51±0,39	94,04±0,44	92,78±0,48
Média	83,09	85,70	86,32	84,01	82,87

Dataset	CN2	FSS(wf,CN2)	FSS(wb,CN2)	FSS(f,ID3)	FSS(f,$\mathcal{CI}$)
bupa	67,82±2,11	65,81±1,83	65,81±1,83	67,82±2,11	55,63±2,40
pima	74,62±1,38	74,75±1,43	74,75±1,43	74,62±1,38	74,09±1,02
breast cancer	94,99±1,42	96,85±0,60	95,57±0,97	94,70±0,86	94,28±1,08
hungaria	77,93±3,06	83,66±2,60	80,65±3,94	79,98±2,79	80,59±1,94
crx	83,20±1,21	86,83±1,23	85,83±0,86	83,34±1,19	83,35±1,35
letter	70,34±0,30	76,17±0,20	76,17±0,20	70,34±0,30	71,56±0,33
hepatitis	81,75±3,83	90,99±1,66	87,13±2,81	82,95±3,49	76,69±2,39
anneal	90,42±1,08	96,44±0,94	96,44±0,94	85,18±0,60	82,85±1,34
sonar	71,19±3,30	81,32±3,46	75,52±2,75	72,59±3,20	72,12±2,73
genetics	79,53±1,45	89,47±0,53	83,26±1,73	78,57±2,13	76,27±1,96
dna	88,15±0,62	94,89±0,41	N/A	89,68±0,53	88,77±0,60
Média	79,99	85,20	82,11	79,98	77,84

Dataset	IB	FSS(wf,IB)	FSS(wb,IB)	FSS(f,ID3)	FSS(f,$\mathcal{CI}$)
bupa	61,66±2,82	65,82±1,68	65,82±1,68	61,66±2,82	55,66±1,71
pima	69,18±1,92	72,56±1,44	70,87±1,43	69,18±1,92	68,53±1,63
breast cancer	95,00±0,88	96,57±0,80	96,57±0,57	95,86±0,86	95,00±0,77
hungaria	78,92±1,93	83,64±3,02	82,00±2,56	79,61±1,73	81,66±2,00
crx	81,74±0,90	87,39±1,18	83,48±1,11	82,90±1,14	80,29±1,19
letter	95,17±0,13	96,19±0,13	96,19±0,13	95,17±0,13	95,69±0,11
hepatitis	81,29±2,23	87,13±2,67	86,54±2,17	76,75±2,57	76,71±2,26
anneal	99,33±0,25	99,78±0,15	99,67±0,17	98,55±0,33	99,78±0,15
sonar	85,60±2,00	92,81±1,27	89,43±1,99	84,67±2,09	86,12±2,30
genetics	78,75±0,71	90,88±0,62	83,13±0,68	80,44±0,85	78,97±0,61
dna	74,23±0,44	93,72±0,34	N/A	78,12±0,70	75,99±0,64
Média	81,90	87,86	85,37	82,08	81,31

Dataset	NB	FSS(wf,NB)	FSS(wb,NB)	FSS(f,ID3)	FSS(f,$\mathcal{CI}$)
bupa	55,44±2,95	60,90±2,32	60,90±2,32	55,44±2,95	56,26±1,64
pima	75,55±1,05	77,25±0,99	77,25±0,99	75,55±1,05	75,69±1,31
breast cancer	96,14±0,68	95,85±0,84	96,42±0,53	96,14±0,68	96,14±0,68
hungaria	83,67±2,83	85,68±2,27	84,36±2,76	84,00±2,80	83,67±2,83
crx	77,97±1,12	87,10±1,25	87,39±1,18	76,81±0,89	75,94±1,02
letter	64,55±0,36	66,31±0,43	66,31±0,43	64,55±0,36	65,81±0,38
hepatitis	83,08±3,72	87,08±2,39	88,29±1,93	82,50±2,96	86,37±3,39
anneal	91,65±0,71	88,42±0,60	91,87±0,66	70,93±1,60	85,30±0,88
sonar	69,26±4,53	78,74±3,24	74,07±3,61	65,90±1,86	69,74±2,32
genetics	95,45±0,32	96,21±0,26	96,11±0,23	95,80±0,28	95,20±0,34
dna	94,04±0,34	96,83±0,23	95,29±0,43	94,66±0,25	94,04±0,60
Média	80,62	83,67	83,48	78,39	80,38
Média Geral	81,40	85,61	84,35	81,12	80,60

Tabela 5.3: Precisão sem FSS, com *wrapper forward*, *wrapper backward*, filtro ID3 e filtro $\mathcal{CI}$

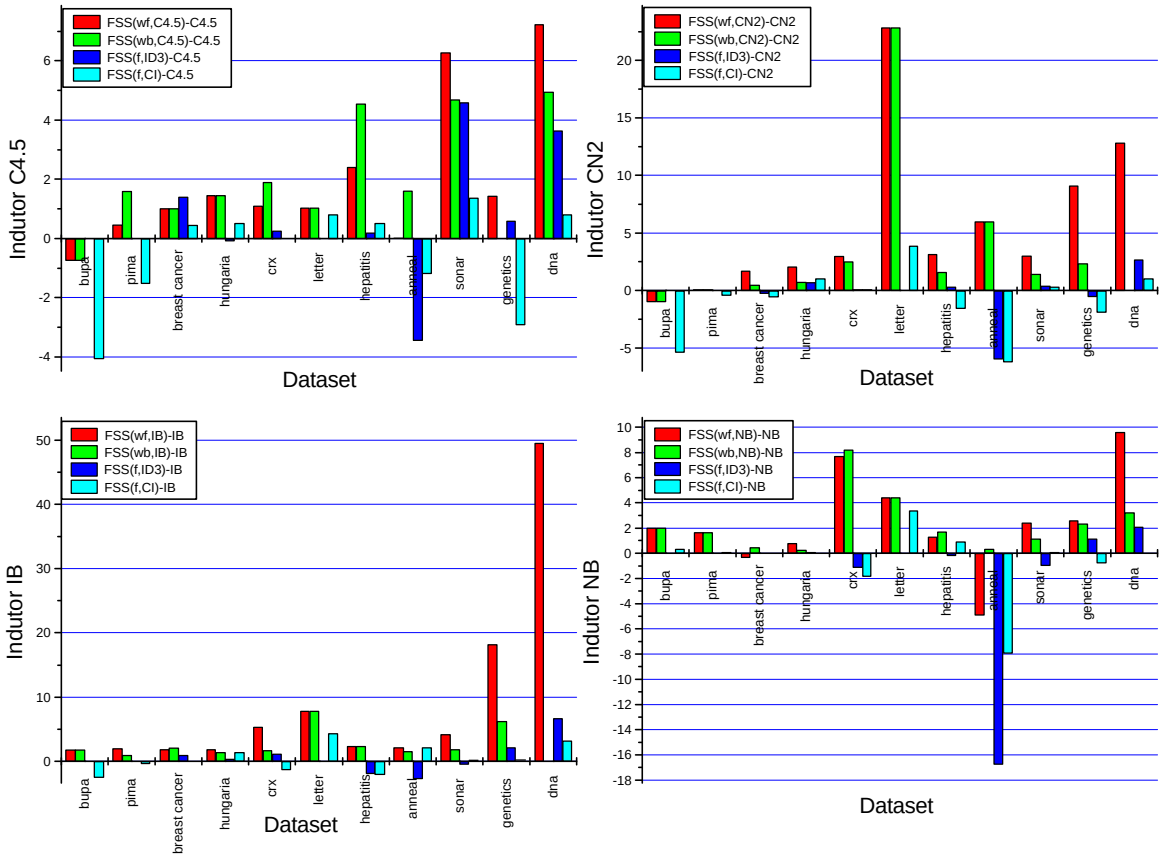


Figura 5.4: Diferença absoluta (em desvios padrões) da precisão

$\mathcal{CI}$.

Desconsiderando o conjunto de exemplos anneal e uma vez que o *bias* de aprendizado de ID3 e C4.5 são similares, em média, a precisão de C4.5 aumentou utilizando FSS(f,ID3). Note que para CN2 e IB não houve danos na precisão média utilizando FSS(f,ID3), mesmo considerando que eles não tenham o mesmo *bias* de aprendizado que ID3. Para NB, FSS(f,$\mathcal{CI}$) aparenta ser um filtro mais robusto do que FSS(f,ID3).

Para o conjunto de exemplos dna ambos *wrappers* assim como o filtro ID3 superam significativamente o indutor padrão. De fato, FSS(wf,NB) obteve a melhor precisão de 96,83% para o conjunto de exemplos dna utilizando apenas 22,22% dos atributos. Com uma penalidade de complexidade maior do que zero, a precisão de 94,60% utilizando apenas 7,78% de atributos é reportada por Kohavi & John (1997). De fato, nenhum outro indutor conhecido supera NB nesse conjunto de exemplos, o qual inclui muitos atributos irrelevantes.

Dataset	FSS(wf,C4.5) −C4.5	FSS(wf,CN2) −CN2	FSS(wf,IB) −IB	FSS(wf,NB) −NB	FSS(wb,C4.5) −C4.5	FSS(wb,CN2) −CN2	FSS(wb,IB) −IB	FSS(wb,NB) −NB	Soma △	Soma ▽
bupa				△				△	2	0
pima									0	0
breast cancer							△		1	0
hungaria		△							1	0
crx		△	△	△		△		△	5	0
letter		△	△	△		△	△	△	6	0
hepatitis	△	△	△		△		△		5	0
anneal		△	△	▽		△			3	1
sonar	△	△	△	△	△				5	0
genetics		△	△	△		△	△	△	6	0
dna	△	△	△	△	△	N/A	N/A	△	6	0
Soma △	3	8	7	6	3	4	4	5	40	
Soma ▽	0	0	0	1	0	0	0	0		1

Dataset	FSS(f,ID3) −C4.5	FSS(f,ID3) −CN2	FSS(f,ID3) −IB	FSS(f,ID3) −NB	FSS(f,$\mathcal{CI}$) −C4.5	FSS(f,$\mathcal{CI}$) −CN2	FSS(f,$\mathcal{CI}$) −IB	FSS(f,$\mathcal{CI}$) −NB	Soma △	Soma ▽
bupa					▽	▽	▽		0	3
pima									0	0
breast cancer									0	0
hungaria									0	0
crx									0	0
letter						△	△	△	3	0
hepatitis							▽		0	1
anneal	▽	▽	▽	▽		▽	△	▽	1	6
sonar	△								1	0
genetics			△		▽				1	1
dna	△	△	△	△			△		5	0
Soma △	2	1	2	1	0	1	3	1	11	
Soma ▽	1	1	1	1	2	2	2	1		11

Tabela 5.4: Precisões com nível de significância: melhorias acima de dois desvios padrões são reportadas com △ e aquelas abaixo com ▽

A Tabela 5.4 mostra a melhoria significativa na precisão para *wrappers* e filtros comparados com o indutor padrão. Melhorias acima de dois desvios padrões são reportadas com △ e aquelas abaixo com ▽. Para *wrappers*, dentre os indutores utilizados, C4.5 mostra o mínimo de melhorias significantes nos mesmos três conjuntos de exemplos (hepatitis, sonar e dna). De fato, CN2, IB e NB apresentam melhorias significativas um número maior de vezes no *wrapper forward* do que no *wrapper backward*.

O cenário geral mostra que *wrappers* para FSS superam significativamente o indutor padrão em 24 dos 44 experimentos para seleção *forward* e em 16 dos 42 experimentos para seleção *backward* para todos os domínios e indutores estudados. Nota-se que o indutor padrão supera significativamente apenas uma única vez FSS(wf,NB) no conjunto de exemplos anneal.

Também os filtros ID3 e $\mathcal{CI}$ superaram significativamente o indutor padrão em 6 e 5 dos 44 experimentos, respectivamente. Entretanto, o indutor padrão supera significativamente os filtros em 11 experimentos (4 para ID3 e 7 para $\mathcal{CI}$), sendo que em 6 deles está envolvido o conjunto de exemplos anneal.

Como esperado, os conjuntos de exemplos com muitos atributos irrelevantes, tais como dna, aparentam se beneficiar mais de FSS utilizando *wrappers* e filtros do que aqueles conjuntos de exemplos com menos atributos.

5.6 Impacto no Classificador CN2

Give me a place to stand, and I will move the Earth.

—*Archimedes*

Embora a precisão seja um fator importante, no caso de Aprendizado de Máquina simbólico as regras obtidas também constituem um fator relevante. Assim, é igualmente importante analisar a complexidade do classificador simbólico resultante sem e com o uso de FSS.

Com esse objetivo, em experimentos complementares (Baranauskas, Monard & Horst 1999a), o subconjunto de atributos selecionado pelos *wrappers*, utilizando CN2 como caixa-preta, e filtros ID3 e $\mathcal{CI}$, foi fornecido ao indutor CN2. O classificador induzido, para cada subconjunto de atributos, foi então avaliado quanto ao número de regras, número de exemplos cobertos e número de condições nas regras. Para uma base de comparação, o conjunto de exemplos original, contendo todos os atributos, também foi fornecido para CN2; o classificador induzido também foi avaliado sob as mesmas condições.

Na Tabela 5.5 é mostrado, na coluna 'CN2', o número de regras induzidas pelo algoritmo CN2 utilizando o conjunto original de exemplos (sem seleção de atributos). Nas colunas FSS(wf,CN2), FSS(wb,CN2), FSS(f,ID3) e FSS(f,$\mathcal{CI}$) é mostrada a mesma informação, mas executando o algoritmo CN2 somente no conjunto reduzido de exemplos pelos *wrappers* e filtros, ou seja, apenas os atributos selecionados no conjunto de exemplos foram fornecidos para CN2. Essa notação se estende para as demais tabelas e gráficos desta seção.

Dataset	CN2	FSS(wf,CN2)	FSS(wb,CN2)	FSS(f,ID3)	FSS(f,$\mathcal{CI}$)
bupa	34	*33	*33	34	41
pima	44	56	56	44	54
breast cancer	14	21	15	15	15
hungaria	22	34	27	27	30
crx	35	*15	47	37	36
letter	242	313	313	242	*226
hepatitis	15	21	*14	*14	20
anneal	36	49	48	85	90
sonar	28	*22	*26	*23	*18
genetics	64	110	*56	64	83
dna	130	*105	N/A	143	153
Média	60,36	70,82	63,50	66,18	69,64

Tabela 5.5: Número de regras induzidas por CN2; valores marcados com * indicam diminuição do número de regras em relação ao algoritmo sem utilizar FSS

Como é possível notar, em média, o número de regras aumentou de 60,36 para 70,82 utilizando FSS(wf,CN2), para 63,50 utilizando FSS(wb,CN2), para 66,18 utilizando FSS(f,ID3) e

para 69,64 utilizando FSS(f,$\mathcal{CI}$). Isto significa um aumento relativo de 17,32%, 5,20%, 9,64% e 15,36% no número médio de regras induzidas, respectivamente.

Observando os resultados apresentados na Tabela 5.1 na página 86, a seleção de atributos por *wrapper forward*, *wrapper backward*, filtro ID3 e filtro $\mathcal{CI}$, em média, foi de 43,20%, 75,46%, 69,74% e 60,72% atributos selecionados, respectivamente. Comparando esses resultados com o número médio de regras, pode-se notar que quanto maior a quantidade de atributos removidos, maior é a quantidade de regras induzidas por CN2.

Na Figura 5.5 é mostrada a proporção da diferença do número de regras induzidas utilizando FSS e sem utilizar FSS. Valores positivos indicam um aumento no número de regras induzidas por CN2 utilizando FSS; valores negativos indicam que FSS ajudou a reduzir o número de regras. No geral, o uso de seleção de atributos ocasionou 12 reduções, 4 igualdades e 27 incrementos no número de regras.

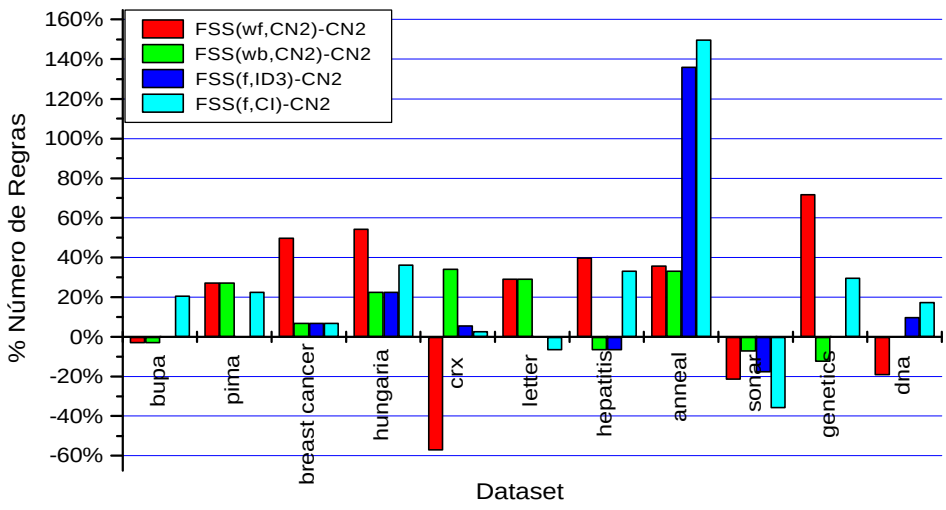


Figura 5.5: Proporção da diferença do número de regras induzidas por CN2 utilizando FSS e sem utilizar FSS

Na Tabela 5.6 na página seguinte é mostrado o número de exemplos cobertos pelas regras. É importante lembrar que a quantidade de exemplos cobertos pode ser maior do que o número de exemplos no conjunto de exemplos. Isso porque cada exemplo pode ser coberto por mais de uma regra pelo classificador CN2 — Seção 3.4. Os resultados mostram um aumento relativo de $3,59\%$ para FSS(f,$\mathcal{CI}$), tendo diminuído nos demais casos: $-0,91\%$ para FSS(wf,CN2), $-13,03\%$ para FSS(wb,CN2) e $-1,97\%$ para FSS(f,ID3).

Na Figura 5.6 é mostrada a proporção da diferença do número de exemplos cobertos pelas

Dataset	CN2	FSS(wf,CN2)	FSS(wb,CN2)	FSS(f,ID3)	FSS(f,$\mathcal{CI}$)
bupa	418	*374	*374	418	*262
pima	1058	1113	1113	1058	*1054
breast cancer	2032	*1738	*1949	*1972	2121
hungaria	430	*283	*373	469	481
crx	1145	*613	*1133	*1137	*1121
letter	17870	20172	20172	17870	17909
hepatitis	285	*197	*256	*259	302
anneal	1380	1428	1412	1495	1715
sonar	344	*293	*305	*311	*287
genetics	5469	*4587	*4491	*4805	6026
dna	9511	*8781	N/A	*9362	10099
Média	3631	3598	3158	3560	3762

Tabela 5.6: Número de exemplos cobertos pelas regras induzidas por CN2; valores marcados com * indicam diminuição do número de exemplo cobertos em relação ao algoritmo sem utilizar FSS

regras utilizando FSS e sem utilizar FSS. Valores positivos indicam um aumento no número de exemplos cobertos por CN2 utilizando FSS (um mesmo exemplo é coberto por várias regras); valores negativos indicam que FSS ajudou a reduzir o número de exemplos cobertos (as regras são mais disjuntas que no caso anterior). No geral, o uso de seleção de atributos ocasionou 25 reduções, 3 igualdades e 15 ampliações no número de exemplos cobertos.

Considerando o número de condições por regra, é possível observar na Tabela 5.7 que, em média, houve um aumento relativo de 4,10% utilizando FSS(wf,CN2) e de 1,81% utilizando FSS(f,ID3) e uma diminuição relativa de 1,39% utilizando FSS(wb,CN2) e de 2,25% utilizando FSS(f,$\mathcal{CI}$).

Dataset	CN2	FSS(wf,CN2)	FSS(wb,CN2)	FSS(f,ID3)	FSS(f,$\mathcal{CI}$)
bupa	3,62±1,18	3,36±1,11	3,36±1,11	3,62±1,18	1,90±0,30
pima	3,52±0,88	3,54±0,99	3,54±0,99	3,53±0,85	3,56±1,02
breast cancer	2,86±1,10	2,95±0,74	2,80±1,15	3,20±1,21	3,13±0,83
hungaria	3,46±1,37	3,21±1,30	3,63±1,36	3,41±1,34	3,30±1,32
crx	3,54±1,09	3,00±0,93	3,57±0,95	3,73±0,90	3,42±0,87
letter	5,88±2,15	6,35±2,31	6,35±2,31	5,88±2,15	5,92±2,27
hepatitis	2,60±0,91	2,81±1,08	3,07±1,14	2,93±1,14	3,10±1,02
anneal	4,06±2,27	3,67±1,65	3,67±1,60	3,31±1,56	3,34±1,36
sonar	2,11±0,57	2,36±0,79	2,31±0,68	2,48±0,73	2,72±0,67
genetics	3,00±1,37	3,80±1,25	3,29±1,46	3,02±1,34	3,07±1,24
dna	5,06±1,13	6,27±2,37	N/A	5,33±1,34	5,34±1,25
Média	3,61	3,76	3,56	3,67	3,53

Tabela 5.7: Número de condições (média e desvio padrão) das regras induzidas por CN2

Na Figura 5.7 na página 96 é mostrada a diferença absoluta, em desvios padrões, do número médio de condições nas regras induzidas por CN2 nos diversos experimentos. Embora em nenhum

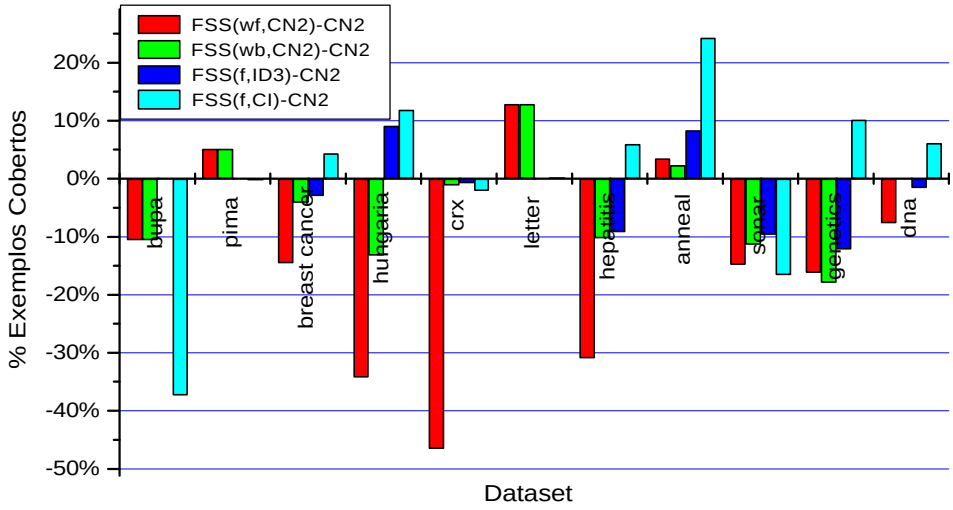


Figura 5.6: Proporção da diferença do número de exemplos cobertos pelas regras induzidas por CN2 utilizando FSS e sem utilizar FSS

caso FSS supere significativamente CN2 no número de regras, a tendência mostrada pelo gráfico refere-se a um aumento no número de condições nas regras: 28 incrementos, 2 igualdades e 11 reduções.

5.7 Resumo dos Resultados

Veritatem dies aperit (Time discovers the truth).

—*Seneca*

Encontrar um bom subconjunto de atributos é um problema difícil e várias abordagens foram propostas na literatura, tais como filtros e *wrappers*. Este trabalho se concentrou no *wrapper* $\mathcal{MLC}$++ para FSS assim como no uso de ID3 e a ferramenta MineSet™ *Column Importance*, experimentalmente avaliados em onze conjuntos de exemplos e quatro indutores. Os resultados obtidos mostram, como esperado, que *wrappers* utilizando seleção *forward* e *backward* fornecem um melhor conjunto de atributos para estimativa de precisão do que ambos filtros utilizados.

Mesmo considerando que os *wrappers* exibem uma melhor precisão, seu tempo computacional mostrou-se, em média, 10^3 vezes mais demorado do que filtros. Quando se considera conjuntos de exemplos com muitos atributos (anneal, sonar, genetics e dna) os filtros foram, em média, 10^4 vezes mais rápidos que *wrappers*. É importante notar que os filtros não provocaram uma perda de precisão significativa quando comparados com o indutor padrão. Por exemplo, o filtro

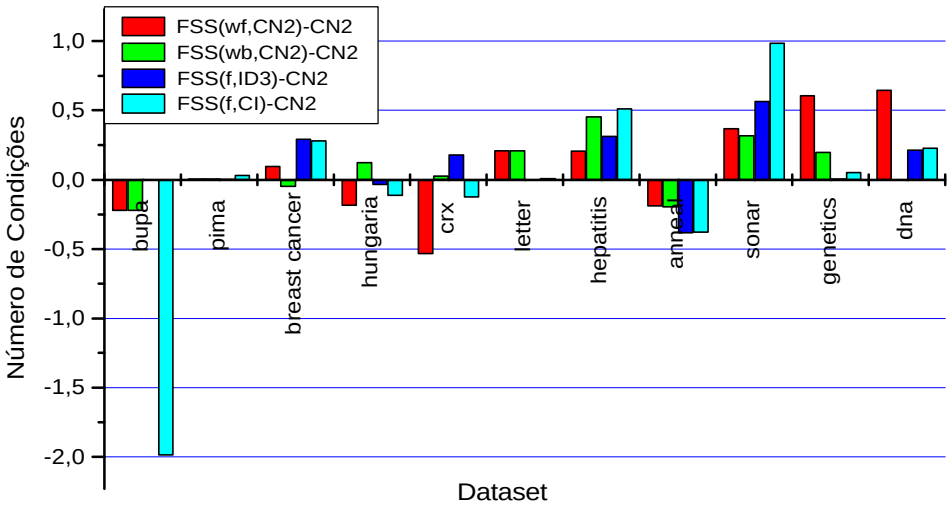


Figura 5.7: Diferença absoluta, em desvios padrões, do número de condições das regras induzidas por CN2

ID3 não prejudicou o desempenho exceto para um dos onze conjuntos de exemplos analisados aqui.

Para Mineração de Dados, uma restrição especial para a seleção de atributos é que o tamanho do conjunto de exemplos é normalmente grande, tanto vertical quanto horizontalmente. Nessa situação, o tempo de execução pode inviabilizar o uso de *wrappers*. Espera-se que FSS seja uma tarefa rápida de pré-processamento e, nesse sentido, a abordagem filtro é mais apropriada que *wrappers*. Uma conclusão deste trabalho é que filtros devem ser considerados antes do uso de *wrappers* para FSS quando se possui muitos atributos ou exemplos.

Pode ser argumentado que a aplicação típica de *wrappers* para seleção de atributos, em grandes conjuntos de exemplos, ocorre apenas uma vez e que um método de estimativa da precisão menos custoso (o *default* $\mathcal{MLC}$++ é 10-*fold cross-validation*) poderia ser usado. Por exemplo, nos experimentos realizados no conjunto de exemplos dna utilizando *wrapper forward*, mais de 1.200 nós foram avaliados para cada indutor. De fato, para grandes conjuntos de exemplos uma estimativa *holdout* pode ser usada ao invés de *cross-validation*, melhorando o tempo de execução por um fator de dez. Além disso, como o *wrapper* $\mathcal{MLC}$++ é um algoritmo que pode ser interrompido a qualquer momento, a busca pode ser interrompida após algum tempo e o melhor subconjunto encontrado até então pode ser usado. Mesmo assim, muito trabalho ainda precisa ser realizado para encontrar abordagens mais efetivas para aplicações de DM.

Uma grande parte dos estudos em AM supervisionado assume a precisão, em um subconjunto de teste, como o componente de avaliação de desempenho e, nos experimentos realizados, o critério de avaliação dos subconjuntos considerou apenas a precisão do classificador induzido. Entretanto, para KDD é também importante uma atenção cuidadosa em obter a aceitação do usuário ou especialista e validar o conhecimento extraído. Para isso, não apenas a precisão, mas também o grau de compreensão do conhecimento induzido deve ser considerado. Entretanto, como mostrado nos experimentos sobre o impacto do uso de FSS com o indutor CN2, em média, há um aumento no número de regras e no número de condições nas regras. Além disso, a proporção de atributos selecionados é inversamente proporcional à quantidade de regras induzidas. Isto tudo indica que, contrariamente ao esperado, a seleção de atributos dificulta a compreensão, por seres humanos, de regras induzidas após o processo de FSS. Entretanto, é importante que o número de condições por regra é apenas uma medida sintática, que não considera a semântica de cada atributo. Estudos futuros podem continuar o trabalho aqui iniciado, verificando se o comportamento mostrado pelo indutor CN2 com atributos selecionados por FSS se estende para outros indutores que geram classificadores simbólicos bem como a utilização de amostras na abordagens *wrapper* e filtros.

5.8 Considerações Finais

> Thinking is the hardest work there is, which is the probable reason why so few engage in it.
>
> —*Henry Ford*

Os algoritmos de Aprendizado de Máquina comumente utilizam conjuntos de exemplos contendo poucos exemplos com número restrito de atributos. Quando se trabalha com uma dimensão restrita é possível utilizar um algoritmo de seleção de atributos que simplesmente procura pelas possíveis combinações, selecionando aquelas que melhoram o desempenho do algoritmo de Aprendizado de Máquina.

Com a aplicação, cada vez mais freqüente, de técnicas de Aprendizado de Máquina em grande volume de dados, o problema de focalizar a informação mais relevante tornou-se muito importante. Assim, um dos principais problemas em AM e DM consiste em selecionar os atributos relevantes e, conseqüentemente, eliminar aqueles irrelevantes. Dentre as várias técnicas de redução de dados, a remoção de um atributo é a que causa a maior diminuição.

Existem diversas razões para se realizar a seleção de atributos. Uma dessas razões é que a maioria dos algoritmos de AM computacionalmente viáveis não trabalham bem na presença de

um número grande de atributos. Assim, a seleção pode melhorar a precisão dos classificadores gerados por esses algoritmos. Outra razão é que a seleção pode melhorar o grau de compreensão pelos humanos e também, por exemplo, das regras de indução geradas por algoritmos simbólicos de AM. Uma terceira razão é o alto custo para a coleta dos dados, já que em muitos domínios isso pode ser um processo muito caro. Um exemplo disso são as informações existentes nos registros médicos. Algumas delas, tais como exames laboratoriais ou de diagnóstico por imagem, possuem um alto custo financeiro e, possivelmente, nem toda a informação é necessária para o aprendizado de um conceito. Finalmente, a seleção pode reduzir os custos de processamento de grande volume de dados.

Mesmo considerando o fato que os algoritmos de AM possam ser utilizados com muitos atributos, sabe-se que o desempenho dos classificadores induzidos é prejudicado quando existem muitos atributos irrelevantes (Dietterich 1997b). Além disso, estatisticamente, exemplos com muitos atributos e ruídos fornecem pouca informação.

A seleção de atributos pode ser definida como o problema de encontrar um subconjunto de atributos que descrevem o conjunto de exemplos reduzido tão bem como conjunto original. O objetivo prático consiste em remover atributos estranhos, e não necessariamente selecionar o subconjunto ótimo, deixando o conjunto de exemplos em uma dimensão que permita sua manipulação por outros algoritmos. É importante ter em mente que é muito mais seguro incluir mais atributos do que menos, uma vez que os indutores podem eventualmente lidar com atributos extras, mas não podem compensar os atributos que foram descartados. Este ponto de vista foi o adotado nos experimentos realizados.

No próximo capítulo é descrita a composição de atributos, atividade de pré-processamento de certa forma oposta à seleção de atributos aqui apresentada. Além das possíveis abordagens para a composição de atributos é proposta uma metodologia, orientada pelo conhecimento fornecido por um especialista/usuário. Experimentos utilizando a metodologia proposta são também reportados juntamente com alguns resultados e as conclusões obtidas.

Capítulo 6

Composição de Atributos

The eternal mystery of the world is its comprehensibility.

—*Albert Einstein*

No aprendizado indutivo, os indutores dependem basicamente dos dados para construir suas hipóteses. Assim, linguagens de representação inadequadas podem tornar difícil um problema de aprendizado. Os atributos, mesmo sendo individualmente inadequados, podem, algumas vezes, ser convenientemente combinados gerando atributos derivados que podem mostrar-se altamente representativos para a descrição de um conceito. Indução construtiva — IC — é o processo de combinar os atributos originalmente presentes no conjunto de exemplos, criando-se atributos derivados (Michalski 1978). Dependendo de como é realizada a combinação dos atributos, isso pode provocar uma mudança do espaço de representação, facilitando o aprendizado.

Os atributos podem ser considerados inadequados para o aprendizado quando são fracamente ou indiretamente relevantes, condicionalmente relevantes ou medidos de modo inapropriado (Lee, Monard & Baranauskas 1999; Baranauskas, Monard & Horst 1999b; Baranauskas, Monard & Horst 1999a). Se os atributos utilizados para descrever o conjunto de treinamento são inadequados, os algoritmos de aprendizado provavelmente criarão hipóteses imprecisas ou excessivamente complexas (Bloedorn & Michalski 1998).

Diferentemente da seleção de atributos, na qual somente os atributos selecionados são mostrados ao algoritmo, decrementando assim o espaço de busca de atributos, a Indução Construtiva aumenta o espaço de busca de atributos, caso os atributos originais que foram combinados para formar os atributos derivados não sejam removidos do conjunto de exemplos.

Para a aplicação da Indução Construtiva é necessário decidir quais operadores construtivos devem ser utilizados assim como quais atributos originais (ou primitivos) devem ser combinados

utilizando os operadores.

Uma outra observação importante é que, em geral, o processo de construção de atributos é intratável, já que o número de atributos que podem ser combinados é uma função combinatória do número de atributos existentes multiplicado pelo número de operadores. Conseqüentemente, a construção de atributos somente é viável quando articulada com heurísticas que possibilitam a redução dos atributos originais e operadores construtivos que serão utilizados para construir os atributos derivados.

Na seção seguinte é introduzida a motivação para a combinação de atributos através de um exemplo. Em seguida, são descritas abordagens que podem ser utilizadas para construir novos atributos. Utilizando uma dessas abordagens, Indução Construtiva guiada pelo conhecimento, é proposta uma metodologia para a construção de atributos, a qual é avaliada com experimentos reportados na Seção 6.4.

6.1 Motivação

Science is organized knowledge.

—*Herbert Spencer*

Como motivação para a combinação de atributos, considere o caso clássico dos robôs amigos e inimigos. Os atributos cabeça, corpo, segura e sorri são utilizados para determinar se um robô é amigo ou inimigo. Na Tabela 6.1 na próxima página é mostrado o conjunto de exemplos sobre robôs.

Construindo a árvore de decisão sem poda utilizando C4.5 para esse conjunto de exemplos, ilustrada na Figura 6.1 na página oposta, pode-se notar que, dos cinco atributos apresentados, dois são considerados para decidir se um robô é amigo ou inimigo, gerando uma árvore de dois níveis de profundidade.

Utilizando Indução Construtiva, é possível derivar um novo atributo denominado mesma-forma a partir da comparação dos valores dos atributos originais cabeça e corpo. Se o robô possui cabeça e corpo com a mesma forma, o atributo derivado mesma-forma assume o valor 'sim', caso contrário assume o valor 'não', segundo a Tabela 6.2 na próxima página.

Considerando o atributo derivado e utilizando-se novamente o indutor C4.5 com o conjunto de exemplos da Tabela 6.2, obtém-se a árvore de decisão mostrada na Figura 6.2 na página oposta. É possível observar que, utilizando apenas o novo atributo, é possível distinguir entre robôs amigos e inimigos e que nenhum dos atributos originais foi explicitamente necessário para a indução do conceito, o qual é mais simples do que o anterior (Flach & Lavrac 2000).

Exemplos	cabeça	corpo	sorri	segura	Classe
T_1	triangular	triangular	sim	balão	amigo
T_2	quadrada	quadrado	sim	balão	amigo
T_3	redonda	redondo	sim	bandeira	amigo
T_4	quadrada	triangular	não	espada	inimigo
T_5	triangular	redondo	sim	espada	inimigo
T_6	redonda	quadrado	não	bandeira	inimigo

Tabela 6.1: Exemplos de robôs amigos e inimigos

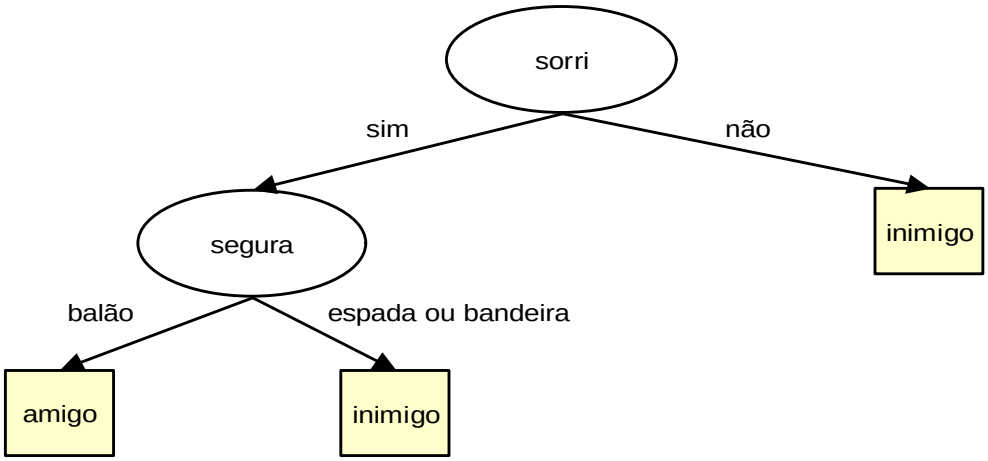


Figura 6.1: Árvore de decisão para o exemplo de robôs amigos e inimigos

Exemplos	cabeça	corpo	sorri	segura	mesma-forma	Classe
T_1	triangular	triangular	sim	balão	sim	amigo
T_2	quadrada	quadrado	sim	balão	sim	amigo
T_3	redonda	redondo	sim	bandeira	sim	amigo
T_4	quadrada	triangular	não	espada	não	inimigo
T_5	triangular	redondo	sim	espada	não	inimigo
T_6	redonda	quadrado	não	bandeira	não	inimigo

Tabela 6.2: Exemplos de robôs amigos e inimigos depois da construção do atributo mesma-forma

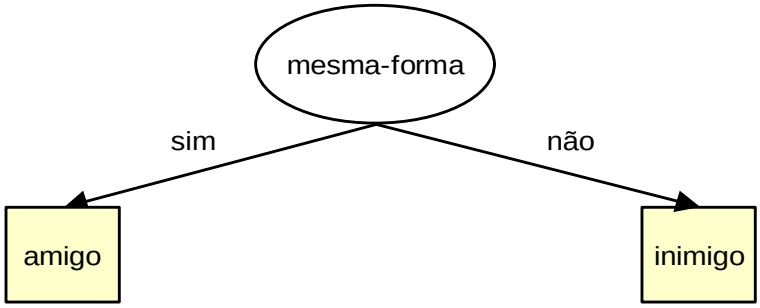


Figura 6.2: Árvore de decisão para o exemplo de robôs amigos e inimigos após a construção do atributo mesma-forma

Entretanto, nem sempre é simples combinar os atributos, formando atributos derivados, com melhor poder de expressão de conceitos, como no exemplo anterior. Na realidade, para efetuar IC, são necessários mecanismos para a geração de novas dimensões — que sejam mais relevantes ao problema — no espaço de representação de exemplos, assim como modificar ou remover atributos dentre aqueles fornecidos originariamente.

Em outros termos, um sistema de IC realiza uma transformação do espaço de representação dos exemplos. Uma vez que o espaço de representação é definido, um algoritmo de aprendizado relativamente simples pode ser suficiente para encontrar uma estrutura de conhecimento desejável, geralmente mais compacta. Para definir o espaço de representação, através da criação de atributos derivados utilizando IC, existem diversas abordagens, as quais são descritas na próxima seção.

6.2 Abordagens para Composição de Atributos

Labor omnia vincit (Labor conquers all things).

—*Virgil*

O processo de construção de atributos pode ser guiado e controlado pelo usuário/especialista ou pode ser conduzido automaticamente pelo sistema de aprendizado. Na realidade, existem relativamente poucos trabalhos na área de IC. Mesmo assim, os métodos de Indução Construtiva podem ser agrupados, de acordo com a informação utilizada na busca do melhor espaço de representação, nas categorias descritas a seguir (Bloedorn & Michalski 1998; Wnek & Michalski 1994; Wnek & Michalski 1993).

6.2.1 Indução Construtiva guiada pelos Dados

A indução construtiva guiada pelos dados (DCI ou *data-driven constructive induction*) é baseada na análise do conjunto de exemplos e na detecção de relacionamentos entre os atributos. Como exemplos de DCI tem-se o aprendizado do mundo dos blocos de Winston, o qual lida com aprendizado de conceitos que caracterizam construções simples de blocos de brinquedo; o programa assume que os exemplos são fornecidos por um professor que, cuidadosamente, escolhe o tipo dos exemplos utilizados, bem como sua ordem de apresentação (Winston 1970). Os programas SPROUTER e VERE procuram encontrar generalizações de um conjunto de exemplos positivos (Dietterich & Michalski 1983). Algumas capacidades de aprendizado construtivo foram incorporadas ao sistema BACON que formula automaticamente expressões matemáticas, incorporando leis matemáticas e químicas (Langley 1983).

6.2.2 Indução Construtiva guiada por Hipótese

Na indução construtiva guiada por hipótese (HCI ou *hypothesis-driven constructive induction*) as mudanças no espaço de representação são propostas com base em análises de hipóteses induzidas em iterações consecutivas. Os padrões detectados em uma iteração são utilizados na iteração seguinte. Como exemplos de sistemas utilizando HCI tem-se BLIP (Emde, Habel & Rollinger 1983), FRINGE (Pagallo & Haussler 1990) e AQ-HCI (Wnek & Michalski 1994).

6.2.3 Indução Construtiva guiada pelo Conhecimento

A indução construtiva guiada pelo conhecimento (KCI ou *knowledge-driven constructive induction*) está baseada no conhecimento do domínio, fornecido por um especialista, o qual é utilizado para construir um novo espaço de representação. Como exemplos, tem-se o programa AM (*Automated Mathematician*) que é um sistema específico para a área de matemática (Lenat 1983). O programa de propósito geral INDUCE, para aprendizado de descrições estruturais a partir de exemplos, incorpora várias técnicas de generalização construtivas (Michalski 1983a).

6.2.4 Indução Construtiva Multi-estratégia

A indução construtiva multi-estratégia (MCI ou *multistrategy constructive induction*) combina dois ou mais dos métodos anteriores. Entre os sistemas que utilizam MCI tem-se INDUCER-1 (Michalski 1983a), DUCE (Muggleton 1987) e LEX (Mitchell, Utgoff & Banerji 1983). Fayyad, Djorgovski & Weir (1996) empregam MCI no sistema SKICAT ao utilizar 40 atributos primitivos obtidos pelo sistema FOCAS; 4 atributos derivados e normalizados a partir dos primitivos e 2 atributos requerendo medidas empíricas, obtidos pela aplicação de um algoritmo de AM.

6.3 Metodologia Proposta

> Less than fifteen per cent of the people do any original thinking on any subject. The greatest torture on the world for most people is to think.
>
> —*Luther Bunbark*

Utilizando a Indução Construtiva guiada pelo conhecimento, a qual é baseada no conhecimento do domínio provido por um usuário/especialista, nesta seção é descrita a metodologia, por nós proposta. Com base nessa metodologia, uma série de experimentos foi realizada, de forma a avaliar a efetividade dos atributos derivados utilizando dois indutores (Lee, Monard &

Baranauskas 2000). A metodologia experimental utilizada, bem com os principais resultados obtidos, são reportados nas próximas seções.

Na metodologia proposta, composta por três etapas, assume-se inicialmente que é dado um conjunto de exemplos T, cada exemplo composto por m atributos originais $\{X_1, X_2, \ldots, X_m\}$. A partir de T, assume-se que L novos atributos $\{X_1^{'}, X_2^{'}, \ldots, X_L^{'}\}$ sejam construídos, com o auxílio do usuário/especialista, combinando alguns atributos originais. Com base nisso, é possível aplicar as seguintes três etapas:

Primeira Etapa Considerando cada um dos L atributos derivados $\{X_1^{'}, X_2^{'}, \ldots, X_L^{'}\}$ que o especialista pode sugerir, o conjunto original de exemplos T é acrescido com cada um desses atributos derivados, um atributo por vez. Obtém-se, assim, L conjuntos derivados de exemplos, denotados por $\{T_1^{'}, T_2^{'}, \ldots, T_L^{'}\}$, cada um contendo $m+1$ atributos, onde $T_i^{'} = \{X_1, X_2, \ldots, X_m, X_i^{'}\}$ e $1 \leq i \leq L$.

Segunda Etapa Cada conjunto derivado de exemplos $\{T_1^{'}, T_2^{'}, \ldots, T_L^{'}\}$ é então fornecido a um (ou mais) indutor que realiza algum tipo de seleção (embutida) de atributos, tal como regras ou árvores de decisão, gerando assim um classificador $h_i^{'}$ para cada conjunto $T_i^{'}$.

A heurística por nós utilizada consiste na idéia que se o novo atributo $X_i^{'}$ não está presente no classificador $h_i^{'}$, então isso é uma indicação que $X_i^{'}$ não é essencial para o conceito aprendido pelo indutor (Michalski & Kaufman 1998). Nesse caso, o conjunto de exemplos $T_i^{'}$ é descartado não sendo considerado para a próxima etapa. Conseqüentemente, após esta etapa apenas o subconjunto dos conjuntos derivados de exemplos que foram gerados na primeira etapa e atendem a condição da segunda etapa são considerados para a terceira etapa.

Terceira Etapa Nesta etapa, a estimativa de erro é realizada. Inicialmente, uma *r-fold stratified cross-validation* é realizada no conjunto original de exemplos T para estimar seu erro médio $\mathsf{mean}(T)$, calculado utilizando-se (3.1) definida na página 49. Em seguida, *r-fold stratified cross-validation* é realizada em cada um dos conjuntos derivados de exemplos provenientes da segunda etapa. Embora qualquer medida de erro possa ser utilizada, recomenda-se *r-fold stratified cross-validation* para que a proporção de exemplos de cada classe em cada *fold* seja a mesma que o conjunto original de exemplos.

Denotando-se o erro, calculado por *r-fold stratified cross-validation* sobre o conjunto derivado de exemplos $T_i^{'}$ por $\mathsf{mean}(T_i^{'})$, somente aqueles conjuntos que possuem uma diferença absoluta em desvios padrões $\mathsf{ad}(T_i^{'} - T) < 0$ são selecionados para investigação futura,

uma vez que isso sugere que o novo atributo, além de aparecer no classificador induzido, também melhorou sua precisão. A diferença ad é calculada utilizando-se (3.6) definida na página 51.

Considerando todo esse processo como uma metodologia geral para a aplicação de indução construtiva, conforme a Figura 6.3, a situação ideal aconteceria quando os atributos primitivos, utilizados na construção dos atributos derivados, não fossem selecionados, pelo indutor, na segunda etapa.

Entretanto, esse pode não ser o caso. Uma razão possível é que os atributos derivados não capturem perfeitamente a informação embutida em cada atributo individual para o indutor escolhido, ou que o seu poder de predição é equivalente a alguns dos atributos originais. Outra razão para isso pode ser devido ao fato que o próprio conjunto de exemplos já tenha sido pré-processado de forma tal que os atributos originais são, por si só, os mais relevantes.

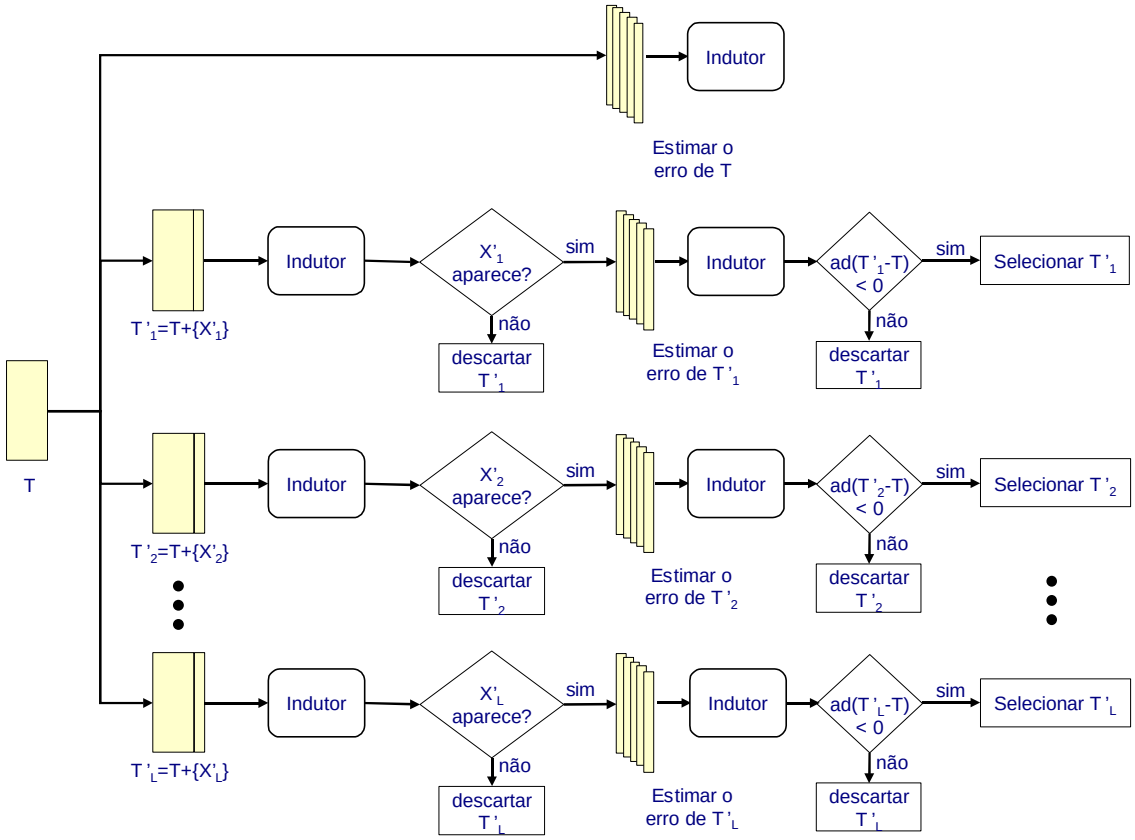


Figura 6.3: Metodologia proposta para Indução Construtiva guiada pelo conhecimento

6.4 Metodologia Experimental

> Our minds possess by nature an insatiable desire to know the truth.
>
> —*Marcus Tullius Cicero*

Utilizando a metodologia proposta, uma série de experimentos foi realizada, de forma a avaliar a efetividade dos atributos derivados utilizando os indutores C4.5RULES e CN2 com os conjuntos de exemplos pima, cmc, smoke e hepatitis, descritos na Seção 3.3 na página 51. Os conjuntos de exemplos não foram pré-processados e os indutores foram executados com seus parâmetros *defaults*. O critério de escolha desses conjuntos de exemplos foi devido ao domínio de conhecimento do usuário/especialista que auxiliou nos experimentos, uma vez que há interesse de sua ajuda para compor os atributos derivados.

Os experimentos realizados seguem as três etapas da metodologia proposta. Na primeira etapa, após analisar cada conjunto de exemplos, o usuário/especialista sugeriu dois novos atributos para os conjuntos de exemplos pima, cmc e smoke e apenas um atributo novo para o conjunto de exemplos hepatitis, como relacionados na Tabela 6.3.

Conjunto Original	Conjuntos Derivados	
pima	pima_1'	pima_2'
cmc	cmc_1'	cmc_2'
smoke	smoke_1'	smoke_2'
hepatitis	$\text{hepatitis}_1'$	—

Tabela 6.3: Conjuntos original e derivados de exemplos

Na segunda etapa, os indutores C4.5RULES e CN2 foram executados com todos os exemplos em cada um dos sete conjuntos de exemplos derivados. Nestes experimentos, as regras induzidas pelos indutores apresentaram os novos atributos X_1' e X_2' para todos os conjuntos de exemplos (para o conjunto de exemplos hepatitis o único atributo derivado X_1' também apareceu no classificador induzido). Portanto, nenhum conjunto derivado de exemplos foi descartado na segunda etapa.

Em seguida, na terceira etapa, os dois indutores foram executados com os conjuntos originais de exemplos assim como com os sete conjuntos derivados de exemplos e a taxa de erro foi medida utilizando 10-*fold stratified cross-validation*. Após isso, foram selecionados, para investigação futura, apenas os conjuntos de exemplos que obtiveram uma melhoria da precisão, quando comparados com a precisão do respectivo conjunto original de exemplos.

6.5 Resultados

> Suppressio veri, expressio falsi (Suppression of truth is a false representation).
>
> —*Latin Proverb*

Os resultados experimentais obtidos são apresentados em uma tabela para cada conjunto de exemplos. Em cada tabela, são mostrados, para cada conjunto original e derivados e para cada indutor, o número de atributos do conjunto de exemplos fornecido ao indutor (Total #A); os atributos individuais utilizados no classificador induzido, sendo que os atributos originais são indicados por '$\circ$' e os derivados são indicados por '$\bullet$'; o número de atributos selecionados pelo classificador (#A) e a proporção de atributos selecionados (%A). O indutor C4.5RULES é representado como C4.5R.

Além disso, na tabela de resultados também são mostrados a média e desvio padrão da taxa de erro (mean) obtida através de 10-*fold stratified cross-validation*, incluindo a diferença absoluta em desvios padrões (ad) entre o classificador induzido utilizando o conjunto derivado de exemplos e o classificador induzido utilizando o conjunto original de exemplos, ou seja, $\mathsf{ad}(T_i^{'} - T)$. Assim, se ad é positivo (negativo), o classificador induzido utilizando o conjunto original (derivado) de exemplos supera aquele construído utilizando o conjunto derivado (original). Entretanto, como já mencionado, para um classificador superar estatisticamente o outro com nível de confiança de 95%, o valor de ad deve ser maior do que 2 ou menor do que -2, respectivamente. Os resultados para os conjuntos de exemplos pima, cmc, smoke e hepatitis são apresentados nas Tabelas 6.4, 6.5, 6.6 e 6.7, respectivamente.

6.5.1 Conjunto de Exemplos pima

Dois atributos derivados foram construídos para este conjunto de exemplos com o auxílio do especialista:

- $X_1^{'}$: este atributo verifica se a taxa de glicose e a pressão sangüínea estão fora dos limites normais. Ele é a combinação dos atributos primitivos *plasma* (atributo #2) e *diastolic* (atributo #3).

- $X_2^{'}$: este atributo verifica se a taxa de glicose está fora dos limites normais e se não há insulina no soro após duas horas. Ele é a combinação dos atributos primitivos *plasma* (atributo #2) e *two* (atributo #5).

Número do Atributo	pima C4.5R	pima CN2	pima$_1^{'}$ C4.5R	pima$_1^{'}$ CN2	pima$_2^{'}$ C4.5R	pima$_2^{'}$ CN2
#1 (X_1)		○				○
#2 (X_2)	○	○	○	○	○	○
#3 (X_3)	○	○	○	○	○	○
#4 (X_4)		○		○		○
#5 (X_5)		○		○	○	○
#6 (X_6)	○	○	○	○	○	○
#7 (X_7)	○	○	○	○	○	○
#8 (X_8)	○	○	○	○	○	○
#9 ($X_1^{'}$)			•			
#10 ($X_2^{'}$)					•	•
Total #A	8	8	9	9	9	9
#A	5	8	6	7	7	9
%A	62,50%	100,00%	66,67%	77,78%	77,78%	100,00%
mean	26,00±1,03	25,38±1,38	25,61±1,12	25,90±1,15	26,52±1,14	25,77±1,33
ad			$-0,36$	0,41	0,48	0,29

Tabela 6.4: Conjunto de exemplos pima antes e após IC

Na Tabela 6.4, pode ser observado que o atributo $X_1^{'}$ não é essencial para CN2 e que há apenas uma pequena melhoria utilizando pima$_1^{'}$ com C4.5RULES. É possível notar também que, em todos os casos nos quais o atributo derivado foi selecionado, o atributo original também o foi. Além de ocorrer no conjunto de exemplos pima, isso também ocorreu com os conjuntos de exemplos cmc e smoke, mas não com hepatitis. Uma pequena degradação no desempenho pode ser notada para pima$_1^{'}$ com CN2 embora o novo atributo não tenha sido selecionado.

6.5.2 Conjunto de Exemplos cmc

Dois atributos derivados foram construídos para este conjunto com o auxílio do usuário:

- $X_1^{'}$: este atributo verifica se o nível educacional do marido é igual ou não ao da esposa. Ele é a combinação dos atributos primitivos *wedu* (atributo #2) e *hedu* (atributo #3).
- $X_2^{'}$: este atributo verifica se a esposa tem um padrão de vida compatível com seu nível educacional. Ele é a combinação dos atributos primitivos *wedu* (atributo #2) e *stdliv* (atributo #8).

Como pode ser visto na Tabela 6.5 na próxima página, embora o atributo $X_1^{'}$ não tenha sido selecionado para o conjunto de exemplos cmc$_1^{'}$ utilizando C4.5RULES, o desempenho degradou quando comparado com o conjunto original de exemplos. O conjunto de exemplos cmc$_2^{'}$ utilizando CN2 apresentou uma degradação considerável mas não com nível de significância de 95%. Ambos cmc$_1^{'}$ utilizando CN2 e cmc$_2^{'}$ utilizando C4.5RULES apresentaram uma suave melhoria no desempenho.

Número do Atributo	cmc C4.5R	cmc CN2	cmc$_1^{'}$ C4.5R	cmc$_1^{'}$ CN2	cmc$_2^{'}$ C4.5R	cmc$_2^{'}$ CN2
#1 (X_1)	○	○	○	○	○	○
#2 (X_2)	○	○	○	○	○	○
#3 (X_3)	○	○	○	○	○	○
#4 (X_4)	○	○	○	○	○	○
#5 (X_5)	○	○	○	○	○	○
#6 (X_6)	○	○	○	○	○	○
#7 (X_7)	○	○	○	○	○	○
#8 (X_8)	○	○	○	○	○	○
#9 (X_9)	○	○	○	○	○	○
#10 ($X_1^{'}$)				•		
#11 ($X_2^{'}$)					•	•
Total #A	9	9	10	10	10	10
#A	9	9	9	10	10	10
%A	100,00%	100,00%	90,00%	100,00%	100,00%	100,00%
mean	45,90±1,38	49,64±1,01	47,87±1,54	49,50±1,04	46,37±0,97	52,22±1,09
ad			1,09	$-0,68$	$-0,07$	1,85

Tabela 6.5: Conjunto de exemplos cmc antes e após IC

6.5.3 Conjunto de Exemplos smoke

Dois atributos derivados foram construídos para este conjunto de exemplos com o auxílio do usuário:

- $X_1^{'}$: este atributo representa o status da pessoa entrevistada na época da pesquisa. Ele é a combinação dos atributos primitivos *smoking1* (atributo #6), *smoking2* (atributo #7), *smoking3* (atributo #8) e *smoking4* (atributo #9).
- $X_2^{'}$: este atributo mostra uma comparação entre o local onde a pessoa entrevistada trabalha com relação à cidade de Toronto, Canadá, se ela trabalha em casa ou não e se ela vive na cidade de Toronto ou fora dela. Ele é a combinação dos atributos primitivos *work1* (atributo #3), *work2* (atributo #4) e *residence* (atributo #5).

Na Tabela 6.6 na página seguinte pode-se observar que a inserção dos dois atributos derivados individualmente aumentou a precisão para o indutor CN2 mas não para C4.5RULES. O atributo derivado $X_1^{'}$ degradou o desempenho para C4.5RULES mesmo $X_1^{'}$ não sendo selecionado.

6.5.4 Conjunto de Exemplos hepatitis

Um atributo derivado foi construído para este conjunto com o auxílio do especialista:

- $X_1^{'}$: este atributo indica se o paciente sobreviverá ou não. Ele é a combinação dos atributos primitivos *liver-firm* (atributo #9), *ascites* (atributo #12) e *varices* (atributo #13).

Número do Atributo	smoke C4.5R	smoke CN2	smoke$_1^{'}$ C4.5R	smoke$_1^{'}$ CN2	smoke$_2^{'}$ C4.5R	smoke$_2^{'}$ CN2
#1 (X_1)	○	○	○	○	○	○
#2 (X_2)	○	○	○	○	○	○
#3 (X_3)	○	○	○	○	○	○
#4 (X_4)	○	○	○	○	○	○
#5 (X_5)	○	○	○	○	○	○
#6 (X_6)	○	○	○	○	○	○
#7 (X_7)	○	○	○	○	○	○
#8 (X_8)		○	○	○	○	○
#9 (X_9)	○	○	○	○	○	○
#10 (X_{10})	○	○	○	○	○	○
#11 (X_{11})	○	○	○	○	○	○
#12 (X_{12})	○	○	○	○	○	○
#13 (X_{13})	○	○	○	○	○	○
#14 ($X_1^{'}$)				•		
#15 ($X_2^{'}$)					•	•
Total #A	13	13	14	14	14	14
#A	12	13	13	14	14	14
%A	92,31%	57,40%	92,86%	100,00%	100,00%	100,00%
mean	32,71±0,65	31,87±0,35	33,28±0,80	31,56±0,45	32,93±0,49	31,49±0,45
ad			0,78	$-0,77$	0,38	$-0,94$

Tabela 6.6: Conjunto de exemplos smoke antes e após IC

Na Tabela 6.7 na próxima página é possível observar que o atributo derivado $X_1^{'}$ foi responsável por um atributo original (atributo #12) não aparecer no classificador induzido do conjunto de exemplos hepatitis$_1^{'}$ com C4.5RULES, mesmo causando uma melhoria do desempenho. Para CN2, o novo atributo mudou drasticamente o subconjunto de atributos utilizado pelo classificador induzido quando comparado com aquele induzido utilizando o conjunto original de exemplos.

6.6 Resumo dos Resultados

> Science is the great antidote to the poison of enthusiasm and superstition.
>
> —*Adam Smith*

Na Tabela 6.8 na página 112 é apresentado um resumo dos resultados obtidos através das três etapas por nós propostas. Nesta tabela é mostrada, para cada conjunto derivado de exemplos, a seguinte informação:

- A - nome do conjunto de exemplos;
- B - número total de atributos no conjunto de exemplos;

Número do Atributo	hepatitis C4.5R	hepatitis CN2	hepatitis$_1^{'}$ C4.5R	hepatitis$_1^{'}$ CN2
#1 (X_1)	○	○	○	○
#2 (X_2)	○		○	○
#3 (X_3)				
#4 (X_4)	○	○		
#5 (X_5)	○	○	○	
#6 (X_6)	○	○	○	
#7 (X_7)				
#8 (X_8)	○		○	○
#9 (X_9)	○	○	○	
#10 (X_{10})				
#11 (X_{11})	○	○	○	○
#12 (X_{12})	○		○	○
#13 (X_{13})				○
#14 (X_{14})				○
#15 (X_{15})				○
#16 (X_{16})	○	○	○	○
#17 (X_{17})	○		○	○
#18 (X_{18})	○	○		○
#19 (X_{19})				○
#20 ($X_1^{'}$)				•
Total #A	19	19	20	20
#A	12	8	10	13
%A	63,16%	42,11%	50,00%	65,00%
mean	21,29±2,99	18,25±3,83	18,00±3,74	17,50±2,04
ad			$-0,97$	$-0,24$

Tabela 6.7: Conjunto de exemplos hepatitis antes e após IC

- C - o primeiro número, entre parênteses, indica o número de atributos primitivos utilizados na construção do atributo derivado, seguido pelo número que identifica o atributo derivado (Derivado) assim como os atributos primitivos utilizados para construí-lo (Primitivos);
- D - atributos utilizados por C4.5RULES;
- E - atributos utilizados por CN2;
- F - indica se a precisão medida utilizando 10-*fold stratified cross-validation* melhorou utilizando C4.5RULES e/ou CN2 nos conjuntos derivados de exemplos. Se melhorou então isso é indicado pelo indutor que teve melhorias na precisão, ou seja, onde $\mathsf{ad}(T_i^{'} - T) < 0$.

Os atributos em negrito sublinhados correspondem aos atributos derivados. Embora existam melhorias na precisão, os resultados estariam na situação perfeita se durante a segunda etapa esses atributos originais utilizados para construir os derivados não fossem selecionados pelos indutores.

Na Tabela 6.8 na próxima página pode-se observar que dos 14 casos analisados (7 con-

1ª Etapa		2ª Etapa			3ª Etapa
A	B	C	D	E	F
Dataset		Derivado → Primitivos	C4.5RULES	CN2	$\mathsf{ad}(T_i^{'} - T) < 0$
$\text{pima}_1^{'}$	9	(2) 9 → 2 3	2 3 6 7 8 **9**	2 3 4 5 6 7 8	C4.5RULES
$\text{pima}_2^{'}$	9	(2) 10 → 2 5	2 3 5 6 7 8 **10**	1 2 3 4 5 6 7 8 **10**	
$\text{cmc}_1^{'}$	10	(2) 10 → 2 3	1 2 3 4 5 6 7 8 9	1 2 3 4 5 5 6 7 8 9 **10**	CN2
$\text{cmc}_2^{'}$	10	(2) 11 → 2 8	1 2 3 4 5 6 7 8 9 **11**	1 2 3 4 5 6 7 8 9 **11**	C4.5RULES
$\text{smoke}_1^{'}$	14	(4) 14 → 6 7 8 9	1 2 3 4 5 6 7 8 9 10 11 12 13	1 2 3 4 5 6 7 8 9 10 11 12 13 **14**	CN2
$\text{smoke}_2^{'}$	14	(3) 15 → 3 4 5	1 2 3 4 5 6 7 8 9 10 11 12 13 **15**	1 2 3 4 5 6 7 8 9 10 11 12 13 **15**	CN2
$\text{hepatitis}_1^{'}$	20	(3) 20 → 9 12 13	1 2 5 6 8 9 11 12 16 17	1 2 8 11 12 13 14 15 16 17 18 19 **20**	C4.5RULES CN2

Tabela 6.8: Resumo dos resultados

juntos de exemplos e 2 indutores), há 7 casos (50%) nos quais alguma melhoria na precisão foi obtida embora não com nível de confiança de 95%. Os conjuntos de exemplos $\text{smoke}_1^{'}$ e $\text{smoke}_2^{'}$ utilizando CN2 apresentaram os maiores ganhos em precisão, seguidos pelo conjunto de exemplos $\text{hepatitis}_1^{'}$ utilizando C4.5RULES. Por outro lado, os conjuntos de exemplos $\text{cmc}_1^{'}$ utilizando C4.5RULES e $\text{cmc}_2^{'}$ utilizando CN2 apresentaram uma degradação considerável na precisão. Nota-se que no conjunto de exemplos $\text{cmc}_1^{'}$ utilizando C4.5RULES, o atributo derivado não foi selecionado, mas, mesmo assim, ele causou uma perda de precisão. Como já mencionado, análises teóricas e experimentais indicam que muitos indutores não apresentam bom desempenho em domínios com muitos atributos irrelevantes ou redundantes (Langley 1996). Isso sugere que seleção de atributos e combinação de atributos possam ser utilizadas em conjunto.

Além disso, nenhum atributo derivado mostra um poder de descrição por si próprio. Se um atributo derivado aparece em um classificador, pode-se observar que pelo menos um dos atributos originais, utilizados para compô-lo, também está presente.

Como o número de atributos derivado é pequeno, decidiu-se prosseguir com a análise, removendo os atributos originais utilizados para compor os derivados (resultados não mostrados aqui). Nesta situação, apenas dois conjuntos de exemplos apresentaram uma melhoria na precisão sem

os atributos originais: smoke_2' utilizando C4.5RULES ($\mathsf{ad} = -0,60$) e CN2 ($\mathsf{ad} = -0,76$) assim como $\text{hepatitis}_1'$ utilizando CN2 ($\mathsf{ad} = -0,25$). Nesses casos, os classificadores não só obtiveram um melhor desempenho utilizando os atributos derivados como também, quando os atributos originais utilizados para compor os derivados foram removidos, a precisão ainda manteve-se acima daquela obtida utilizando apenas o conjunto original de atributos (Lee & Monard 2000).

6.7 Considerações Finais

> The aim of science is to seek the simplest explanation of complex facts. We are apt to fall into the error of thinking that the facts are simple because simplicity is the goal of our quest. The guiding motto in the life of every natural philosopher should be, 'Seek Simplicity and distrust it'.
>
> —*Alfred North*

Neste capítulo foi descrita a metodologia, por nós proposta, para Indução Construtiva baseada no conhecimento do domínio fornecido pelo usuário ou especialista. A metodologia consiste em três etapas: (i) criar e adicionar atributos derivados sugeridos pelo usuário/especialista; (ii) aplicar um indutor no conjunto derivado de exemplos e avaliar se ele aparece no classificador induzido e finalmente (iii) avaliar o aumento de precisão no classificador induzido devido à introdução do atributo derivado. O conjunto derivado de exemplos, e conseqüentemente o atributo derivado, que são aprovados em todas as etapas são selecionados para investigações futuras.

A situação ideal aconteceria quando os atributos primitivos, utilizados na construção dos atributos derivados, não fossem selecionados na segunda etapa. Todavia, isso pode não acontecer, uma vez que é possível que os atributos derivados não capturem perfeitamente, para o indutor escolhido, a informação embutida em cada atributo individual ou que o seu poder de predição seja equivalente a alguns dos atributos originais. Outra razão para isso pode ser devido ao fato que o próprio conjunto de exemplos já tenha sido pré-processado de forma tal que os atributos originais são, por si só, os mais relevantes.

Neste capítulo foram também mostrados alguns resultados empíricos de Indução Construtiva guiada pelo conhecimento. A precisão e o conjunto de atributos selecionados foram avaliados, fornecendo diferentes conjuntos de atributos aos indutores C4.5RULES e CN2. Um atributo é considerado relevante para o aprendizado se ele é utilizado por um desses algoritmos para induzir regras.

Os resultados mostram que, mesmo tendo o auxílio do usuário/especialista, não foi possível construir atributos derivados que tenham sido realmente relevantes para aprender o conceito

embutido nos conjuntos de exemplos estudados. Acreditamos que um dos motivos é que a metodologia proposta foi testada utilizando conjuntos de exemplos de repositórios (naturais), os quais, muitas vezes, já foram pré-processados. Esse fato foi confirmado, por um trabalho posterior, com dados do mundo real (Lee, Monard & Esteves 2000).

Um dos desafios que existe é como explorar e combinar algoritmos de AM eficientemente. Freqüentemente, uma grande ênfase é dada para o poder de predição do conhecimento induzido em novos exemplos, ou seja, na precisão. Embora a precisão seja uma meta importante, a compreensão e avaliação por seres humanos, do conhecimento induzido também possui um papel relevante que, normalmente, é negligenciado. Esses tópicos são abordados nos capítulos seguintes.

Capítulo 7

Melhorando o Desempenho dos Classificadores

> It is the mark of an educated mind to rest satisfied with the degree of precision which the nature of the subject admits and not to seek exactness where only an approximation is possible.
>
> —*Aristotle*

No Capítulo 1 foi introduzido o conceito de *bias* de Aprendizado de Máquina, o qual se constitui em certas suposições e escolhas efetuadas pelos indutores na busca de uma hipótese. Embora o *bias* possa ser visto como algo indesejável, ele é necessário para o processo de aprendizado. De fato, aprendizado sem *bias* é impossível.

Por outro lado, o termo '*bias*' é amplamente usado, com diferentes significados, nas áreas de Aprendizado de Máquina e estatística. Embora diferentes, existe uma relação entre eles, de forma que o *bias* e variância estatísticos podem ser aplicados para diagnosticar problemas com o *bias* de AM.

Além disso, o *bias* e a variância podem ser vistos como componentes do erro de um classificador. Assim, métodos que reduzam o *bias* ou a variância, ou ambos, constituem uma área de interesse em AM.

Um desses métodos, a construção de *ensembles*, consiste em induzir vários classificadores e então combiná-los em um único classificador final, através de um mecanismo de votação. Em geral, *ensembles* mostraram ser eficazes quando aplicados a indutores que são, de certa forma, instáveis, tais como indução de regras, árvores de decisão ou redes neurais. Esses e outros conceitos são tratados nas próximas seções.

7.1 Bias de Aprendizado de Máquina

> Without a bias, all possible functions must be entertained as hypotheses, and, together, these functions predict that all possible future outcomes are equally likely, so they cannot provide basis for generalization or predition.
>
> —*Thomas G. Dietterich (Dietterich & Kong 1997)*

Várias pesquisas em Aprendizado de Máquina afirmam que todo indutor deve adotar algumas suposições de forma a generalizar os exemplos de treinamento. Essas suposições são denominadas *biases* de Aprendizado de Máquina. Embora os *biases* de AM possam ser vistos como algo indesejável, pois limitam o conceito que pode ser aprendido pelo indutor, eles são essenciais para o processo de aprendizado. Segundo Mitchell (1982):

> "Mesmo que a remoção de todos os *biases* de um sistema de generalização possa ser vista como uma meta desejada, de fato o resultado é praticamente inútil. A habilidade de um sistema de aprendizado sem *bias* classificar novos exemplos não é melhor do que se ele simplesmente armazenasse todos os exemplos de treinamento e realizasse uma busca quando fosse necessário classificar um exemplo subseqüente ... Um sistema sem *bias* é aquele cujas inferências seguem logicamente a partir dos exemplos de treinamento, enquanto a classificação de novos exemplos não segue logicamente a classificação dos exemplos de treinamento."

Uma questão de interesse, por parte dos pesquisadores em Aprendizado de Máquina, consiste em como definir o *bias* de um algoritmo e como saber se um dado *bias* é apropriado, com base no conhecimento do domínio. De forma a compreender melhor o *bias* de AM, ele pode ser dividido em dois tipos, denominados *absoluto* e *relativo*, descritos a seguir:

Absoluto O *bias* absoluto (ou de restrição do espaço de hipóteses) ocorre quando o algoritmo de indução assume que a função objetivo (que é desconhecida) a ser aprendida é um elemento de um determinado conjunto de funções, ou seja, ela pertence a um espaço restrito de hipóteses. Normalmente, esta restrição é definida pela linguagem de representação adotada pelo indutor. Por exemplo, o algoritmo perceptron restringe a busca ao espaço de funções lineares, enquanto árvores de decisão univariadas traçam hiperplanos paralelos aos eixos.

Relativo O *bias* relativo (ou de preferência) ocorre quando o algoritmo de aprendizado introduz uma ordem de preferência nas hipóteses, assumindo que a função a ser aprendida é mais similar a um determinado conjunto de funções. Em várias situações, essa ordem é

definida pela forma segundo a qual a busca é conduzida através do espaço de hipóteses. Um grande parte dos *biases* relativos tenta minimizar alguma medida de complexidade, geralmente seguindo o princípio da navalha de Ockham de escolher hipóteses mais simples que sejam consistentes com os exemplos de treinamento (Kearns & Vazirani 1994). Por exemplo, os algoritmos de indução de árvores de decisão consideram árvores menores antes de considerar as árvores maiores. Se esses algoritmos encontram uma árvore pequena que classifica corretamente os dados de treinamento então uma árvore maior não é considerada.

Qualquer que seja o *bias* de um indutor, em geral, ele está *alinhado* ou em concordância com fatos do mundo em que vivemos. Por exemplo, os algoritmos de indução de árvores de decisão, em sua maioria, restringem o espaço de hipóteses para aquele formado por árvores finitas, cujas derivações são baseadas em (i) teste de igualdade ou de pertinência a um subconjunto de atributos discretos ou (ii) teste de valor limiar (um atributo é maior/menor que um determinado valor) para atributos contínuos. O teste de valor limiar, para atributos contínuos, é um exemplo de *bias* utilizado em muitos outros algoritmos de aprendizado.

Assim, o teste de um valor limiar alinha-se à experiência humana, a qual indica que o mundo real possui um comportamento, de certa forma, suave em muitos casos. Por exemplo, se a taxa máxima de batimentos cardíacos é relevante para prever se um paciente pode apresentar uma desordem cardíaca, espera-se que a classe dependa do fato do valor ser alto ou baixo e não dele ser par ou ímpar. De forma similar, se a temperatura de um paciente é relevante para determinar se ele possui alguma doença, espera-se uma certa suavidade, no sentido que pacientes com temperaturas próximas comportem-se de forma similar, se todos os demais fatores forem constantes.

7.2 Bias e Variância Estatísticos

> How do we tell a kid that not everything the computer says is right? Most of us know better, but we still have a longstanding trust in computers — they don't make mistakes, they don't have biases, they don't lie or cheat.
>
> —*Clifford Stoll*

O *bias* de aprendizado fornece uma especificação do comportamento desejado do algoritmo, assim como torna mais claro o projeto e implementação de algoritmos de AM. Em estatística, o termo '*bias*' é usado de forma mais precisa, contudo relacionada com o *bias* de aprendizado. Nesta seção são fornecidos alguns conceitos básicos sobre o *bias* e a variância estatísticos; maiores

detalhes podem ser encontrados em (Breiman 1996c; Dietterich & Kong 1997; Bauer & Kohavi 1999).

Formalmente, o *bias* de um método que estima um parâmetro θ é definido como o valor esperado $E[\hat{\theta}]$ menos o valor de θ, dado por (7.1).

$$\mathsf{bias}(\theta) = E[\hat{\theta}] - \theta \tag{7.1}$$

O *bias* estatístico captura a idéia de *erro sistemático* para um dado tamanho de amostra. Por exemplo, se a função real é uma onda senoidal $f(x) = \sin(x)$ e o algoritmo de aprendizado constrói hipóteses lineares $h(x) = ax+b$, então haverá erros sistemáticos, a cada subida e descida da onda senoidal.

Outro conceito estatístico relacionado ao *bias* estatístico é a variância. A variância de um algoritmo é definida como o valor esperado do quadrado da diferença entre o estimador do parâmetro θ e o seu valor esperado, dado por (7.2).

$$\mathsf{var}(\theta) = E\left[(\hat{\theta} - E[\hat{\theta}])^2\right] \tag{7.2}$$

A variância captura variações aleatórias no algoritmo, de uma amostra para outra. A variação pode ser devida à variação do conjunto de treinamento, de ruído aleatório nos exemplos ou pelo comportamento aleatório do próprio algoritmo de aprendizado, tais como os valores iniciais aleatórios que são freqüentemente utilizados nos pesos de uma rede neural.

Na realidade, tanto o *bias* como a variância estatísticos podem ser vistos como componentes do erro de uma hipótese, como é descrito a seguir.

7.2.1 Decomposição Bias-Variância

> This decomposition is useful in understanding the properties of predictors.
>
> —*Leo Breiman (Breiman 1996c)*

O princípio da *Decomposição Fundamental* de uma hipótese h afirma que seu erro pode ser decomposto em três componentes básicos (Breiman 1996c; Breiman 1996a):

1. o *erro mínimo*, que é o limiar inferior no erro esperado de qualquer indutor e que é obtido através da hipótese ideal, normalmente desconhecida;

2. o *bias*, que mede quão aproximadamente o indutor erra, em média, sobre todos os conjuntos de treinamento de um dado tamanho;

3. a *variância*, que mede quanto o erro do indutor varia em relação a cada amostra, ou seja, quão freqüentemente ele flutua para diferentes conjuntos de treinamento de um dado tamanho.

A relação entre esses três componentes básicos é dada por (7.3). Em problemas de classificação, $f = B^*$ é o classificador de Bayes, que atinge a taxa mínima de erro $\mathsf{err}(f) = \mathsf{err}(B^*)$. No caso de regressão, $\mathsf{err}(f)$ é o erro mínimo da regressão dado pela verdadeira hipótese (função) f. *Bias* e variância são sempre termos positivos[1]. Em alguns exemplos, o *bias* predomina, em outros a variância. Entretanto, em geral, em cada exemplo ambas as contribuições são positivas.

$$\mathsf{err}(h) = \mathsf{err}(f) + \mathsf{bias}(h) + \mathsf{var}(h) \tag{7.3}$$

Essa decomposição é importante para a compreensão do relacionamento entre *bias*-variância e o comportamento de um indutor. Em geral, um indutor constrói partições, no espaço de descrição, de forma tal que podem ser consideradas como uma família de funções H. Por exemplo, a maior parte dos indutores de árvores de decisão ou de indução de regras divide o espaço em regiões retangulares, enquanto redes neurais dividem o espaço em regiões mais complexas. Em qualquer caso, cada indutor tenta selecionar a melhor hipótese h, utilizando o conjunto de treinamento, a partir do conjunto de funções H.

Por exemplo, se a família de funções H, que podem ser geradas por um indutor, é um pequeno conjunto de funções lineares e a função objetivo f é não linear, então o *bias* de h será grande. Por outro lado, uma vez que um número pequeno de parâmetros são estimados a partir do pequeno conjunto H, a variância de h será pequena. Mas se H é uma grande família de funções, como as funções representadas por árvores de decisão ou redes neurais, então o *bias* é usualmente pequeno (uma vez que é quase sempre possível aproximar a função f por alguma $h \in H$), mas a variância pode ser grande (já que muitos parâmetros podem ser ajustados).

Embora a decomposição *bias*-variância seja interessante, existe uma limitação do seu emprego em exemplos do mundo real. De forma a estimar o *bias*, a variância e o erro mínimo para um problema particular, é necessário conhecer a verdadeira função objetivo f sendo aprendida. Essa informação, para a maioria dos dados provenientes do mundo real, não encontra-se disponível.

[1] Breiman (1996c) considera o quadrado do *bias* definido em (7.1)

Entretanto, é possível solucionar esse problema, mesmo que parcialmente, através de técnicas de amostragem (Bauer & Kohavi 1999; Opitz & Maclin 1999).

7.2.2 Estabilidade dos Indutores

> It would be illogical to assume that all conditions remain stable.
>
> —*Spock, The Enterprise Incident, stardate 5027.3*

Alguns indutores são *instáveis* no sentido que pequenas perturbações no conjunto de treinamento podem resultar em grandes alterações na hipótese induzida. Por exemplo, árvores de decisão em regressão ou classificação, redes neurais e seleção de atributos em regressão são instáveis. Assim, indutores instáveis são caracterizados pela alta variância: quando o conjunto de treinamento do mesmo domínio (uma amostra) é alterado, as hipóteses induzidas podem diferir consideravelmente umas das outras.

Já os indutores *estáveis*, por outro lado, não se alteram muito com pequenas mudanças no conjunto de treinamento. Por exemplo, análise de discriminantes lineares e K-NN são estáveis (Breiman 1996d). Existe um *trade-off* entre *bias* e variância: indutores instáveis usualmente têm uma grande variância mas podem apresentar um pequeno *bias*. Indutores estáveis possuem uma pequena variância, mas podem ter um grande *bias*.

7.3 Relação entre Bias de Aprendizado e Bias e Variância Estatísticos

> Do or do not, there is not 'try'.
>
> —*Yoda, Jedi Master, The Empire Strikes Back*

Como já mencionado, o *bias* de AM pode ser descrito em termos de um *bias* absoluto, no qual certas hipóteses são inteiramente eliminadas do espaço de hipóteses, e um *bias* relativo, quando certas hipóteses são preferidas em relação a outras. Em qualquer problema, um *bias* absoluto pode ser caracterizado como *apropriado* ou *inapropriado*. O espaço de hipóteses de um *bias* absoluto apropriado contém boas aproximações à função objetivo; um *bias* absoluto inapropriado não contém boas aproximações.

Um *bias* relativo pode ser caracterizado como *muito forte* ou *muito fraco*. Um *bias* muito forte é aquele que, mesmo que não exclua boas aproximações da função objetivo, prefere as

hipóteses menos representativas. Um *bias* que é muito fraco não focaliza o algoritmo de indução nas hipóteses apropriadas mas, ao invés disso, permite que ele considere muitas hipóteses.

Embora diferentes, os *biases* de AM e o *bias* e variância estatísticos estão relacionados, como pode ser observado na Tabela 7.1 (Dietterich & Kong 1997). Em geral, se o *bias* relativo de um algoritmo é muito forte então o algoritmo terá uma pequena variância; se ele for muito fraco então o algoritmo terá uma grande variância. Se o *bias* absoluto é inapropriado então o algoritmo terá um grande *bias* estatístico. Entretanto, o *bias* estatístico pode ser grande mesmo em situações nas quais o *bias* absoluto é apropriado: se o *bias* relativo for muito forte. Para conseguir ambos, *bias* e variância pequenos, o algoritmo deve ter um *bias* absoluto apropriado e um bom nível de força de expressão para o *bias* relativo.

Bias	Variância	Biases de Aprendizado	
Estatístico	Estatística	Absoluto	Relativo
grande	pequena	apropriado	muito forte
pequeno	pequena	apropriado	bom
pequeno	grande	apropriado	muito fraco
grande	pequena	inapropriado	muito forte
grande	média	inapropriado	bom
grande	grande	inapropriado	muito fraco

Tabela 7.1: Relação entre *biases* de aprendizado e *bias* e variância estatísticos

7.4 Redução de Bias e Variância

If we try to gild the lily by using both options together ...

—Quinlan (Quinlan 1993)

Uma vez que o *bias* e variância estatísticos são resultantes do projeto do algoritmo de aprendizado, qualquer alteração no algoritmo pode alterar o *bias* ou a variância. Alterações, que aumentem o poder de representação do algoritmo, podem reduzir tanto o *bias* de aprendizado como o *bias* estatístico. Alterações, que aumentem o número de alternativas do algoritmo, ou que tornem essas alternativas dependentes de uma pequena fração dos exemplos, podem aumentar a variância do algoritmo.

Por exemplo, na indução de árvores de decisão, existem diversos critérios de escolha do teste em um nó — entre eles, ganho ou razão de ganho de informação (Quinlan 1993) e índice Gini (Breiman, Friedman, Olshen & Stone 1984). Em geral, a escolha do teste pode ser alterada pela simples inclusão ou exclusão de um único exemplo de treinamento. Com isso, todos os testes subseqüentes nos nós descendentes são influenciados pelo teste no nó raiz. Conseqüentemente,

a alteração de um único exemplo de treinamento pode produzir uma cascata de alterações nos nós descendentes que alteram toda a estrutura da árvore induzida.

Algumas pesquisas sugerem que uma fonte de variância, na indução de árvores de decisão, seja resultante da escolha do teste em nós próximos às folhas bem como a atribuição da classe a uma dada folha. Normalmente, esses testes são baseados em uma quantidade pequena de exemplos e, assim, espera-se que apresentem uma alta variância. Para diminuí-la, a solução convencional adotada consiste na poda da árvore, removendo, assim, as folhas com grande variância (Dietterich & Bakiri 1995).

Embora alterações nos algoritmos de indução possam melhorar tanto os *biases* de AM como o *bias* e a variância estatísticos, não é viável implementar todas as situações possíveis em um único indutor. Fortuitamente, existe uma técnica geral que permite melhorar os *biases* e variância de um algoritmo de AM. Essa técnica consiste em construir um conjunto de classificadores e usá-los de forma combinada para predizer a classe de novos exemplos. Esse é o tópico da próxima seção.

7.5 Ensembles: Combinando Classificadores

> Ensembles are well-established as a method for obtaining highly accurate classifiers by combining less accurate ones. There are still many questions, however, about the best way to construct ensembles as well as issues about how to understand the decisions made by ensembles.
>
> —*Thomas G. Dietterich (Dietterich 1997b)*

Um *ensemble* h^* consiste em um conjunto de L classificadores individuais $\{h_1, h_2, \ldots, h_L\}$, cujas predições são combinadas para determinar o rótulo de um novo exemplo. Na sua forma mais simples, para um problema de k classes $\{C_1, C_2, \ldots, C_k\}$, a combinação é efetuada utilizando o voto majoritário, dado por (7.4). No voto majoritário, a classe prevista com maior freqüência por todos os classificadores $\{h_1, h_2, \ldots, h_L\}$ é a classe prevista pelo classificador final h^*. Para problemas de regressão, h^* é, normalmente, a média dos valores obtidos por cada hipótese individual, dada por (7.5).

$$h^*(x) = \underset{c \in \{C_1, C_2, \ldots, C_k\}}{\arg\max} \sum_{l=1}^{L} \| \, h_l(x) = c \, \| \tag{7.4}$$

$$h^*(x) = \frac{1}{L} \sum_{l=1}^{L} h_l(x) \tag{7.5}$$

Freqüentemente, um *ensemble* é mais preciso que qualquer uma das hipóteses individuais contida nele, isto é, o uso de múltiplas hipóteses (para classificação ou regressão) mostra uma melhoria na precisão ao rotular exemplos que não se encontram no conjunto de treinamento.

Um *ensemble* pode ser mais preciso que cada um dos classificadores que o compõe somente se cada classificador individual discorda de um outro. Uma explicação simples para esse fato é dada pelo seguinte exemplo. Suponha que um *ensemble* seja composto por três classificadores $\{h_1, h_2, h_3\}$ e considere um novo exemplo x a ser rotulado. Se os três classificadores são idênticos, então quando $h_1(x)$ erra, $h_2(x)$ e $h_3(x)$ também erram. Todavia, se os erros dos três classificadores não estão correlacionados, então quando $h_1(x)$ está errado, $h_2(x)$ e $h_3(x)$ podem estar certos. Se isso ocorrer, através do voto majoritário, o novo exemplo x será corretamente classificado pelo *ensemble*[2].

Existe uma diversidade considerável de métodos usados para compor *ensembles*, alguns dos quais efetuam a manipulação dos atributos (por exemplo, cada classificador individual tem acesso a um subconjunto dos atributos originais); a manipulação da classe (por exemplo, problemas com muitas classes podem ser vistos como vários problemas com classes binárias); inserção de um componente aleatório (por exemplo, os pesos aleatórios definidos inicialmente em uma rede neural ou a escolha entre, diga-se, 20 dos melhores testes para um nó em uma árvore de decisão); uso de amostragem, entre outros. Nas seções seguintes são descritos *ensembles* obtidos através dessa última técnica. Uma revisão mais detalhada sobre *ensembles* pode ser encontrada em (Baranauskas & Monard 2000c; Opitz & Maclin 1999; Dietterich 1997b).

7.5.1 Window

Windowing pode ser visto como um precursor dos *ensembles*, já que ele não combina vários classificadores em um classificador final mas obtém um classificador final através da iterações sucessivas, descartando os classificadores anteriores. *Windowing* seleciona um subconjunto de exemplos (uma *window*) do conjunto de treinamento e induz uma hipótese h_1 a partir dele (Quinlan 1993). A hipótese h_1 é então usada para classificar os exemplos remanescentes do conjunto de treinamento, isto é, aqueles exemplos que não foram incluídos na *window*. Se existem exemplos classificados incorretamente, uma seleção deles é adicionada à *window* inicial e uma segunda hipótese h_2 é construída, a partir da *window* aumentada. Este ciclo é repetido até que a hipótese, construída a partir da *window* atual, classifique corretamente todos os exemplos de treinamento fora da *window* ou o número de ciclos exceda um valor L pré-definido.

[2]Se os erros de L classificadores forem independentes e menores que $1/2$ então a probabilidade que o voto majoritário erre é dada pela área sob a distribuição binomial na qual $L/2$ classificadores estão errados.

7.5.2 Stack

No método *Stacking*, as descrições dos exemplos de treinamento são estendidas para incluir os resultados da classificação de exemplos com uma seleção inicial de classificadores. As novas descrições são então analisadas para compor novos classificadores e assim por diante (Wolpert 1992).

Mais precisamente, suponha L diferentes algoritmos $A_1, A_2, \ldots, A_L$ e um conjunto de n exemplos $\{(x_1, y_2), (x_2, y_2), \ldots, (x_n, y_n)\}$. Cada algoritmo é aplicado ao conjunto de treinamento, induzindo os classificadores $\{h_1, h_2, \ldots, h_L\}$. Após induzido, cada classificador é usado para rotular os exemplos. Isso implica que, para cada exemplo x_i, uma nova tupla é formada, composta pela classe obtida por cada uma das hipóteses e a classe verdadeira do exemplo, isto é, $(h_1(x_i), h_2(x_i), \ldots, h_L(x_i), y_i)$. Essas tuplas constituem o conjunto de treinamento de nível 2, cujos atributos são as classes previstas por cada um dos L classificadores. Com isso, é possível aplicar um outro algoritmo de aprendizado aos exemplos de nível 2 para aprender h^*.

7.5.3 Bagg

Bagging — Bootstrap Aggregation — extrai uma amostra *bootstrap* S_1 do conjunto de treinamento e induz uma hipótese h_1 (Breiman 1996b). Este ciclo é repetido L vezes para cada amostra *bootstrap* $S_2, S_3, \ldots, S_L$ induzindo hipóteses $h_2, h_3, \ldots, h_L$, respectivamente. Depois disso, todas as hipóteses induzidas são combinadas em uma hipótese final h^* utilizando o voto majoritário, definido por (7.4), na página 122. *Bagging* mostrou ser mais efetivo em indutores instáveis, nos quais, como já mencionado, pequenas alterações no conjunto de treinamento resultam em grandes alterações nas hipóteses induzidas.

7.5.4 Wagg

Wagging — Weight Aggregation — é similar a *bagging* mas, neste caso, o peso dos exemplos no conjunto de treinamento é alterado, ao invés de retirar-se uma amostra. De fato, este método perturba repetidamente o conjunto de treinamento assim como *bagging* mas, ao invés de amostrá-lo, *wagging* adiciona um ruído gaussiano para cada peso com média zero e um desvio padrão fixo, usualmente dois. O ciclo inicial começa com todos os pesos dos exemplos iguais e então ruído é adicionado aos pesos para, posteriormente, induzir a hipótese (Bauer & Kohavi 1999).

7.5.5 Boost

Boosting foi introduzido por Schapire (1990) como um método para melhorar o desempenho de um sistema de aprendizado. Após algumas melhorias, o método AdaBoost — *Adaptive Boosting* — foi introduzido por Freund & Schapire (1995), algumas vezes denominado AdaBoost.M1 (Freund & Schapire 1996).

No algoritmo de *boosting*, cada exemplo de treinamento possui um peso associado. A cada ciclo de iteração, uma hipótese é induzida a partir dos exemplos ponderados. Então, cada exemplo de treinamento tem seu peso associado modificado, considerando se ele foi ou não classificado corretamente pela hipótese induzida. Assim, diferentemente do algoritmo de *bagging*, o qual pode induzir hipóteses em paralelo, o algoritmo de *boosting* induz hipóteses seqüencialmente.

Ao induzir a primeira hipótese, todos os exemplos são considerados equiprováveis, ou seja, cada exemplo possui o mesmo peso $1/n$ associado a ele, onde n é o número de exemplos. Depois de induzir a primeira hipótese, *boosting* altera os pesos dos exemplos de treinamento, fato também denominado *reweighting*, com base nas hipóteses induzidas anteriormente. Este ciclo é repetido L vezes. A hipótese final, calculada por (7.6), é formada utilizando o esquema de voto ponderado, no qual o peso de cada hipótese depende de seu desempenho no conjunto de treinamento usado para construí-la.

$$h^*(x) = \underset{c \in \{C_1, C_2, \ldots, C_k\}}{\arg\max} \sum_{l=1}^{L} \log\left(\frac{1 - \mathsf{err}(h_l)}{\mathsf{err}(h_l)}\right) \parallel h_l(x) = c \parallel \tag{7.6}$$

O algoritmo AdaBoost requer um indutor cujo erro seja limitado por uma constante estritamente menor que 1/2 (Freund & Schapire 1995). Existem muitas revisões do algoritmo AdaBoost. Algumas das implementações de *boosting* utilizam amostragem, uma vez que alguns indutores não são capazes de trabalhar com exemplos ponderados (Freund & Schapire 1996). Algumas evidências mostram que *reweighting* funciona melhor na prática, uma vez que, ao indutor, são mostrados todos os exemplos (com pesos) ao invés de apenas uma amostra (Quinlan 1996).

Duas outras revisões especiais são sugeridas por Breiman (1996c) e Breiman (1996a). Uma delas indica que se o erro da hipótese $\mathsf{err}(h_l)$ torna-se igual ou maior que 1/2, resultados melhores podem ser obtidos alterando todos os pesos para $1/n$ e reiniciando. Além disso, se $\mathsf{err}(h_l) = 0$, o que significa que a etapa subseqüente do algoritmo será indefinida, novamente todos os pesos devem ser alterados para $1/n$. Outra mudança implícita está relacionada ao uso de amostragem ao invés de *reweighting*. Este é um ponto importante uma vez que, para indutores determinísticos,

tais como regras os árvores de decisão, alterar todos os pesos iguais a $1/n$ induziria a mesma hipótese novamente.

Bauer & Kohavi (1999) utilizam *reweighting*, como sugerido no algoritmo AdaBoost original, mas quando $\mathsf{err}(h_l) \geq 1/2$ ou $\mathsf{err}(h_l) = 0$, uma amostra *bootstrap* é extraída com pesos iguais a $1/n$ para todos os exemplos e, no máximo, $L = 25$ hipóteses são construídas; também são reportados problemas de precisão para os pesos, os quais podem ser resolvidos alterando a distribuição dos pesos originais para somar n ao invés de 1.

Algumas pesquisas mostraram que ambos *bagging* e *boosting* reduzem o erro pela redução da variância (Breiman 1996c). Freund & Schapire (1996) argumentam que *boosting* também tenta reduzir o erro pela redução do *bias*, uma vez que ele se concentra nos exemplos incorretamente classificados. Sob esse ponto de vista, *boosting* pode construir uma função que pode nunca ser produzida por cada um dos classificadores que o compõe, por exemplo, mudar classificadores lineares para não lineares. Em um trabalho mais recente, Bauer & Kohavi (1999) mostraram que *boosting*, de fato, reduz o *bias* em alguns problemas do mundo real. Surpreendentemente, eles também mostraram que *bagging* pode também reduzir o *bias*, freqüentemente, para os mesmos conjuntos de exemplos para os quais *boosting* também o reduz.

Em geral, *bagging* é quase sempre mais preciso que uma única hipótese, mas em alguns casos é menos preciso que *boosting*. Por outro lado, *boosting* pode criar *ensembles* que são menos precisos que uma única hipótese. Nessa situaçãos, *boosting* pode causar um *overfitting* em conjunto de exemplos com muito ruído, diminuindo assim seu desempenho.

7.5.6 Arc

Arcing — Adaptively Resample and Combine — é um termo definido por Breiman (1996a) para descrever a família de algoritmos que retiram amostras e combinam de forma adaptativa, incluindo o algoritmo AdaBoost que ele denomina arc-fs (em honra a Freund & Schapire) como um exemplo primário de um algoritmo *arcing*. Breiman também contrasta arcing com a família P&C (Perturbar e Combinar), da qual *bagging* é um exemplo primário.

O algoritmo arc-x4 foi descrito por Breiman como sendo tão preciso como arc-fs sem utilizar o esquema de ponderação para compor h^* e então construindo o classificador arc-fs final. Considerando isso, o poder do algoritmo AdaBoost é derivado da ponderação adaptativa dos exemplos e não devido à forma da combinação final (Breiman 1996c).

7.6 Necessidade de Ensembles

I know nothing except the fact of my ignorance.

—Socrates

Como mencionada, a melhoria de desempenho, proporcionada pelo uso de *ensembles*, é explicada pelo fato que erros, feitos pelos classificadores individuais, podem ser removidos pelo processo de votação. Existem algumas explicações prováveis para o fato que não é possível encontrar um classificador único que tenha um desempenho tão bom quanto um *ensemble* (Dietterich 1997b).

A primeira explicação da necessidade de *ensembles* é que os dados de treinamento podem não conter informação suficiente para a escolha da melhor e única hipótese h a partir do espaço completo de hipóteses H. A maioria dos algoritmos de AM considera espaços de hipóteses muito grandes e, dessa forma, mesmo eliminando hipóteses que classificam incorretamente os exemplos de treinamento, ainda restam muitas outras hipóteses. Essas hipóteses restantes aparentam ser igualmente precisas com relação aos dados de treinamento. A combinação desse conjunto de hipóteses restante em *ensembles*, normalmente, permite que diferentes regiões do espaço de exemplos possam ser cobertas.

Uma segunda explicação é que os algoritmos de aprendizado podem não ser capazes de resolver um problema difícil, que é exposto a eles. Por exemplo, o problema de encontrar a menor árvore de decisão, que seja consistente com os exemplos de treinamento, é um problema NP. Na prática, os algoritmos de indução de árvores de decisão empregam heurísticas para guiar o processo de busca por árvores menores. De forma similar, encontrar os pesos da menor rede neural, consistente com os exemplos de treinamento, é também um problema NP. Assim, os algoritmos de redes neurais empregam métodos de busca local, tais como gradiente descendente, para encontrar pesos ótimos locais. Uma conseqüência dessa imperfeição na busca é que — mesmo se a combinação dos exemplos de treinamento, com o conhecimento prévio do domínio, determinem a melhor e única hipótese — pode ser impossível encontrá-la. Portanto, *ensembles* podem ser vistos como uma forma de compensar a imperfeição de busca dos indutores.

Uma última explicação é que o espaço de hipóteses H pode não conter a função objetivo f. Ao invés disso, H pode incluir várias aproximações igualmente boas para f. Pela combinação dessas aproximações é possível representar classificadores que estejam fora de H. Uma forma de entender isso consiste em visualizar as fronteiras de regiões construídas pelos algoritmos de Aprendizado de Máquina. Como já descrita, uma região é uma superfície, tal que os exemplos que se localizam dentro são rotulados com uma classe diferente das classes dos exemplos que

se localizam do lado de fora da região (vide Figura 2.2 na página 25). Por exemplo, as regiões construídas por árvores de decisão univariadas são hiperplanos paralelos aos eixos. Se a fronteira entre duas classes for um hiperplano oblíquo aos eixos, os indutores de árvores de decisão devem aproximar o hiperplano oblíquo por uma série de hiperplanos paralelos aos eixos (formando uma 'escada'), similar à mostrada na Figura 7.1.

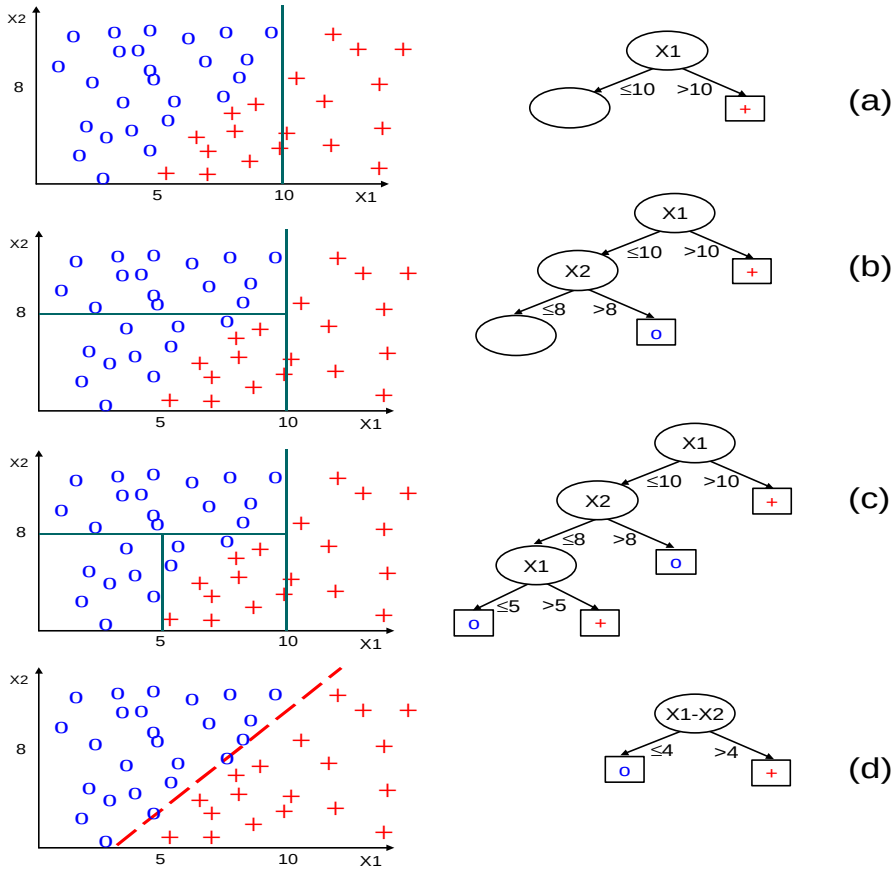


Figura 7.1: Regiões disjuntas são formadas por uma árvore de decisão. A construção de uma árvore univariada em (a), (b) e (c) divide o espaço em hiperplanos paralelos aos eixos, aproximando a região real que é um hiperplano oblíquo, dado pela árvore multivariada (d)

Essas aproximações em escada são equivalentes a uma árvore de decisão muito co pl xa co o xe pl fic a n Fi ur 7. na r´xi a pag na En re an o, ar in uz r e sa ar or t˜o e te sa no sp ç de ipot se H, seriam necessários muitos exemplos de treinamento, que nem sempre estão disponíveis. Dessa forma, o uso de *ensembles* pode propiciar uma forma de ultrapassar as deficiências de representação do espaço de hipóteses.

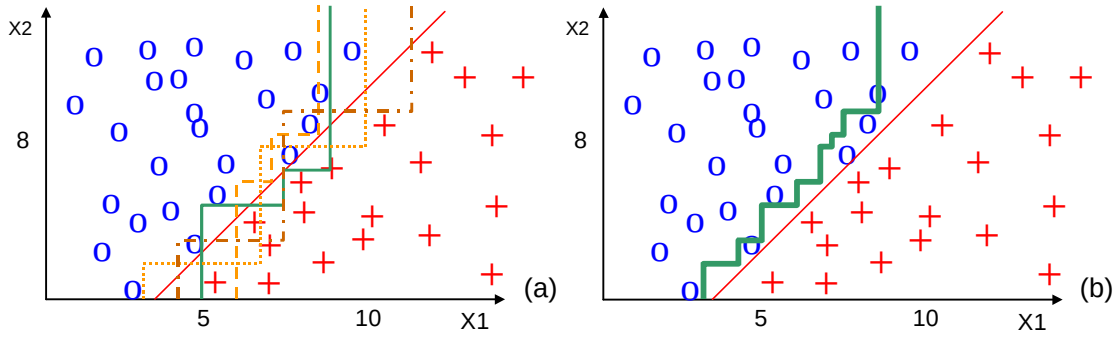


Figura 7.2: A região real de separação de classes, indicada pela linha diagonal, é aproximada de formas diferentes por quatro árvores de decisão em (a); a combinação das quatro árvores, em um *ensemble*, fornece uma região que se aproxima mais da diagonal em (b)

7.7 Considerações Finais

No, I'm not interested in developing a powerful brain. All I'm after is just a mediocre brain.

—*Alan Turing*

Para obter êxito na utilização das técnicas de Aprendizado de Máquina, deve-se investigar estruturas diferentes que podem ser apropriadas para diferentes contextos bem como entender seu poder e limitação. Aprendizado de Máquina é eficaz, em várias aplicações práticas, porque os conceitos meta não são equiprováveis e os algoritmos são, de certa forma, alinhados com fenômenos do mundo real: os atributos são usualmente selecionados por especialistas e certas suposições que os algoritmos assumem se aplicam em várias situações.

O erro de um classificador pode ser decomposto em três componentes básicos: o erro mínimo dado pelo classificador de Bayes, o *bias* e a variância estatísticos. Assim, métodos que reduzam o *bias* ou a variância, ou ambos, tendem a melhorar o desempenho de um classificador. Como existe uma relação entre *bias* e variância estatísticos e os *biases* de AM, para reduzir o *bias* e variância é importante escolher um indutor com *bias* absoluto apropriado e um bom *bias* relativo. Entretanto, essa escolha não é tão simples, pois não existe um único indutor que apresente o melhor desempenho em todos os domínios (Schaffer 1994; Kohavi, Sommerfield & Dougherty 1996).

Além da escolha do indutor (ou indutores) mais apropriado para cada situação, uma outra melhoria no desempenho pode ser obtida através da construção de *ensembles* de classificadores. A melhoria resultante pode ser explicada pelo fato que os *ensembles* têm a capacidade de ampliar as linguagens de representação de hipóteses, as quais nunca poderiam ser obtidas pelos

classificadores individuais que o compõem.

Apesar do ganho em desempenho que geralmente é proporcionado pelo uso de *ensembles*, se os classificadores individuais forem simbólicos, a compreensibilidade do classificador final, por seres humanos, fica extremamente prejudicada. A compreensibilidade, na combinação de classificadores simbólicos, é o tópico do capítulo que se segue.

Capítulo 8

O Sistema XRULER

> It is only the limitations of the human mind that make the possible, impossible.
>
> —*Marc Drake*

Como já mencionado, uma das preocupações do aprendizado simbólico é que os classificadores induzidos devem ser compactos e fáceis de serem compreendidos pelos seres humanos. Para isso, deve-se escolher o indutor com *bias* de AM mais adequado para cada tipo de situação, já que pesquisas mostraram que não existe o *melhor* indutor para todos os domínios.

Aliada a essa escolha, é possível fazer uso de *ensembles*, os quais possuem a tendência de melhorar o desempenho na classificação de exemplos não vistos durante o processo de aprendizado. Entretanto, o emprego de *ensembles* claramente dificulta a compreensão humana sobre o comportamento do classificador final. Talvez seja esse o motivo que o AM simbólico simplesmente evite o emprego de *ensembles*. Na realidade, a combinação de classificadores simbólicos em um classificador final também simbólico é um tópico sobre o qual recentemente estão sendo realizadas pesquisas. Com o objetivo de preencher essa lacuna, é apresentado, nas próximas seções, o sistema XRULER, por nós desenvolvido.

8.1 Motivação

> The truth is not always the same as the majority decision.
>
> —*Pope John Paul II*

Como visto no capítulo anterior, o emprego de *ensembles*, em geral, propicia um ganho de precisão, reduzindo *bias* e variância. Um exemplo disso é reportado por Margineantu &

Dietterich (1997), utilizando o conjunto de exemplos letter, atingindo um bom desempenho no conjunto de teste pelo voto de 200 classificadores, sendo que cada classificador induzido ocupa 295 Kbytes. Entretanto, o conjunto de exemplos letter requer menos de 700 Kbytes (de fato, menos de 370 Kbytes sem delimitadores). Portanto, um *ensemble* de 200 classificadores requer 58 Mbytes, mais de 85 vezes maior que o conjunto de exemplos (ou 164 vezes, considerando o conjunto de exemplos sem delimitadores). Assim, normalmente, um *ensemble* é muito grande.

Por outro lado, suponha que três classificadores simbólicos $\{h_1, h_2, h_3\}$ sejam induzidos e, através do mecanismo de voto majoritário, eles sejam combinados no classificador final h^*. Por serem simbólicos, os classificadores, h_1, h_2 e h_3 podem ser convertidos, cada um, em um conjunto de regras. Entretanto, não é possível converter o classificador final h^* da mesma forma. Assim, outro problema, associado ao uso de *ensembles*, é que a combinação de classificadores simbólicos resulta em um classificador final não simbólico, dificultando, assim, sua compreensão por um ser humano.

Outra questão que resulta do uso de *ensembles* é sua redundância, ou seja, é provável que um classificador apresente uma parte do conhecimento induzido idêntica a outro classificador do *ensemble*. Por exemplo, se um *ensemble* for composto por três árvores de decisão $\{h_1, h_2, h_3\}$, então é possível que um ramo da árvore de h_1 seja idêntico, ou muito similar, a outro ramo em h_2 ou h_3. Situações como essa tornam *ensembles*, freqüentemente, muito redundantes.

Esses três aspectos — tamanho, interpretação e redundância — trazem um valor agregado indesejável aos *ensembles*, sob o ponto de vista de compreensibilidade humana. Para isso, métodos alternativos são necessários, os quais, mesmo propiciando uma redução do *bias* e variância, também mantenham o componente simbólico no classificador final. De acordo com Dietterich & Kong (1997):

> "Algum método é necessário para converter uma combinação de árvores (ou outras hipóteses mais complexas) em uma hipótese menor equivalente. Estas árvores são muito redundantes; como podemos remover essa redundância, enquanto ainda reduzindo *bias* e variância?"

Esse constitui o tema principal das próximas seções, nos quais propomos uma possível resposta à pergunta anterior.

8.2 O Sistema RULER

> When Kepler found his long-cherished belief did not agree with the most precise observation, he accepted the uncomfortable fact. He preferred the hard truth to his dearest illusions, that is the heart of Science.
>
> —*Carl Sagan*

Uma possível resposta à questão do final da seção anterior consiste na combinação de classificadores simbólicos, de forma tal que o classificador final também seja simbólico, diferentemente do que ocorre no caso de *ensembles*. Isso requer que os classificadores sejam vistos sob a perspectiva de caixa-branca, ou seja, como um conjunto de regras.

Essa idéia foi utilizada no projeto SKICAT, que obteve grande sucesso na área de Mineração de Dados (Fayyad, Djorgovski & Weir 1996). Nesse projeto, os exemplos, resultantes do 2º levantamento cósmico do observatório de Palomar (POSS-II), são provenientes de 3000 chapas fotográficas. Cada chapa possui 23040 x 23040 pontos, com 16 bits de cor, totalizando mais de 3 terabytes, com cerca de 5×10^7 galáxias e 2×10^9 objetos cósmicos. Os atributos foram obtidos utilizando multi estratégia de indução construtiva: 40 atributos primitivos foram obtidos pelo sistema de processamento de imagens astronômicas FOCAS (Jarvis & Tyson 1981); 4 atributos derivados e normalizados a partir dos primitivos e 2 atributos requerendo medidas empíricas, obtidos pela aplicação de um algoritmo de AM.

O sistema RULER , mostrado na Figura 8.1 na página seguinte, foi utilizado na Extração de Conhecimento a partir desses exemplos. Em cada iteração, RULER particiona aleatoriamente uma amostra dos exemplos em um conjunto de treinamento e em um conjunto de teste. O conjunto de treinamento é fornecido para o indutor O-BTREE que constrói uma árvore de decisão, estritamente binária e não podada (Fayyad & Irani 1992).

Ao invés de podar a árvore de decisão induzida em cada iteração, RULER adota uma estratégia diferente. A árvore induzida é transformada em regras que são avaliadas utilizando o conjunto de teste. Dada uma regra na forma $L \rightarrow R$, RULER avalia cada condição existente na parte esquerda L pela relevância em relação à conclusão R, dada pela parte direita da regra. Condições que são consideradas irrelevantes são removidas, ou seja, as regras são podadas.

Esse processo resulta em um grande número de regras redundantes obtidas a partir de várias árvores. RULER separa as melhores regras de cada árvore e descarta a maioria das regras que são o resultado de correlações fracas de suporte nos dados. Então, um algoritmo guloso de cobertura é empregado para selecionar um conjunto mínimo de regras que cobre os exemplos. Segundo Fayyad, Djorgovski & Weir (1996), o sistema RULER produz tipicamente um conjunto

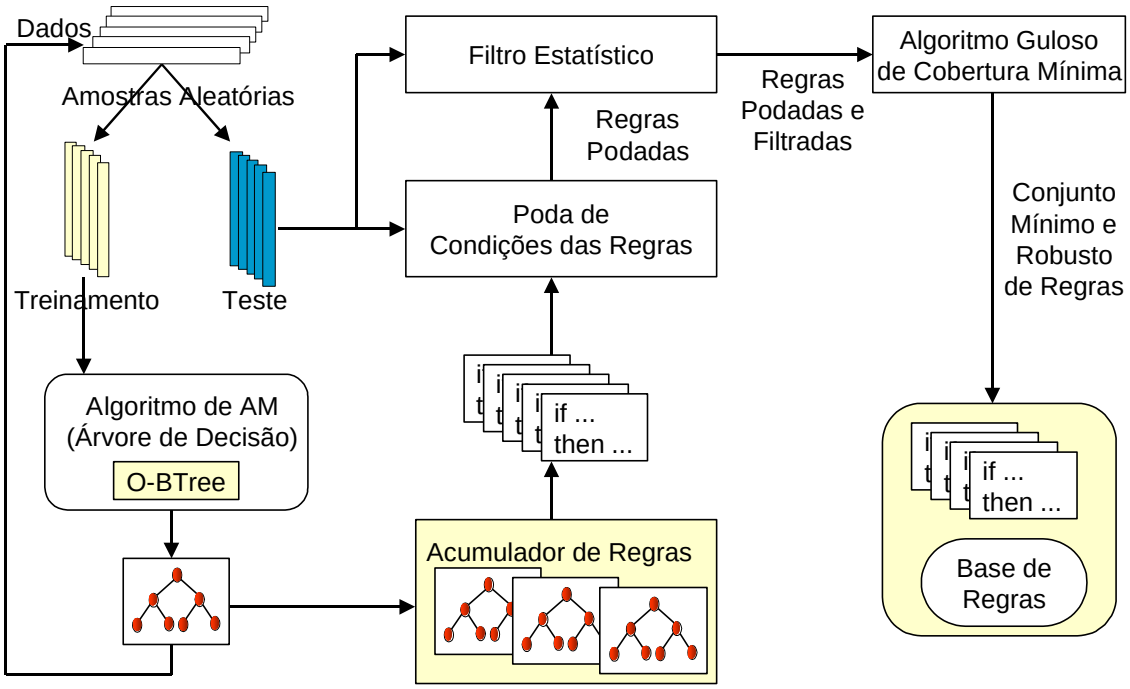


Figura 8.1: O sistema RULER utilizado no projeto SKICAT

robusto de regras contendo menos regras do que qualquer uma das árvores de decisão usadas para criá-lo. Além disso, RULER também mostrou ser mais preciso do que astrônomos humanos na classificação de objetos cósmicos a partir de chapas fotográficas.

A Tabela 8.1 mostra os resultados obtidos por Fayyad, Djorgovski & Weir (1996) para o sistema RULER (utilizando o indutor O-BTREE) e os indutores GID3* (Fayyad 1994), O-BTREE e ID3. Embora os desvios padrões correspondentes não sejam fornecidos pelos autores, comparando o erro 5,80% de RULER com 8,80% de O-BTREE aplicado isoladamente, isso representa uma redução relativa na taxa de erro de 51,72%, o que é bem significativo.

Algoritmo	Precisão
RULER	94,20%
GID3*	90,10%
O-BTREE	91,20%
ID3	75,60%

Tabela 8.1: Medidas de precisão reportadas no projeto SKICAT

Entretanto, existem algumas questões importantes a respeito do sistema RULER, tais como:

1. Caso RULER fosse aplicado a outros domínios, seria possível esperar a mesma tendência dos resultados obtidos no único domínio em que foi aplicado?

2. Quais seriam os resultados, caso RULER fosse comparado com um indutor geralmente utilizado como base de comparação em várias pesquisas de AM, tal como C4.5, aplicado isoladamente? Ou seja, os resultados obtidos dependem especificamente do indutor O-BTREE?

3. O que aconteceria se O-BTREE fosse substituído por um indutor que efetuasse a poda da árvore (sem que fosse necessário podar as regras) ou mesmo um indutor que induzisse regras diretamente?

4. Seria possível combinar diversos classificadores, provenientes de indutores diferentes?

A partir desses questões, surgiu então nossa proposta do sistema XRULER, descrita a seguir.

8.3 O Sistema XRULER

> The aim of science is to seek the simplest explanation of complex facts. We are apt to fall into the error of thinking that the facts are simple because simplicity is the goal of our quest. The guiding motto in the life of every natural man should be, 'seek simplicity and distrust it'.
>
> —*Alfred North*

A proposta apresentada neste trabalho tem como objetivo estender a metodologia empregada no sistema RULER. Tal extensão é denominada XRULER (*eXtended* RULER) e consiste em permitir o emprego de vários indutores, ao invés de um único, no processo de extração dos classificadores simbólicos.

Na Figura 8.2 na página seguinte é mostrada a idéia geral do sistema XRULER. Inicialmente, os exemplos são particionados nos conjuntos de aprendizado e de avaliação. Utilizando o conjunto de aprendizado, são criadas várias amostras de treinamento e teste. As amostras de treinamento são submetidas aos indutores que geram classificadores simbólicos. Os classificadores obtidos são avaliados nas amostras de teste correspondentes, calculando-se a matriz de contingência para cada regra. Após isso, filtros podem ser aplicados de forma a remover regras estatisticamente não significativas. Então, tem início o processo de cobertura, o qual utiliza a amostra de aprendizado, tentando encontrar um subconjunto de todas as regras induzidas. Finalmente, o conjunto mínimo de regras é avaliado utilizando o conjunto de avaliação, o qual nunca foi visto por XRULER durante a fase de aprendizado.

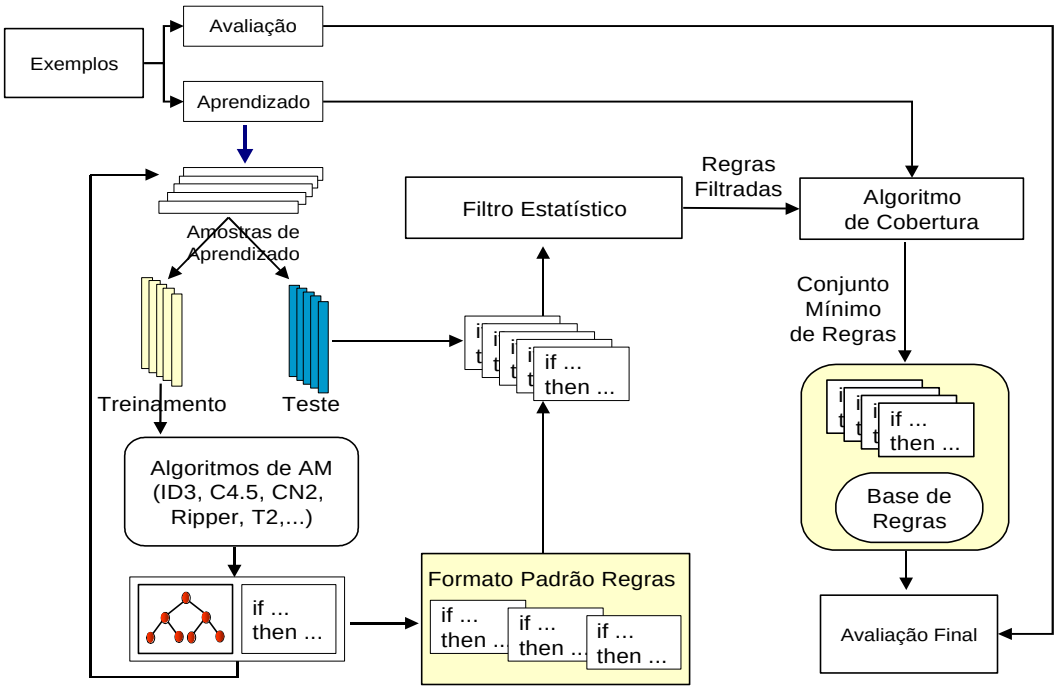


Figura 8.2: O sistema XRULER proposto neste trabalho

8.3.1 Componentes

O sistema XRULER é formado por oito componentes, ou objetos, cada um responsável pela realização de uma atividade específica. Dentre os componentes, tem-se:

- importer, importa o conjunto de exemplos no formato definido por Batista (2001) para o formato do sistema XRULER;
- splitter, separa os exemplos em duas amostras (partições) disjuntas, uma para aprendizado e outra para avaliação final;
- sampler, define, a partir da partição de aprendizado, as amostras de treinamento e teste. Inicialmente, foi implementado *bootstrap* como padrão, embora outras técnicas de amostragem possam ser introduzidas sem afetar os demais componentes do sistema;
- classifier, responsável em aplicar os algoritmos de indução às amostras de treinamento, obter os classificadores simbólicos associados e transformá-los no formato padrão de regras definido na Seção 8.3.2. Inicialmente, foram considerados os indutores C4.5 e CN2 em XRULER, sendo que a inclusão de outros indutores já encontra-se prevista;
- contingency, calcula a matriz de contingência associada a cada classificador induzido e já transformado no formato padrão de regras. Caso não seja especificada qual a amostra a ser

usada, este componente utiliza a amostra de teste para calcular a matriz de contingência das regras, mas é possível informar a ele para que utilize a amostra de treinamento ou de aprendizado;

- filter, após calculada a matriz de contingência, filtros podem ser opcionalmente aplicados às regras induzidas, com o objetivo de eliminar regras com propriedades indesejáveis, tais como baixo suporte, novidade, etc. As regras filtradas não são consideradas pelo componente cover;
- cover, responsável pela realização da cobertura, utilizando todo o conjunto de aprendizado, resultando em um conjunto final das regras induzidas;
- evaluator, usado para obter estatísticas sobre precisão, número de regras e número de condições por regras do conjunto final de regras obtido pelo emprego do componente cover.

Em geral, os componentes de XRULER são aplicados na ordem descrita anteriormente, ou seja, iniciando-se pelo componente splitter até evaluator, já que o componente importer é utilizado uma única vez para importar o conjunto de dados para o formato de XRULER.

Ainda que a implementação desenvolvida não faça uso de paralelismo, XRULER é flexível e capaz de assimilar seu emprego em diversos componentes. Por exemplo, as amostras podem ser extraídas em paralelo pelo componente sampler, os classificadores podem ser induzidos em paralelo pelo componente classifier, assim como o cálculo da tabela de contingência e a aplicação de filtros pelos componentes contingency e filter, respectivamente. Embora o algoritmo de cobertura de regras (vide Seção 8.3.3) seja basicamente seqüencial na escolha da melhor regra, o cálculo dos valores que permitem avaliar cada regra (vide Seção 8.3.4) pode ser efetuado em paralelo. Para grandes conjuntos de dados, tipicamente encontrados em DM, o paralelismo pode ser crucial para a obtenção de resultados de forma mais rápida.

Além disso, XRULER foi concebido como um software que pudesse ser utilizado livremente pela comunidade científica, em várias plataformas, utilizando software de domínio público. Os componentes de XRULER foram escritos na linguagem PERL (Wall, Christiansen & Schwartz 1996). Os exemplos, amostras, classificadores, regras e demais resultados são mantidos em MySQL, uma base de dados relacional (MySQL 2000).

Todavia, um problema que surge quando se utilizam diversos algoritmos de aprendizado que induzem classificadores simbólicos, é que a sintaxe da linguagem utilizada para descrever a hipótese induzida é diferente para cada classificador obtido. Para solucionar essa situação, foi proposto o formato padrão de regras, descrito a seguir, para o qual todo classificador simbólico é convertido.

8.3.2 Formato Padrão de Regras

O formato padrão de regras, denominado PBM, foi definido num trabalho conjunto entre Prati, Baranauskas & Monard (2001b). Na Tabela 8.2 é mostrada a gramática G = (T,NT,S,RR) que define o formato PBM, na qual:

T é o conjunto de símbolos terminais (em negrito), assim como os operadores relacionais;

NT é o conjunto de símbolos não terminais representados por palavras entre colchetes angulares;

S é o símbolo inicial <regra>;

RR é o conjunto de regras de reescrita da gramática na forma *LHS* ⇒ *RHS*, onde *LHS* é não terminal e *RHS* é uma seqüência de zero, um ou mais símbolos terminais ou não terminais.

S =	<regra>	⇒	<número-regra> **IF** <complexo> **THEN** <classe>
	<número-regra>	⇒	R0001 | R0002 | ...
	<complexo>	⇒	<fator> |
			<fator> **OR** <complexo>
	<fator>	⇒	<termo> |
			<termo> **AND** <fator>
	<termo>	⇒	<atributo> <operador> <valor>
	<operador>	⇒	< | <= | > | >= | = | <>
	<classe>	⇒	**CLASS** = <valor>
	<atributo>	⇒	$X_1 \mid X_2 \mid \ldots \mid X_m$
	<valor>	⇒	$x_{11} \mid x_{12} \mid \ldots \mid x_{nm} \mid y_1 \mid y_2 \mid \ldots \mid y_n$

Tabela 8.2: Gramática BNF que define o formato padrão de regras PBM

Com isso, é possível converter um classificador simbólico para o formato PBM. O formato padrão de regras é usado pelo componente classifier do sistema XRULER logo após a indução do classificador e antes de importar as regras para uma tabela na base de dados em MySQL. Informações adicionais providas por cada indutor, tais como número de exemplos cobertos corretamente e número de exemplos cobertos incorretamente, são descartadas. Isto deve-se ao fato que cada indutor fornece um tipo diferente de informação associada a cada regra ou nó de uma árvore de decisão (Baranauskas & Monard 2000d). A conversão é efetuada por um programa escrito na linguagem PERL que reconhece classificadores gerados por ID3, C4.5, C4.5RULES, CN2, OC1, RIPPER, T2 e MC4, sendo que outros indutores podem ser incorporados facilmente ao conversor PBM existente.

Como exemplo, na Figura 8.3 na página oposta é mostrada uma parte do classificador ID3 obtido utilizando o conjunto de exemplos breast-cancer. Esse mesmo classificador, no formato padrão, é mostrado na Figura 8.4 na página 140, sendo mais fácil de ser interpretado.

```
Rooted Decision Graph Categorizer ID3
Root: Uniformity of Cell Size

[0](Uniformity of Cell Size, level 0) : [0]--(?)-->[1][0]--(<= 2.5)-->[2][0]--(> 2.5)-->[21]
[1](2, level 1) :
[2](Bare Nuclei, level 1) : [2]--(?)-->[3][2]--(<= 3.5)-->[4][2]--(> 3.5)-->[11]
[3](2, level 2) :
[4](Clump Thickness, level 2) : [4]--(?)-->[5][4]--(<= 7.5)-->[6][4]--(> 7.5)-->[7]
[5](2, level 3) :
[6](2, level 3) :
[7](Sample code number, level 3) : [7]--(?)-->[8][7]--(<= 869119.5)-->[9][7]--(> 869119.5)-->[10]
[8](4, level 4) :
[9](2, level 4) :
[10](4, level 4) :
[11](Clump Thickness, level 2) : [11]--(?)-->[12][11]--(<= 3.5)-->[13][11]--(> 3.5)-->[14]
[12](2, level 3) :
[13](2, level 3) :
[14](Bland Chromatin, level 3) : [14]--(?)-->[15][14]--(<= 2.5)-->[16][14]--(> 2.5)-->[20]
[15](4, level 4) :
[16](Sample code number, level 4) : [16]--(?)-->[17][16]--(<= 1190989.5)-->[18][16]--(> 1190989.5)-->[19]
[17](4, level 5) :
[18](2, level 5) :
[19](4, level 5) :
[20](4, level 4) :
[21](Uniformity of Cell Size, level 1) : [21]--(?)-->[22][21]--(<= 4.5)-->[23][21]--(> 4.5)-->[66]
[22](4, level 2) :
[23](Bare Nuclei, level 2) : [23]--(?)-->[24][23]--(<= 2.5)-->[28][23]--(> 2.5)-->[38]
[24](Sample code number, level 3) : [24]--(?)-->[25][24]--(<= 559323.5)-->[26][24]--(> 559323.5)-->[27]
[25](4, level 4) :
[26](2, level 4) :
[27](4, level 4) :
[28](Normal Nucleoli, level 3) : [28]--(?)-->[29][28]--(<= 2.5)-->[30][28]--(> 2.5)-->[31]
[29](2, level 4) :
[30](2, level 4) :
[31](Sample code number, level 4) : [31]--(?)-->[32][31]--(<= 1144396.5)-->[33][31]--(> 1144396.5)-->[34]
[32](2, level 5) :
[33](4, level 5) :
[34](Sample code number, level 5) : [34]--(?)-->[35][34]--(<= 1343270)-->[36][34]--(> 1343270)-->[37]
[35](2, level 6) :
[36](2, level 6) :
[37](4, level 6) :
[38](Clump Thickness, level 3) : [38]--(?)-->[39][38]--(<= 6.5)-->[40][38]--(> 6.5)-->[59]
[39](4, level 4) :
[40](Bland Chromatin, level 4) : [40]--(?)-->[41][40]--(<= 3.5)-->[42][40]--(> 3.5)-->[52]
[41](4, level 5) :
[42](Uniformity of Cell Size, level 5) : [42]--(?)-->[43][42]--(<= 3.5)-->[44][42]--(> 3.5)-->[51]
[43](2, level 6) :
[44](Uniformity of Cell Shape, level 6) : [44]--(?)-->[45][44]--(<= 2.5)-->[46][44]--(> 2.5)-->[47]
[45](4, level 7) :
[46](2, level 7) :
[47](Clump Thickness, level 7) : [47]--(?)-->[48][47]--(<= 5.5)-->[49][47]--(> 5.5)-->[50]
[48](4, level 8) :
[49](4, level 8) :
[50](2, level 8) :
```

Figura 8.3: Parte do classificador induzido por ID3 utilizando o conjunto de exemplos breast-cancer

```
Standard Rules Conversor v1.1.3 Copyright (c)Ronaldo C. Prati
Inducer: id3                    Input File: breast_cancer.id3
Date: Mon May  7 00:04:43 2001

R0001   IF Uniformity of Cell Size = ?
        THEN CLASS = 2

R0002   IF Uniformity of Cell Size <= 2.5
                AND Bare Nuclei = ?
        THEN CLASS = 2

R0003   IF Uniformity of Cell Size <= 2.5
                AND Bare Nuclei <= 3.5
                AND Clump Thickness = ?
        THEN CLASS = 2

R0004   IF Uniformity of Cell Size <= 2.5
                AND Bare Nuclei <= 3.5
                AND Clump Thickness <= 7.5
        THEN CLASS = 2

R0005   IF Uniformity of Cell Size <= 2.5
                AND Bare Nuclei <= 3.5
                AND Clump Thickness > 7.5
                AND Sample code number = ?
        THEN CLASS = 4

R0006   IF Uniformity of Cell Size <= 2.5
                AND Bare Nuclei <= 3.5
                AND Clump Thickness > 7.5
                AND Sample code number <= 869119.5
        THEN CLASS = 2

R0007   IF Uniformity of Cell Size <= 2.5
                AND Bare Nuclei <= 3.5
                AND Clump Thickness > 7.5
                AND Sample code number > 869119.5
        THEN CLASS = 4

R0008   IF Uniformity of Cell Size <= 2.5
                AND Bare Nuclei > 3.5
                AND Clump Thickness = ?
        THEN CLASS = 2

R0009   IF Uniformity of Cell Size <= 2.5
                AND Bare Nuclei > 3.5
                AND Clump Thickness <= 3.5
        THEN CLASS = 2
```

Figura 8.4: Parte do classificador induzido por ID3 utilizando o conjunto de exemplos breast-cancer no formato padrão de regras

8.3.3 Algoritmo Básico de Cobertura

Uma vez induzidos os classificadores, através do componente classifier, torna-se possível obter um conjunto inicial das regras mais adequadas, conforme algum critério definido pelo usuário, através da aplicação dos componentes contingency e filter. Entretanto, mesmo utilizando filtros, a quantidade de regras obtidas pode ser grande. Assim, para obter um conjunto menor de regras tal que possam ser usadas como um classificador simbólico final, faz-se necessária a aplicação de um algoritmo de cobertura.

Considerando o componente cover, o sistema XRULER procura por um conjunto mínimo de regras. As regras obtidas pelo componente cover são não ordenadas (vide Seção 3.4), já que são mais fáceis de serem compreendidas por seres humanos.

No Algoritmo 8.1 é mostrado o procedimento básico de cobertura implementado para seleção de regras não ordenadas no sistema XRULER. Dado o conjunto de regras *RuleSet* e um conjunto de exemplos de cobertura *InstanceSet*, o algoritmo procura pela melhor regra *best_rule*, segundo algum critério. Uma vez encontrada a melhor regra, ela é então incluída no conjunto de cobertura *CoverSet* e removida de *RuleSet*. Em seguida, os exemplos corretamente cobertos pela melhor regra são removidos do conjunto de cobertura *InstanceSet*. O motivo de se manter os exemplos cobertos incorretamente (removendo apenas os exemplos corretamente cobertos pela melhor regra), possibilita que o algoritmo encontre, nas próximas iterações, uma regra que os cubra corretamente. A função *covered(I,best_rule)* é verdadeira se o exemplo I é coberto corretamente pela regra *best_rule* ou falsa, caso contrário. O processo se repete, enquanto houver exemplos não cobertos, até que o conjunto de regras se esgote ou se nenhuma melhor regra for encontrada.

Algoritmo 8.1 Algoritmo básico de cobertura para regras não ordenadas

Require: InstanceSet: conjunto de exemplos a serem cobertos
RuleSet: conjunto de regras
Ensure: ConverSet: conjunto mínimo de regras de RuleSet que cobre InstanceSet

```
1: procedure basic_cover(InstanceSet,RuleSet)
2: CoverSet := ∅
3: repeat
4:    best_rule := selecionar melhor regra do conjunto RuleSet sobre InstanceSet
5:    CoverSet := CoverSet + {best_rule}
6:    RuleSet := RuleSet − {best_rule}
7:    InstanceSet := InstanceSet − {I : I in InstanceSet and covered(I,best_rule)}
8: until (InstanceSet = ∅) or (RuleSet = ∅) or not found(best_rule)
9: return CoverSet
```

8.3.4 Critérios para Seleção da Melhor Regra

Analisando o Algoritmo 8.1 na página precedente, é possível observar que o fator determinante no resultado final depende do critério de seleção da melhor regra. Na realidade, existem inúmeros critérios que podem ser escolhidos, cada um proporcionando um conjunto final de regras bem diversificado (Horst & Monard 2000; Gomes 2001).

Um dos possíveis critérios de seleção da melhor regra consiste em escolher a regra com o melhor grau de adequação (*rule fitness*) para o problema em questão. Denotando-se o grau de adequação de uma regra $L \rightarrow R$ por $\mathsf{rf}(L \rightarrow R)$, em geral, o critério de seleção escolhe a melhor regra como sendo aquela que possui o maior valor de rf. A seguir são descritos três possíveis critérios que permitem estimar o grau de adequação de uma regra.

Confiabilidade Positiva Uma possível definição para o grau de adequação rf utiliza a *precisão*. Dada uma regra $L \rightarrow R$, Todorovski, Flach & Lavrač (2000) definem a precisão $\mathsf{acc}(L \rightarrow R)$ como sendo a probabilidade condicional da conclusão R, dada que a condição L é satisfeita, conforme (8.1).

$$\mathsf{rf}(L \rightarrow R) \stackrel{def}{=} \mathsf{acc}(L \rightarrow R) = p(R|L) \tag{8.1}$$

Uma forma simples de estimar $p(R|L)$ consiste na utilização da confiabilidade positiva $\mathsf{prel}(L \rightarrow R)$, definida por (2.18) na página 37, que é a razão entre o número de exemplos corretamente cobertos pela regra e o número total de exemplos cobertos pela regra, ou seja, lr/l, segundo a notação apresentada na Tabela 2.7 na página 36.

Equivalentemente, $\mathsf{prel}(L \rightarrow R) = lr/l$ pode ser expressa como $lr/(lr + l\bar{r})$, onde lr é o número de exemplos cobertos corretamente pela regra e $l\bar{r}$ é número de exemplos cobertos incorretamente pela regra. É possível notar que $0 \leq \mathsf{prel}(L \rightarrow R) \leq 1$ e, segundo esse critério, quanto maior a confiabilidade positiva, melhor a regra.

Entretanto, a confiabilidade positiva possui uma propriedade indesejada. Por exemplo, considere duas regras $\mathcal{R}_1 \equiv L_1 \rightarrow R_1$ e $\mathcal{R}_2 \equiv L_2 \rightarrow R_2$, com $l_1 r_1 = 100$, $l_1 \bar{r}_1 = 1$ e $l_2 r_2 = 5$, $l_2 \bar{r}_2 = 0$. Nessa situação, $\mathsf{prel}(\mathcal{R}_1) = l_1 r_1 / l_1 = 0,99$ e $\mathsf{prel}(\mathcal{R}_2) = l_2 r_2 / l_2 = 1,00$, indicando que $\mathcal{R}_2$ é uma regra melhor do que $\mathcal{R}_1$, o que não procede neste caso.

Precisão de Laplace Uma solução para o problema apresentado pela confiabilidade positiva ao estimar $p(R|L)$ consiste em substituí-la pela precisão de Laplace (Clark & Boswell 1991), definida por (8.2), onde K é o número de classes do conjunto de exemplos.

$$\mathsf{rf}(L \rightarrow R) \stackrel{def}{=} \mathsf{lacc}(L \rightarrow R) = \frac{lr+1}{l+K} = \frac{lr+1}{lr+l\bar{r}+K} \tag{8.2}$$

Considerando esse critério, quanto maior o valor da precisão de Laplace, melhor é a regra. Assim, ainda considerando o exemplo anterior em um problema com $K = 2$ classes, tem-se que $\mathsf{lacc}(\mathcal{R}_1) = 0,98$ e $\mathsf{lacc}(\mathcal{R}_2) = 0,85$, indicando que a regra $\mathcal{R}_1$ é melhor do que a regra $\mathcal{R}_2$. Assim como a confiabildidade positiva, a precisão de Laplace assume valores no intervalo $0 \leq \mathsf{lacc}(L \rightarrow R) \leq 1$.

Novidade A novidade $\mathsf{nov}(L \rightarrow R)$, definida por (2.25) na página 37, compara o resultado observado lr contra o valor esperado sob a consideração de independência $l \cdot r/n$, onde n é o número exemplos. Assim, é possível estimar o grau de adequação de uma regra usando a novidade, conforme (8.3).

$$\mathsf{rf}(L \rightarrow R) \stackrel{def}{=} \mathsf{nov}(L \rightarrow R) = p(L)(p(R|L) - p(R)) \tag{8.3}$$

Como mencionado, a novidade assume valores no intervalo $-0,25 \leq \mathsf{nov}(L \rightarrow R) \leq 0,25$ e, considerando esse critério, quanto maior um valor positivo (próximo de 0,25), melhor é a regra. O termo $p(R|L)$ em (8.3) pode ser estimado usando a precisão de Laplace, definida anteriormente.

A utilização de novidade como uma estimativa do grau de adequação de regras permite reduzir a quantidade do conjunto final de regras. Nos experimentos realizados por Todorovski, Flach & Lavrač (2000), em média, o emprego da novidade reduziu em nove vezes o número de regras não ordenadas induzidas pelo algoritmo CN2, com um aumento do erro em torno de 5%. Essa redução do número de regras com pequena perda de precisão constitui um avanço no sentido de uma maior compreensibilidade do conjunto final de regras por seres humanos.

8.3.5 Algoritmo de Cobertura

No Algoritmo 8.2 na próxima página é descrito o procedimento de cobertura do sistema XRULER, que consiste no detalhamento do Algoritmo 8.1. Para estimar o grau de adequação rf de uma regra, inicialmente foram implementadas a precisão de Laplace e a novidade, sendo

Algoritmo 8.2 Algoritmo de cobertura utilizado no sistema XRULER

Require: InstanceSet: conjunto de exemplos a serem cobertos
RuleSet: conjunto de regras
MIN_SUP: suporte absoluto mínimo esperado para uma regra (default 2)
MAX_RULES: número máximo de regras resultantes (default 1000)
Ensure: ConverSet: conjunto mínimo de regras de RuleSet que cobre InstanceSet

```
1: procedure cover(InstanceSet,RuleSet)
2: CoverSet := ∅        // conjunto de regras cobertas pelo algoritmo
3: K := #classes(InstanceSet)        // número de classes no conjunto InstanceSet
4: nrules := 0
5: repeat
6:    max_rule_fitness = 0.0
7:    max_sup = 0.0
8:    for all rule (no formato L → R) in RuleSet do
9:       rule_fitness = rf(L → R)        // calcular grau de adequação da regra
10:      sup = lr        // suporte absoluto da regra
11:      if rule_fitness > max_rule_fitness then
12:         max_rule_fitness := rule_fitness        // encontrada regra mais adequada
13:         best_rule := rule        // marcar melhor regra
14:         max_sup := sup
15:      end if
16:   end for
17:   if max_sup ≥ MIN_SUP then
18:      nrules := nrules + 1
19:      RuleSet := RuleSet − {best_rule}
20:      CoverSet := CoverSet + {best_rule}
21:      InstanceSet := InstanceSet − {I : I in InstanceSet and covered(I,best_rule)}
22:   end if
23: until (InstanceSet = ∅) or (RuleSet = ∅) or (nrules > MAX_RULES) or (max_sup <
    MIN_SUP)
24: return CoverSet
```

possível adicionar novas estimativas ao sistema XRULER. Devido aos problemas descritos na Seção 8.3.4, a confiabilidade positiva não foi utilizada. Todas as medidas de regras utilizadas pelo algoritmo são calculadas sobre o conjunto *InstanceSet*.

Além disso, o critério de parada foi alterado, em relação ao algoritmo básico de cobertura, em razão de dois parâmetros adicionais: *MAX_RULES*, que indica o número máximo de regras que o usuário pretende obter no processo de cobertura e *MIN_SUP*, que é o suporte absoluto mínimo esperado para a melhor regra em cada iteração, ou seja, a melhor regra deve cobrir corretamente, no mínimo, MIN_SUP exemplos para ser selecionada. A alteração foi efetuada para impedir que regras cobrindo poucos exemplos fossem selecionadas, bem como permitir ao usuário definir o número máximo desejado de regras.

8.3.6 Classificação de Novos Exemplos

Uma vez realizada a cobertura de regras pelo componente cover do sistema XRULER, é possível avaliar o conjunto final de regras obtido com relação à precisão. Considerando o componente evaluator, o sistema XRULER utiliza um critério similar ao indutor CN2 ao rotular novos exemplos. O algoritmo de avaliação do sistema XRULER é representado no Algoritmo 8.3.

Algoritmo 8.3 Algoritmo de classificação de novos exemplos utilizado no sistema XRULER

Require: NewInstance: novo exemplo a ser rotulado
InstanceSet: conjunto de exemplos usado na cobertura
CoverSet: conjunto de regras obtidas pelo Algoritmo 8.2
Ensure: the_class: classe associada ao novo exemplo NewInstance
1: **procedure** classify(NewInstance,InstanceSet,CoverSet)
2: K := número de classes no conjunto InstanceSet
3: majority_class := classe majoritária de exemplos não cobertos por nenhuma regra em CoverSet
4: **for all** class **in** InstanceSet **do**
5: Total[class].lr := 0
6: Total[class].lnr := 0
7: **end for**
8: FiredRules := $\emptyset$
9: **for all** rule (no formato $L \rightarrow R$) **in** CoverSet **do**
10: **if** covered(NewInstance,L) **then**
11: FiredRules := FiredRules + {rule}
12: Total[R].lr := Total[R].lr + lr
13: Total[R].lnr := Total[R].lnr + $l\bar{r}$
14: **end if**
15: **end for**
16: **if** FiredRules = $\emptyset$ **then**
17: the_class := majority_class
18: **else**
19: max_lacc := 0.0
20: **for all** class **in** Total **do**
21: lacc := (Total[class].lr + 1)/(Total[class].lr + Total[class].lnr + K)
22: **if** lacc > max_lacc **then**
23: max_lacc := lacc
24: the_class := class
25: **end if**
26: **end for**
27: **end if**
28: **return** the_class

Inicialmente, para cada regra $L \rightarrow R$ são calculados os valores lr e $l\bar{r}$ utilizando o mesmo conjunto de exemplos utilizado pelo Algoritmo 8.2 de cobertura. Dado um novo exemplo, todas as regras que disparam são coletadas e seus valores associados são totalizados para as respectivas

classes. Após somados os valores das regras por classe, a precisão de Laplace é então calculada e a classe com maior precisão é selecionada para rotular o novo exemplo.

Por exemplo, suponha que, dado um novo exemplo, três regras $\mathcal{R}_1$, $\mathcal{R}_2$ e $\mathcal{R}_3$ disparam para esse exemplo, sendo que a classe prevista por $\mathcal{R}_1$ e $\mathcal{R}_3$ é diferente de $\mathcal{R}_2$. Assumindo $l_1 r_1 = 30$, $l_1 \bar{r}_1 = 5$, $l_2 r_2 = 41$, $l_2 \bar{r}_2 = 2$ e $l_3 r_3 = 10$, $l_3 \bar{r}_3 = 0$, e sendo a classe de $\mathcal{R}_1$ e $\mathcal{R}_3$ é a mesma, seus valores são somados, totalizando $l_{1,3} r_{1,3} = 40$, $l_{1,3} \bar{r}_{1,3} = 5$. O valor da precisão de Laplace para as regras $\mathcal{R}_1$ e $\mathcal{R}_3$ é $\mathsf{lacc}(\mathcal{R}_1, \mathcal{R}_3) = (40 + 1)/(40 + 5 + 2) = 0,87$ e para a regra $\mathcal{R}_2$ é $\mathsf{lacc}(\mathcal{R}_2) = (41 + 1)/(41 + 2 + 2) = 0,93$. Como $\mathsf{lacc}(\mathcal{R}_2) > \mathsf{lacc}(\mathcal{R}_1, \mathcal{R}_3)$, a classe prevista pela regra $\mathcal{R}_2$ é selecionada para rotular o novo exemplo.

8.3.7 XRULER e o Projeto DISCOVER

No decorrer do desenvolvimento desta tese e através do contato com os demais integrantes do grupo de pesquisa do LABIC, foi possível perceber que muitas pesquisas que estavam sendo realizadas tinham, em geral, alguns componentes em comum. Em algumas situações, havia uma duplicação de esforços quando diferentes pesquisadores desenvolviam softwares contendo partes em comum.

Aliado a esse fato, a biblioteca $\mathcal{MLC}$++, mesmo estando disponível, tem seu uso mais voltado à análise da precisão ou taxa de aprendizado de um algoritmo, ou seja, a biblioteca $\mathcal{MLC}$++ focaliza o interesse nos classificadores como caixas-preta, não fornecendo uma visão única dos classificadores simbólicos que podem ser extraídos pela biblioteca. Outro problema associado é que alguns dos utilitários da $\mathcal{MLC}$++ somente funcionam com os indutores que foram reescritos na própria biblioteca (por exemplo, ID3) e não com os indutores externos com os quais faz interface, tais como C4.5 e CN2. Uma opinião pessoal sobre $\mathcal{MLC}$++ é que ela é de difícil compilação e testes efetuados, com o intuito de adicionar novos utilitários, não foram bem sucedidos.

Outra ferramenta disponível no LABIC, o MineSet™, embora utilize $\mathcal{MLC}$++ como base de seus utilitários, não especifica detalhes de seu funcionamento, sendo um software voltado para usuários finais e não para pesquisadores. Novamente, o interesse da ferramenta é voltado para classificadores como caixas-preta.

Esses e outros fatores motivaram a idéia inicial do projeto DISCOVER , proposto conjuntamente por Baranauskas & Batista (2000) e representado esquematicamente na Figura 8.5 na próxima página. Esse projeto consiste em uma estratégia de trabalho conjunto, envolvendo um conjunto de ferramentas que se integram e que venham a suprir as necessidades dos trabalhos realizados e em andamento relacionados com Aprendizado de Máquina e *Data Mining* no LA-

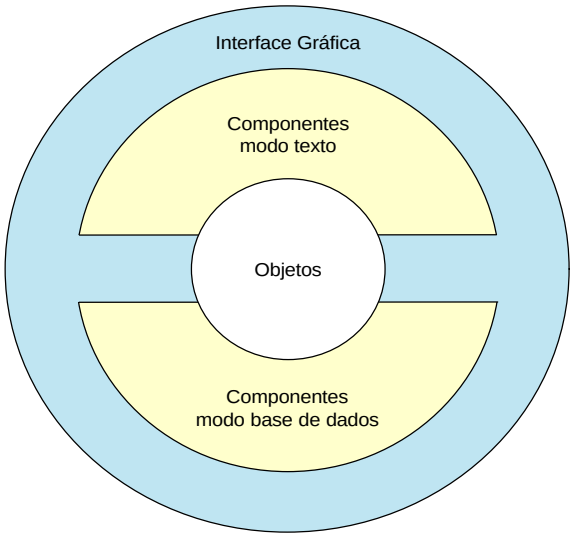


Figura 8.5: O projeto DISCOVER

BIC (Gomes 2001; Martins 2001; Pila 2001; Pugliesi 2001; Batista 2000; Caulkins 2000; Imamura 2000; Lee 2000; Milaré 2000). O projeto contempla todas as fases do processo de Extração de Conhecimento, incluindo funcionalidades não encontradas nem na biblioteca $\mathcal{MLC}$++ e nem na ferramenta MineSet™, tais como pré-processamento de dados e de texto para aprendizado supervisionado e não-supervisionado; amostragem e avaliação do erro; avaliação de regras; mesclagem de regras; cobertura de regras, entre outras funcionalidades voltadas para AM, DM e Mineração de Texto (*Text Mining* — TM) (Dörre, Gerstl & Seiffert 1999).

Inicialmente, foram propostas duas abordagens no projeto DISCOVER: ferramentas de manipulação de objetos em modo texto (ASCII) e ferramentas de manipulação de objetos em base de dados. No DISCOVER, um objeto pode ser um conjunto de exemplos, um classificador, uma regra, uma amostra, um texto, etc.

Tanto no modo texto como no modo base de dados, os utilitários convertem objetos ou executam ações sobre objetos. Por exemplo, os conjuntos de exemplos, no formato padrão do DISCOVER, são convertidos para o formato de entrada dos diversos indutores através de um utilitário de conversão. Como pode ser notado, o projeto adota o conceito de 'formato padrão' para todos os seus objetos. O formato padrão de um conjunto de exemplos, definido por Batista (2001), é similar ao adotado na biblioteca $\mathcal{MLC}$++, mas contendo extensões para a aprendizado não supervisionado e TM, além de permitir a inclusão de atributos derivados, ou seja, atributos que são construídos em função de outros atributos (aprendizado construtivo).

Os classificadores simbólicos são convertidos para o formato padrão de regras PBM, descrito na Seção 8.3.2. Uma vez convertidos os classificadores para o formato padrão, eles podem

ser avaliados utilizando o conjunto de treinamento, ou outro conjunto, para obter medidas padrões. Essas medidas incluem os quatro valores da matriz de contingência, calculada para cada regra, nos exemplos que contenham valores desconhecidos ou não (Prati, Baranauskas & Monard 2001a).

No modo base de dados, é possível utilizar comandos SQL para extrair informações sobre os exemplos, classificadores, regras, entre outras. O servidor de base de dados adotado é o MySQL e a maioria dos utilitários estão sendo escritos como *scripts* na linguagem PERL. Embora PERL seja a linguagem padrão, nada impede o uso de outras linguagens de programação, desde que as entradas e saídas sejam compatíveis com os demais utilitários do projeto.

Embora a idéia do projeto DISCOVER seja aparentemente simples, sua implementação não é trivial. Por exemplo, a coexistência do modo texto e modo base de dados no projeto é devido ao fato que para determinadas atividades, a utilização de um sistema de base de dados facilita o armazenamento e manipulação de estruturas induzidas. O sistema XRULER trabalha no modo base de dados, armazenando amostras, classificadores e regras em tabelas. Portanto, diferentemente de outros projetos, DISCOVER tem como finalidade seu uso e extensão por pesquisadores de AM. O ponto crucial é que o projeto seja flexível o suficiente para permitir que novas pesquisas sejam englobadas e, simultaneamente, imponha determinados padrões que permitam a integração de todos os seus componentes.

A seguir é descrita a metodologia experimental adotada para avaliar o desempenho do sistema XRULER em vários conjuntos de exemplos.

8.4 Metodologia Experimental

A whole is that which has beginning, middle and end.

—*Aristotle*

De forma a avaliar o sistema XRULER, alguns experimentos foram conduzidos utilizando os conjuntos de exemplos bupa, pima, breast-cancer, hungaria, crx, hepatitis, anneal, sonar, genetics e dna, descritos na Seção 3.3 na página 51 e os indutores C4.5 e CN2. Como nos outros experimentos, os conjuntos de exemplos não foram pré-processados de nenhuma maneira. Os experimentos foram realizados da seguinte forma:

1. Cada conjunto original de exemplos foi aleatoriamente particionado em dois subconjuntos disjuntos: o conjunto de aprendizado, contendo 70% dos exemplos, e o conjunto de avaliação final, contendo os 30% dos exemplos restantes. Deve ser ressaltado que o conjunto

de avaliação não foi utilizado pelo sistema XRULER na fase de aprendizado.

2. Considerando o conjunto de aprendizado (contendo apenas 70% dos exemplos originais), 10 amostras *bootstrap* foram extraídas, resultando em 20 amostras (10 amostras de treinamento e 10 amostras de teste).

3. Para cada amostra *bootstrap* de treinamento, os indutores C4.5 e CN2 foram aplicados. Ambos os algoritmos foram executados com seus parâmetros *default*, ou seja, sem ajustes. Ao final desta etapa, 20 classificadores (10 classificadores C4.5 e 10 classificadores CN2) foram induzidos.

4. Para cada classificador induzido, a matriz de contingência e as demais medidas dela derivada foram calculadas, utilizando a amostra de teste associada à amostra *bootstrap* de treinamento. Embora XRULER permita a aplicação de filtros, nenhum filtro foi aplicado nesta etapa do experimento.

5. Em seguida, foi aplicado o algoritmo de cobertura de forma a extrair um conjunto mínimo de regras que cobrissem os exemplos contidos no conjunto de aprendizado. O algoritmo de cobertura foi aplicado duas vezes: uma vez utilizando a precisão de Laplace e a segunda utilizando a novidade.

6. Finalmente, o conjunto mínimo de regras foi avaliado quanto a precisão utilizando os exemplos contidos no conjunto de avaliação final (contendo 30% dos exemplos nunca vistos por XRULER durante o aprendizado). Como uma base de comparação, os indutores C4.5 e CN2, com seus parâmetros *default*, foram aplicados ao conjunto de aprendizado e sua precisão foi medida no conjunto de avaliação.

7. Uma vez que os conjuntos de dados possuem tamanhos moderados, todas as etapas anteriores foram repetidas 10 vezes, de forma a permitir melhores estimativas. Entretanto, para grandes conjuntos de dados tipicamente disponíveis em DM, isso não seria necessário.

8.5 Resultados

> The most beautiful thing we can experience is the mysterious. It is the source of all true art and Science.
>
> —*Albert Einstein*

Nesta seção são apresentados os resultados dos experimentos realizados. Na Tabela 8.3 na próxima página é mostrada a precisão (média e desvio padrão) obtida por C4.5, CN2 e XRULER

Dataset	C4.5	CN2	XRULER (lacc)	XRULER (nov)
bupa	64,07±1,05	65,43±1,02	66,12±1,78	64,56±2,00
pima	74,44±0,83	72,74±0,88	73,17±0,76	73,74±0,49
breast-cancer	93,82±0,30	95,07±0,32	94,31±0,37	93,97±0,57
hungaria	77,62±1,55	78,18±1,38	77,05±1,52	78,98±1,35
crx	85,91±0,72	82,71±0,45	84,40±0,69	85,60±0,66
hepatitis	76,74±1,58	79,35±0,81	79,78±1,38	77,61±1,65
anneal	90,65±0,70	91,82±0,99	96,77±0,51	93,83±1,51
sonar	72,42±1,45	71,29±1,66	74,19±2,07	71,45±1,25
genetics	93,60±0,27	79,85±1,59	93,00±0,15	89,85±1,67
dna	92,49±0,34	87,94±0,27	92,45±0,31	93,16±0,21
Média	82,18	80,44	83,12	82,28

Tabela 8.3: Precisão de C4.5, CN2 e XRULER utilizando *bootstrap*

utilizando Laplace (lacc) e novidade (nov). Em média, a precisão de C4.5 foi 82,18%, de CN2 foi 80,44%, enquanto a precisão de XRULER (lacc) foi 83,12% e XRULER (nov) foi 82,28%. Isso significa que XRULER (lacc) proporcionou uma redução relativa na taxa de erro de 5,32% para C4.5 e de 13,73% para CN2, enquanto XRULER (nov) proporcionou uma redução relativa na taxa de erro de 0,56% para C4.5 e de 9,39% para CN2.

Na Figura 8.6 na página oposta é mostrada a diferença absoluta em desvios padrões da precisão. Qualquer barra com altura maior que dois indica que o resultado é significativo, com nível de confiança de 95%. Quando a barra encontra-se acima de zero significa que XRULER supera o segundo indutor (C4.5 ou CN2); se a barra encontra-se abaixo então o segundo indutor supera XRULER. Quando a altura da barra estiver acima (abaixo) de dois significa que XRULER (segundo algoritmo) supera significativamente o segundo algoritmo (XRULER).

Na Tabela 8.4 na página 152 é mostrado o número médio de regras. Como pode ser notado, em média, C4.5 induziu 48,60 regras e CN2 induziu 38,34 regras, enquanto que XRULER (lacc) obteve 36,78 regras, uma redução relativa de 24,32% e de 4,07%, respectivamente. Uma redução maior foi obtida por XRULER (nov), obtendo, em média, 14,26 regras. Isso significa 70,66% menos regras do que C4.5 e 62,81% menos regras do que CN2.

Na Figura 8.7 na página 152 é mostrada a diferença absoluta (em desvios padrões) do número de regras. Valores negativos (positivos) indicam que XRULER reduziu (aumentou) o número de regras em comparação com os indutores C4.5 e CN2. Como pode ser observado, XRULER aparenta aumentar o número de regras em conjuntos com poucos exemplos e atributos, mas diminui a quantidade de regras nos conjuntos de exemplos de maiores dimensões, exceto para genetics utilizando CN2.

Na Tabela 8.5 na página 153 é mostrado o número médio de condições por regra. Em média,

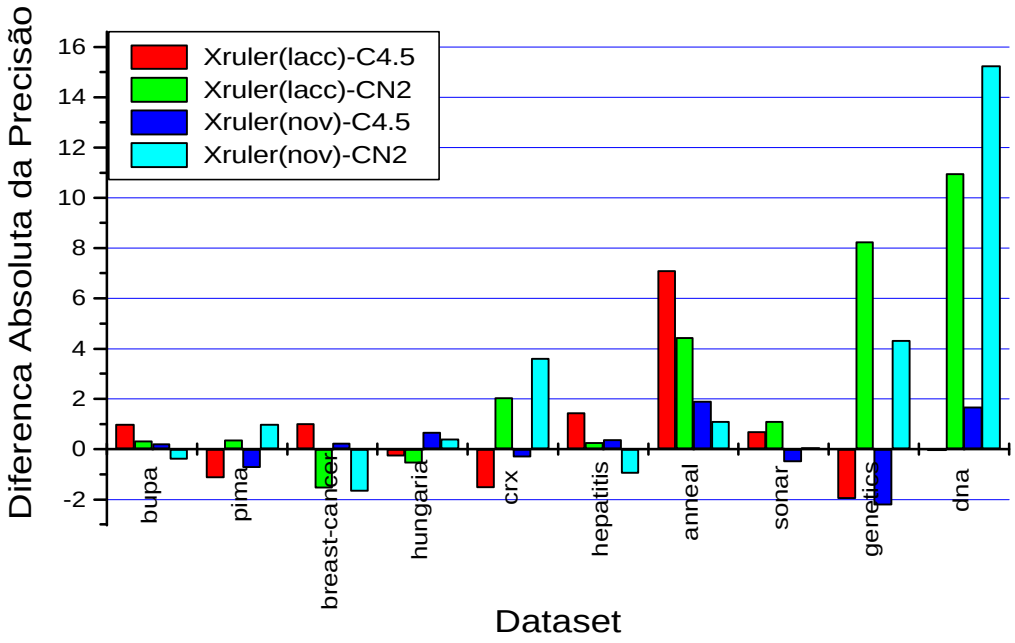


Figura 8.6: Diferença absoluta (em desvios padrões) da precisão

C4.5 induziu 4,69 e CN2 induziu 3,22 condições por regra, enquanto XRULER (lacc) obteve 3,98 e XRULER (nov) 4,28 condições por regra. Assim, XRULER (lacc) promoveu uma redução de 15,09% para C4.5 e um aumento de 23,61% para CN2 no número de condições por regra. Já XRULER (nov) promoveu uma redução de 8,60% para C4.5 e um aumento de 33,05% para CN2 no número de condições por regra.

Na Figura 8.8 na página 153 é mostrada a diferença absoluta (em desvios padrões) do número de condições. Valores negativos (positivos) indicam que XRULER reduziu (aumentou) o número de condições por regra em comparação com os indutores C4.5 e CN2. Nota-se que não apenas na média, mas em geral, XRULER provocou um aumento do número de condições em comparação com CN2 e uma redução mais significativa para C4.5 nos conjuntos de exemplos de maior dimensão.

Na Tabela 8.6 na página 154 é comparada a precisão (eixo horizontal) e o número de regras (eixo vertical) entre XRULER (indicado por X) e C4.5 ou CN2. No primeiro quadrante, indicado por $(\oplus, \oplus)$, estão marcados os conjuntos de exemplos nos quais houve um aumento tanto na precisão quanto no número de regras. No segundo quadrante, indicado por $(\ominus, \oplus)$, estão marcados os conjuntos de exemplos nos quais houve uma diminuição na precisão e um aumento no número de regras. No terceiro quadrante, indicado por $(\ominus, \ominus)$, estão marcados os conjuntos de exemplos

Dataset	C4.5	CN2	XRULER (lacc)	XRULER (nov)
bupa	18,40±4,38	27,60±2,01	28,00±0,94	11,60±3,03
pima	19,20±9,34	39,80±5,37	49,30±3,16	21,40±2,80
breast-cancer	11,20±4,10	14,80±1,32	12,60±2,50	6,90±1,52
hungaria	10,00±2,98	21,70±2,45	17,10±1,85	6,40±1,07
crx	16,20±7,63	31,70±2,67	33,90±3,18	12,00±3,09
hepatitis	6,50±2,37	15,10±1,91	7,60±1,71	3,70±0,95
anneal	45,80±11,10	45,90±6,30	13,80±1,03	8,90±1,29
sonar	12,60±1,58	22,50±1,96	11,20±1,03	4,70±1,25
genetics	276,10±20,62	65,20±9,64	118,80±4,44	32,40±9,61
dna	70,00±9,49	99,10±5,76	75,50±1,96	34,60±4,14
Média	48,60	38,34	36,78	14,26

Tabela 8.4: Número de regras de C4.5, CN2 e XRULER

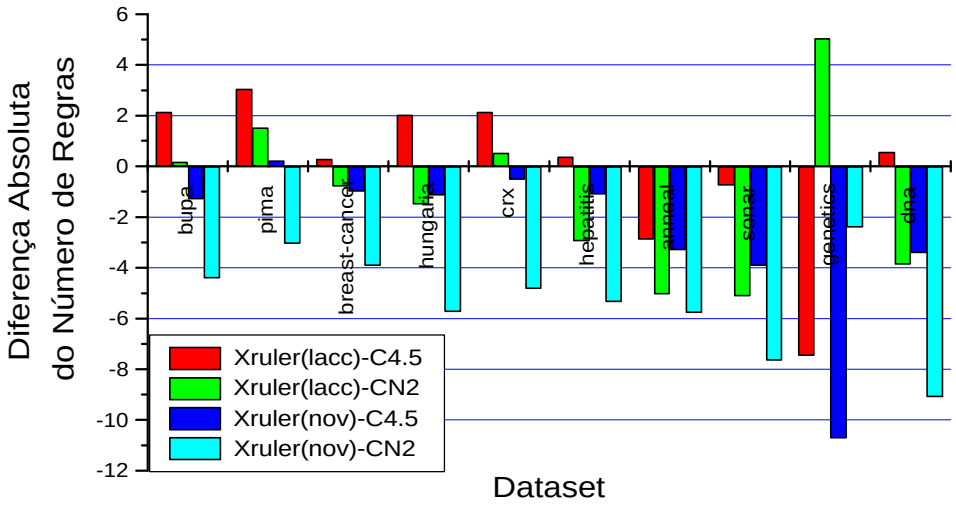


Figura 8.7: Diferença absoluta (em desvios padrões) do número de regras

nos quais houve uma diminuição tanto na precisão quanto no número de regras. No quarto quadrante, indicado por $(\oplus, \ominus)$, estão marcados os conjuntos de exemplos nos quais houve um aumento na precisão e uma diminuição no número de regras. Quando a diferença, tanto para a precisão quanto para o número de regras, é significativa, ou seja com nível de confiança de 95%, isto é indicado por '$\bullet$', caso contrário por '$\circ$'. De forma análoga, são efetuadas comparações entre a precisão e o número de condições por regra bem como entre o número de regras e o número de condições por regra, mostradas nas Tabelas 8.7 e 8.8, respectivamente.

Considerando a Tabela 8.6 na página 154, é possível notar uma maior concentração de resultados no terceiro e quarto quadrantes. No terceiro quadrante, há uma concentração de resultados nos conjuntos de exemplos menores, exceto no conjunto de exemplos genetics, na coluna que

Dataset	C4.5	CN2	XRULER (lacc)	XRULER (nov)
bupa	5,44±0,27	3,09±0,04	4,09±0,06	4,29±0,17
pima	5,50±0,33	3,24±0,03	5,05±0,13	5,24±0,12
breast-cancer	3,92±0,27	2,50±0,04	3,25±0,11	2,98±0,14
hungaria	3,68±0,20	3,36±0,05	3,89±0,07	3,54±0,14
crx	3,85±0,18	3,29±0,03	3,87±0,10	4,05±0,16
hepatitis	3,01±0,22	3,18±0,09	3,12±0,07	3,43±0,07
anneal	4,41±0,10	3,75±0,05	4,36±0,10	4,07±0,13
sonar	4,31±0,18	1,97±0,02	3,01±0,11	4,32±0,30
genetics	4,54±0,04	3,06±0,06	3,56±0,03	4,29±0,05
dna	8,20±0,14	4,75±0,02	5,59±0,06	6,62±0,11
Média	4,69	3,22	3,98	4,28

Tabela 8.5: Número de condições por regra de C4.5, CN2 e XRULER

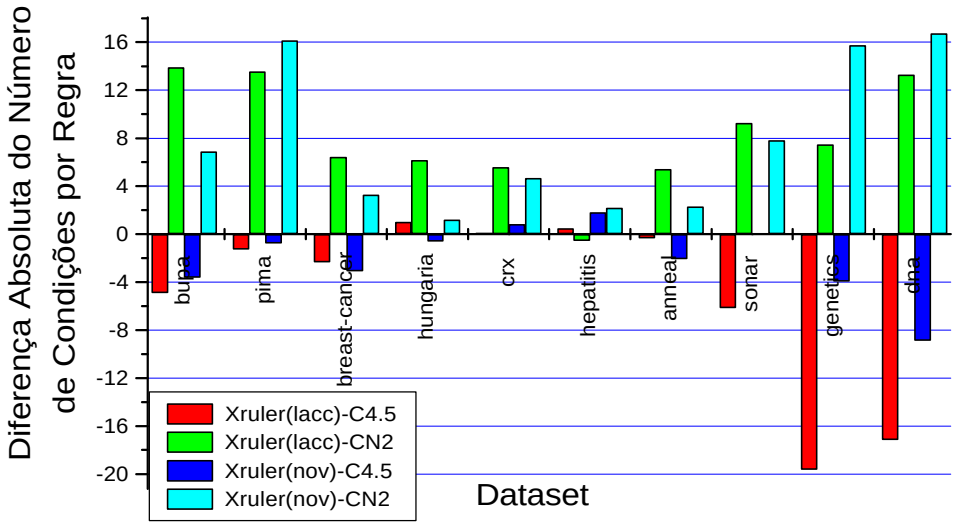


Figura 8.8: Diferença absoluta (em desvios padrões) do número de condições por regra

compara os resultados de XRULER (nov) com C4.5, na qual a redução, tanto da precisão quanto do número de regras, foi significativa. Já no quarto quadrante, é possível notar uma concentração de resultados para os conjuntos de exemplos maiores. Observando apenas os resultados nos quais a diferença é significativa, os resultados se concentram nos casos em que houve aumento da precisão e diminuição do número de regras. Esses resultados são encorajadores, pois mostram que XRULER induz menos regras que os classificadores correspondentes e que essas regras classificam exemplos nunca vistos com maior precisão que o classificador original.

Considerando a Tabela 8.7 na próxima página, é possível observar que os resultados significativos se concentram no primeiro quadrante nos conjuntos de exemplos maiores e apenas para o indutor CN2, ou seja, nos casos em que houve uma aumento tanto da precisão quanto do número

Dataset	X(lacc) − C4.5	X(lacc) − CN2	X(nov) − C4.5	X(nov) − CN2	X(lacc) − C4.5	X(lacc) − CN2	X(nov) − C4.5	X(nov) − CN2
	$(\ominus,\oplus)$				$(\oplus,\oplus)$			
bupa	○					○		
pima	○	○	○					
breast-cancer					○			
hungaria	○							
crx	○					○		
hepatitis					○			
anneal								
sonar								
genetics						●		
dna	○							
bupa			○	○				
pima				○				
breast-cancer		○	○	○				
hungaria		○	○					○
crx							○	●
hepatitis		○	○	●				
anneal					●	●	○	○
sonar				○	○	○	○	
genetics	○		●					●
dna						●	○	●
	$(\ominus,\ominus)$				$(\oplus,\ominus)$			

Tabela 8.6: Precisão *versus* número de regras

Dataset	X(lacc) − C4.5	X(lacc) − CN2	X(nov) − C4.5	X(nov) − CN2	X(lacc) − C4.5	X(lacc) − CN2	X(nov) − C4.5	X(nov) − CN2
	$(\ominus,\oplus)$				$(\oplus,\oplus)$			
bupa				○		○		
pima		○		○				
breast-cancer		○		○				
hungaria	○	○						○
crx	○					○	○	●
hepatitis			○	●	○			
anneal						●		○
sonar				○		○	○	
genetics						●		●
dna						●		●
bupa	○		○					
pima	○		○					
breast-cancer			○		○			
hungaria			○					
crx								
hepatitis		○						
anneal					○		○	
sonar					○			
genetics	○		●					
dna	○						○	
	$(\ominus,\ominus)$				$(\oplus,\ominus)$			

Tabela 8.7: Precisão *versus* número de condições por regra

Dataset	X(lacc) − C4.5	X(lacc) − CN2	X(nov) − C4.5	X(nov) − CN2	X(lacc) − C4.5	X(lacc) − CN2	X(nov) − C4.5	X(nov) − CN2
		$(\ominus,\oplus)$				$(\oplus,\oplus)$		
bupa				●		○		
pima				●		○		
breast-cancer		○		●				
hungaria		○		○	○			
crx			○	●	○	○		
letter								
hepatitis			○	●	○			
anneal		●		●				
sonar		●	○	●				
genetics				●		●		
dna		●		●				
bupa			○		●			
pima					○		○	
breast-cancer			○		○			
hungaria			○					
crx								
hepatitis		○						
anneal	○		●					
sonar	○							
genetics	●		●					
dna			●		○			
		$(\ominus,\ominus)$				$(\oplus,\ominus)$		

Tabela 8.8: Número de regras *versus* número de condições por regra

de condições por regra. Assim, pode-se concluir que para CN2, XRULER consegue incrementar a precisão, mas incrementa o número de condições por regra. Na realidade, comparando a coluna relativa ao indutor CN2 das Tabelas 8.6 e 8.7, as duas versões de XRULER, independentemente da precisão, incrementam o número de condições por regra, exceto para XRULER (lacc) no conjunto de exemplo hepatitis.

Considerando a Tabela 8.8, nota-se uma grande quantidade de resultados significativos no segundo quadrante, no qual houve diminuição do número de regras e aumento no número de condições, em relação ao indutor CN2. Observa-se, também, no terceiro quadrante, que houve quatro reduções significativas, tanto do número de regra quanto do número de condições por regra, nos conjuntos exemplos maiores, em relação ao indutor C4.5. A maioria dos casos corresponde a uma maior precisão de XRULER. No caso de C4.5, XRULER consegue, em algumas situações, melhorar a precisão em exemplos não vistos, decrementando tanto o número de regras quanto o número de condições por regra.

8.6 Considerações Finais

> Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.
>
> —*Winston Churchill*

Quanto maior a compreensão sobre as estruturas fundamentais usadas por classificadores, mais adequadamente pode-se aplicar ou alterá-las com base no conhecimento do domínio. De forma similar, quanto maior a compreensão dos algoritmos de indução e suas suposições, torna-se mais fácil modificá-los.

Em problemas relativamente pequenos, algoritmos simbólicos de AM já desempenham de forma eficaz a indução de classificadores, ou seja, descobrir regras de indução a partir de exemplos fornecidos por especialistas, o que simplifica a transferência de conhecimento dos seres humanos para computadores. Para problemas complexos — de forma similar ao comportamento humano, quando vários especialistas discutem e votam uma questão difícil para chegar a uma solução final — é possível induzir vários classificadores simbólicos e combiná-los em um único através de voto majoritário ou outro mecanismo de votação. Entretanto, esse processo não produz como resultado um classificador simbólico, o que dificulta sua compreensão por seres humanos.

Para solucionar esta questão, neste capítulo foi proposta uma metodologia que tem como objetivo promover a combinação de classificadores simbólicos em um classificador final também simbólico. Isto foi obtido através da transformação de todos os classificadores induzidos por diferentes indutores em um conjunto de regras colocadas no formato padrão PBM. Após a conversão para o formato PBM, as regras podem ser filtradas, eliminando-se aquelas regras que não sejam significativas ou de interesse para o usuário final. Adicionalmente, um algoritmo de cobertura pode ser aplicado a um conjunto de regras induzidas para se obter um classificador simbólico final, ou seja, um conjunto final de regras que cobre os exemplos de treinamento. Após isso, o classificador final obtido pode ser avaliado quanto à precisão ou outros aspectos que sejam considerados relevantes.

Recentemente, muito esforço tem sido empregado no desenvolvimento de ferramentas que auxiliem a utilização de técnicas de AM e DM. Entretanto, nessas ferramentas — tais como MineSet™, Weka e a biblioteca $\mathcal{MLC}$++ mencionadas na Seção 3.4 — não existe a possibilidade de induzir e avaliar ou combinar diferentes classificadores simbólicos de forma automática. Na realidade, unificar a linguagem de representação do conceito induzido por vários algoritmos de AM, bem como combinar classificadores simbólicos num classificador final também simbólico é um tópico recentemente de pesquisa. Ambas questões foram tratadas neste trabalho, no

qual foi proposto o sistema XRULER assim como foram realizados experimentos para avaliar seu desempenho.

Nos experimentos realizados em dez conjuntos de exemplos, foram utilizados os indutores C4.5 e CN2. O algoritmo de cobertura de XRULER utilizou dois critérios para selecionar a melhor regra: precisão de Laplace e novidade. Em média, os resultados obtidos mostraram um aumento da precisão obtido por XRULER em comparação com os indutores C4.5 e CN2. O ganho de precisão foi maior para XRULER utilizando a precisão de Laplace do que a novidade.

Um outro benefício mostrado pelo sistema XRULER relaciona-se ao número de regras obtidas pelo algoritmo de cobertura. Em média, houve uma redução para XRULER utilizando a precisão de Laplace, sendo significativa (acima de 60%) para XRULER utilizando a novidade, com uma redução da taxa de erro acima de 0,5% para C4.5 e de quase 10% para CN2. Em experimentos realizados por Todorovski, Flach & Lavrač (2000), nos quais houve apenas uma substituição do critério de seleção da melhor regra no indutor CN2, em média, o emprego da novidade reduziu em nove vezes o número de regras não ordenadas induzidas pelo indutor CN2, com uma aumento do erro em torno de 5%. Isso parece comprovar o fato que é possível combinar classificadores simbólicos num classificador final também simbólico, com aumento da precisão e diminuição do número de regras obtidas.

Considerando o número de condições por regra, em média, o sistema XRULER proporcionou uma diminuição acima de 8% em comparação com o indutor C4.5, enquanto houve um aumento de quase 34% em comparação com o indutor CN2. Entretanto, é importante notar que, em média, o número de condições por regra encontra-se dentro do sugerido pelo postulado da compreensibilidade proposto por Michalski (1983a).

De qualquer forma, o ganho de precisão e a redução do número de regras (com pequeno aumento, em média, do número de condições por regra) podem ser considerados avanços no sentido de uma maior compreensibilidade, por seres humanos, do conjunto final de regras. Entretanto, é importante salientar que o número de regras ou condições consiste em medidas que avaliam a complexidade sintática e não semântica do conhecimento induzido (Pazzani 2000).

Capítulo 9

Conclusões

> One never notices what has been done. One can only see what remains to be done.
>
> —*Marie Curie*

Nesta tese foram apresentados alguns tópicos sobre Aprendizado de Máquina, uma área de pesquisa em Inteligência Artificial (Monard & Baranauskas 2000). Entre esses tópicos, foram descritos métodos que permitem avaliar o desempenho de um algoritmo, comparação de algoritmos, seleção e composição de atributos como também combinação de classificadores. O objetivo desta tese concentrou-se em *classificadores simbólicos*, propondo-se uma maneira de combiná-los em um classificador final também simbólico. Na próxima seção é apresentado um resumo dos resultados e contribuições principais desta tese, finalizando com alguns comentários gerais sobre a área de AM.

9.1 Resumo dos Resultados e Contribuições Principais

> Science is a self-correcting process. To be accepted, new ideas must survive the most rigorous standards of evidence and scrutiny.
>
> —*Carl Sagan*

No Capítulo 2 foram definidos alguns conceitos e termos básicos sobre Aprendizado de Máquina, utilizados no decorrer deste trabalho. Normalmente, tais definições encontram-se em diversos trabalhos da área de AM, utilizando notações distintas. Já a compilação apresentada nesse capítulo foi efetuada sob uma perspectiva unificada, com o objetivo de facilitar a compreensão desses conceitos e termos.

No Capítulo 3 foram descritos diversos métodos de amostragem, normalmente utilizados para estimar o desempenho de um algoritmo. Com base nisso, foi escolhida a metodologia de comparação entre dois algoritmos, que é utilizada nos experimentos efetuados e reportados no decorrer desta tese.

No Capítulo 4 foi proposta uma perspectiva mais simples para o processo de KDD, composta apenas por três etapas: Pré-processamento, Mineração de Dados e Pós-processamento. Já o processo de KDD proposto em Fayyad, Piatetsky-Shapiro & Smyth (1996b) possui nove etapas ao passo que o processo proposto por Weiss & Indurkhya (1998) possui quatro etapas. Embora possuindo etapas em comum com esses processos, acreditamos que a proposta apresentada nesse capítulo torne mais simples a compreensão do processo de KDD.

No Capítulo 5 foi efetuada uma revisão sobre uma atividade de pré-processamento, a seleção de atributos, também conhecida como FSS. Foram descritas três abordagens geralmente empregadas para a seleção de atributos: embutida, filtro e *wrapper*. Experimentos que permitem avaliar as abordagens filtro e *wrapper* foram por nós conduzidos. Como resultado dos experimentos sobre FSS, uma conclusão deste trabalho é que filtros devem ser considerados antes de se cogitar a utilização de *wrappers*, no caso de existirem muitos atributos para descrever os exemplos. O uso de filtros foi, em média, 10^3 a 10^4 vezes mais rápido do que *wrappers*, sem provocar uma perda de precisão significativa quando comparados com o indutor padrão.

Normalmente, pesquisas sobre FSS avaliam apenas a precisão do classificador final obtido. Neste trabalho também avaliamos FSS sob a perspectiva de compreensibilidade do classificador simbólico induzido. Sob essa perspectiva, os experimentos utilizando FSS com o indutor CN2 mostraram um aumento no número de regras e no número de condições nas regras. Além disso, a proporção de atributos selecionados foi inversamente proporcional à quantidade de regras induzidas. Isso indica que, quando o classificador final deve ser avaliado por um ser humano, o processo de FSS deve ser efetuado de forma cuidadosa. Trabalhos futuros podem continuar o aqui iniciado, avaliando se a compreensibilidade é também penalizada utilizando outros classificadores simbólicos.

No Capítulo 6 foi efetuada uma revisão sobre uma atividade de pré-processamento, a composição de atributos, também conhecida como indução construtiva. Foram descritas quatro abordagens empregadas para IC: guiada pelos dados, pela hipótese, pelo conhecimento e multi-estratégia. Neste trabalho foi proposta uma metodologia para composição de atributos guiada pelo conhecimento. Os resultados dos experimentos utilizando essa metodologia, mostram que, mesmo tendo o auxílio do usuário/especialista, é difícil construir atributos derivados que sejam realmente relevantes para aprender o conceito embutido nos conjuntos de exemplos analisados.

É possível que isso seja devido ao fato que os dados já tenham sido pré-processados de forma tal que os atributos originais são, por si só, os mais relevantes. Como trabalhos futuros, para aplicações de descoberta de conhecimento, deve-se ir além dos dados disponíveis em repositórios e trabalhar com conjuntos de exemplos que não tenham sido pré-processados.

No Capítulo 7 foi visto que o *bias* de AM de um indutor consiste em certas suposições e escolhas por ele efetuadas na busca de uma solução. Embora o *bias* possa ser visto como algo indesejável, ele é necessário para o processo de aprendizado. Uma forma de melhorar o *bias* no processo de aprendizado consiste na combinação de classificadores, formando *ensembles*. Em geral, *ensembles* mostraram ser eficazes quando aplicados a indutores instáveis, tais como indução de regras, árvores de decisão ou redes neurais. Apesar do ganho em precisão, isso traz outros tipos de problemas, entre eles o fato que a combinação de classificadores simbólicos em um *ensemble* resulta em um classificador não simbólico, sendo difícil sua compreensão por seres humanos.

No Capítulo 8 foi proposto e implementado o sistema XRULER que tem como objetivo fazer uso dos benefícios da combinação de classificadores simbólicos mantendo o classificador final obtido também simbólico (Baranauskas & Monard 2001). Para isso, inicialmente foi definido o formato padrão de regras PBM em um trabalho conjunto efetuado entre Prati, Baranauskas & Monard (2001b). Tal formato fornece uma perspectiva unificada sob a qual todo classificador simbólico pode ser convertido e analisado. O formato PBM foi adotado no sistema XRULER, proposto neste trabalho, assim como em outros trabalhos do LABIC que também atuam no projeto DISCOVER.

Dentre outros componentes, o sistema XRULER conta com um algoritmo de cobertura a ser aplicado a conjuntos de regras induzidas por diversos classificadores e/ou utilizando amostras diferentes de exemplos para se obter um classificador simbólico final, ou seja, um conjunto final de regras que cubra os exemplos de treinamento. Após isso, o classificador final obtido pode ser avaliado quanto à precisão ou outros aspectos que sejam considerados relevantes. Nos experimentos com o sistema XRULER, no algoritmo de cobertura foram utilizados dois critérios para selecionar a melhor regra em cada iteração: precisão de Laplace e novidade. Em média, os resultados mostraram um aumento da precisão obtido por XRULER em comparação com os indutores C4.5 e CN2. O ganho de precisão foi maior para XRULER utilizando a precisão de Laplace do que a novidade.

Um outro benefício mostrado pelo sistema XRULER relaciona-se ao número de regras obtidas pelo algoritmo de cobertura. Em média, houve uma redução para XRULER utilizando a precisão de Laplace, sendo significativa (acima de 60%) para XRULER utilizando a novidade, com uma

redução da taxa de erro acima de 0,5% para C4.5 e de quase 10% para CN2. Embora os conjuntos de exemplos sejam diferentes e considerando apenas como um ponto de referência, é interessante notar que XRULER obteve melhores resultados quando comparado aos experimentos realizados por Todorovski, Flach & Lavrac (2000) com o indutor CN2 utilizando novidade, ou seja, sem fazer uso da combinação de classificadores, como proposto nesta tese. Esse fato parece comprovar que é possível combinar classificadores simbólicos num classificador final também simbólico, com aumento da precisão e diminuição do número de regras obtidas.

Considerando o número de condições por regra, em média, o sistema XRULER proporcionou uma diminuição acima de 8% em comparação com o indutor C4.5, enquanto houve um aumento de quase 34% em comparação com o indutor CN2. Entretanto, é importante notar que, em média, o número de condições por regra encontra-se dentro do sugerido pelo postulado da compreensibilidade de Michalski (1983a).

Mesmo assim, o ganho de precisão e a redução do número de regras (com pequeno aumento, em média, do número de condições por regra) pode ser considerado um avanço no sentido de fornecer uma maior compreensibilidade do conjunto final de regras por seres humanos.

Para o futuro pode-se sugerir o desenvolvimento de um algoritmo de cobertura que utilize uma estratégia menos gulosa ou ainda que o critério de seleção de regras inclua o grau de interesse para o usuário (por exemplo, definindo uma ordem ou relação entre os atributos mais relevantes para o usuário).

9.2 Considerações Finais

> Gentlemen, what is easier than to make this egg stand on end which you said was impossible? It is the simplest thing in the world. Anybody could have done it — after he had known how.
>
> —*Columbus*

A área de Extração de Conhecimento é relativamente nova e ferramentas que suportem todo o processo são raras. Assim, nos concentramos em um dos tópicos que desafia pesquisadores de KDD, que consiste na utilização de técnicas de Aprendizado de Máquina.

Um anúncio antigo de uma empresa fotográfica afirmava "você pressiona o botão, nós fazemos o resto". Seria interessante se em AM e KDD fosse possível fazer o mesmo: coletar dados e deixar que o algoritmo de indução fizesse o resto. Entretanto, esse sonho ainda parece remoto. Uma grande quantidade de esforço é necessário ao selecionar os exemplos, escolher os atributos, integrar o conhecimento prévio sobre domínio, selecionar o método que fornece a melhor repre-

sentação, avaliar os resultados e repetir todo esse processo. Langley & Simon (1995) afirmam, sobre os projetos que inspecionaram, que "muito de seu poder não é proveniente do método de indução específico, mas da formulação apropriada dos problemas e sua representação, que tornam o aprendizado tratável".

Aprendizado de Máquina pode nunca substituir a engenharia do conhecimento como técnica para construir sistemas baseados em conhecimento. Entretanto, um progresso significativo na automação já foi feito. Ao que tudo indica, a indução de regras e outros métodos de aprendizado se tornarão progressivamente prevalentes, à medida que seus benefícios se tornarem melhor compreendidos.

Fazer previsões sobre o futuro do Aprendizado de Máquina é tentador, mas extremamente difícil. Como citado por Dietterich (1997b), "é uma época extremamente excitante para se trabalhar em Aprendizado de Máquina".

Apêndice A

Abreviaturas

Logic is the beginning of wisdom; not the end.

—Spock (Star Trek VI)

Sigla	Significado
AM	Aprendizado de Máquina
$\mathcal{CI}$	Algoritmo *Column Importance*
CV	Validação cruzada (*cross-validation*)
DCI	Indução Construtiva guiada pelos Dados (*Data-driven Constructive Induction*)
DM	Mineração de Dados (*Data Mining*)
DNA	Ácido Desoxirribonucléico
EOS	Sistema de Observação da Terra (*Earth Observing System*)
FSS	Seleção de um Subconjunto de Atributos (*Feature Subset Selection*)
HCI	Indução Construtiva guiada por Hipótese (*Hypothesis-driven Constructive Induction*)
IB	Algoritmo Baseado em Exemplos (*Instance Based*)
IC	Indução Construtiva
KCI	Indução Construtiva guiada pelo Conhecimento (*Knowledge-driven Constructive Induction*)
KDD	Extração de Conhecimento de Bases de Dados (*Knowledge Discovery in Databases*)

(continua na próxima página)

(continuação da página anterior)

Sigla	Significado
LABIC	Laboratório de Inteligência Computacional (ICMC-USP)
LRC	Linguagem de Representação de Conhecimento do Domínio
LRE	Linguagem de Representação de Exemplos
LRH	Linguagem de Representação de Hipóteses
MCI	Indução Construtiva Multi-estratégia (*Multistrategy Constructive Induction*)
NASA	Agência Espacial Norte Americana (*National Space Agency*)
NB	Algoritmo *naïve* Bayes
NN	Algoritmo Vizinhos mais Próximos (*Nearest Neighbors*)
$\mathcal{MLC}$++	Biblioteca de Aprendizado de Máquina em $\mathcal{C}$++ (*Machine Library in* $\mathcal{C}$++)
OC1	Algoritmo Classificador Oblíquo 1(*Oblique Classifier 1*)
SQL	Linguagem de Consulta Estruturada (*Structured Query Language*)
TDIDT	Indução de Árvores de Decisão (*Top Down Induction of Decision Trees*)
TM	Mineração de Texto (*Text Mining*)

Apêndice B

Definição de Símbolos

The beginning of wisdow is the definition of terms.

—*Socrates*

Notação	Descrição
m	Número de atributos num conjunto de exemplos
n	Número de exemplos num conjunto de exemplos
T	Conjunto de dados
T'	Conjunto de dados derivado utilizando IC
T_i	Exemplo i do conjunto de dados T
x_i	Os atributos (sem a classe) do exemplo i
y_i	A classe do exemplo i
X_i	Atributo i do conjunto de dados T
Y	Classe do conjunto de dados T
k	Número de classes de um conjunto de exemplos
C_i	Classe i de um conjunto de exemplos
$f(x_i)$	Função objetivo que rotula o exemplo i, normalmente desconhecida
$h(x_i)$	Hipótese que aproxima a função objetivo ao rotular o exemplo i ($h(x_i) \approx f(x_i)$)
$\parallel E \parallel$	$= 1$ se expressão E é verdadeira $= 0$ se expressão E é falsa

(continua na próxima página)

(continuação da página anterior)

Notação	Descrição
$\mathcal{R}$	Regra
r	Número de partições (*folds*) de um conjunto de dados
covered(T_i,$\mathcal{R}$)	= verdadeiro se exemplo T_i é coberto corretamente pela regra $\mathcal{R}$
	= falso, caso contrário

Referências

ACM (1999). *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99)*, San Diego, CA. ACM.

ACM (2000). *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, Boston, MA. ACM.

Agrawal, R., Imielinski, T. & Swami, A. (1993). Mining associations between sets of items in massive databases. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, Washington D.C., pp. 207–216.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H. & Verkamo, I. (1996). *Fast Discovery and Data Mining*, pp. 307–328. In Fayyad, Piatetsky-Shapiro, Smyth & Uthurusamy (1996).

Agrawal, R. & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Databases*. `http://www.almaden.ibm.com/u/ragrawal/pubs.html`.

Aha, D. W. (1992). Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies 36*, 267–287.

Aha, D. W. (1997). Lazy learning. *Artificial Intelligence Review 11*, 7–10.

Aha, D. W. (1998). *Feature Weighting for Lazy Learning Algorithms*, Chapter 2. In Liu & Motoda (1998).

Almuallim, H. & Dietterich, T. G. (1991). Learning with many irrelevant features. In *Proceedings AAAI-91*, Anaheim, CA, pp. 547–552. MIT Press, Cambridge, MA.

Almuallim, H. & Dietterich, T. G. (1997). Efficient algorithms for identifying relevant features. `ftp://ftp.cs.orst.edu/pub/tgd/papers`.

Araujo, D. L. A., Lopes, H. S. & Freitas, A. A. (1999). A parallel genetic algorithm for rule discovery in large databases. In *IEEE Systems, Man and Cybernetics Conf.*, Volume 3, pp. 940–945.

Baranauskas, J. A. & Batista, G. E. A. P. A. (2000). O projeto DISCOVER: Idéias iniciais (comunicação pessoal).

Baranauskas, J. A. & Monard, M. C. (1998a). Experimental feature selection using the wrapper approach. In *Proceedings of the International Conference on Data Mining*, Rio de Janeiro, RJ, pp. 161–170. `http://www.fmrp.usp.br/~augusto/`.

Baranauskas, J. A. & Monard, M. C. (1998b). Metodologias para seleção de atributos. *Workshop de Teses e Dissertações do Simpósio Brasileiro de Inteligência Artificial (SBIA)*. `http://www.fmrp.usp.br/~augusto/`.

Baranauskas, J. A. & Monard, M. C. (1999). The $\mathcal{MLC}$++ wrapper for feature subset selection using decision tree, production rule, instance based and statistical inducers: Some experimental results. Technical Report 87, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_87.ps.zip`.

Baranauskas, J. A. & Monard, M. C. (2000a). An approach for extracting symbolic ensembles from databases. See Vaz (2000). (CD-ROM).

Baranauskas, J. A. & Monard, M. C. (2000b). An environment for rule extraction and evaluation from databases. See Monard & Sichman (2000), pp. 187–196.

Baranauskas, J. A. & Monard, M. C. (2000c). Reviewing some machine learning concepts and methods. Technical Report 102, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_102.ps.zip`.

Baranauskas, J. A. & Monard, M. C. (2000d). An unified overview of six supervised symbolic machine learning inducers. Technical Report 103, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_103.ps.zip`.

Baranauskas, J. A. & Monard, M. C. (2001). Combining symbolic classifiers from multiple inducers. *Knowledge-Based Systems*. (submetido).

Baranauskas, J. A., Monard, M. C. & Batista, G. E. A. P. A. (2000). A computational environment for extracting rules from databases. In N. Ebecken & C. A. Brebbia (Eds.), *Proceedings of the Second International Conference on Data Mining*, Cambridge, UK, pp. 321–330.

Baranauskas, J. A., Monard, M. C. & Horst, P. S. (1999a). Evaluation of CN2 induced rules using feature selection. In *Proceedings of the Argentine Symposium on Artificial Intelligence (ASAI/JAIIO/SADIO)*, Buenos Aires, Argentine, pp. 141–154. `http://www.fmrp.usp.br/~augusto/`.

Baranauskas, J. A., Monard, M. C. & Horst, P. S. (1999b). Evaluation of feature selection by wrapping around the CN2 inducer. In *Encontro Nacional de Inteligência Artificial (ENIA/SBC)*, Rio de Janeiro, RJ, pp. 315–326. `http://www.fmrp.usp.br/~augusto/`.

Batista, G. E. A. P. A. (1997). Um ambiente de avaliação de algoritmos de aprendizado de máquina utilizando exemplos. Dissertação de Mestrado, ICMC-USP.

Batista, G. E. A. P. A. (2000). Pré-processamento de dados em aprendizado de máquina supervisionado. Exame de Qualificação de Doutorado, ICMC-USP.

Batista, G. E. A. P. A. (2001). Sintaxe padrão do arquivo de exemplos do projeto DISCOVER. `http://www.icmc.sc.usp.br/~gbatista/SintaxePadraoFinal.htm` (04/06/2001).

Batista, G. E. A. P. A., Carvalho, A. C. P. L. & Monard, M. C. (2000). Applying one-sided selection to unbalanced datasets. In *Proceedings of the Mexican Congress on Artificial Intelligence (MICAI), Lecture Notes in Artificial Intelligence*, pp. 315–325. Springer-Verlag.

Batista, G. E. A. P. A. & Monard, M. C. (1998). Descrição da implementação dos métodos estatísticos e de resampling do ambiente AMPSAM. Technical Report 68, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_68.ps.zip`.

Bauer, E. & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning 36*, 105–139.

Bayardo, R. J. & Agrawal, R. (1999). Mining the most interesting rules. See ACM (1999), pp. 145–154. `http://www.almaden.ibm.com/u/ragrawal/pubs.html`.

Blake, C. L. & Merz, C. J. (1998). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Bloedorn, E. & Michalski, R. S. (1998). Data-driven construtive induction. *IEEE Intelligent Systems 13*(2), 30–37. March/April 1998.

Blum, A. L. & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence 97*(1–2), 245–271.

Breiman, L. (1996a). Arcing classifiers. Technical report, Statistics Department, University of California, `ftp://ftp.stat.berkeley.edu/pub/users/breiman/`.

Breiman, L. (1996b). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breiman, L. (1996c). Bias, variance and arcing classifiers. Technical Report 460, Statistics Department, University of California, `ftp://ftp.stat.berkeley.edu/pub/users/breiman/`.

Breiman, L. (1996d). The heuristics on instability in model selection. Technical report, Statistics Department, University of California, `ftp://ftp.stat.berkeley.edu/pub/users/breiman/`.

Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984). *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth & Books.

Bull, S. (1994). Analysis of attitudes toward workplace smoking restrictions. *Case Studies in Biometry*, 249–271.

Cardie, C. (1993). Using decision trees to improve case-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA, pp. 25–32. Morgan Kaufmann.

Caulkins, C. W. (2000). Aquisição de conhecimento utilizando aprendizado de máquina relacional. Dissertação de Mestrado, ICMC-USP.

Cheeseman, P. & Stutz, J. (1990). Bayesian classification (AutoClass): Theory and results advances in knowledge discovery and data mining. `http://ic.arc.nasa.gov/ic/projects/bayes-group/AutoClass-c-program.html%`.

Clark, P. & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In Y. Kodratoff (Ed.), *Proceedings of the 5th European Conference (EWSL 91)*, pp. 151–163. Springer-Verlag.

Clark, P. & Niblett, T. (1987). Induction in noise domains. In I. Bratko & N. Lavrač (Eds.), *Proceedings of the Second European Working Session on Learning*, Wilmslow, UK, pp. 11–30. Sigma.

Clark, P. & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning 3*(4), 261–283.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, San Francisco, CA, pp. 115–123. Morgan Kaufmann.

Decker, K. M. & Focardi, S. (1995). Technology overview: A report on data mining. Technical Report CSCS TR-95-02, Swiss Scientific Computer Center.

Dietterich, T. G. (1986). Learning at the knowledge level. *Machine Learning 1*(3), 287–315. Reprinted in Shavlik and Dietterich (eds.), 1990. Readings in Machine Learning, Morgan Kaufmann Publishers, Inc.

Dietterich, T. G. (1997a). Limitations on inductive learning (extended abstract). `ftp://ftp.cs.orst.edu/pub/tgd/papers`.

Dietterich, T. G. (1997b). Machine learning research: Four current directions. `ftp://ftp.cs.orst.edu/pub/tgd/papers`.

Dietterich, T. G. (1997c). Statistical tests for comparing supervised classification learning algorithms. `ftp://ftp.cs.orst.edu/pub/tgd/papers`.

Dietterich, T. G. & Bakiri, G. (1995). Solving multicast learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research 2*, 263–286. Reprinted in Shavlik and Dietterich (eds.), 1990. Readings in Machine Learning, Morgan Kaufmann Publishers, Inc.

Dietterich, T. G. & Kong, E. B. (1997). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. `ftp://ftp.cs.orst.edu/pub/tgd/papers`.

Dietterich, T. G. & Michalski, R. S. (1983). *A Comparative Review of Selected Methods for Leaning from Examples*, pp. 41–82. In Michalski, Carbonell & Mitchell (1983).

Domingos, P. (1999). MetaCost: A general method for making classifiers cost-sensitive. See ACM (1999), pp. 155–164.

Dörre, J., Gerstl, P. & Seiffert, R. (1999). Text mining: Finding nuggets in mountains of textual data. See ACM (1999), pp. 398–401.

Efron, B. & Tibshirani, R. (1993). *An Introduction to the Bootstrap*. Chapman & Hall.

Emde, W., Habel, C. U. & Rollinger, C. R. (1983). The discovery of the equator or concept driven learning. In *Proceedings of IJCAI-83*, Karlsruhe, Germany, pp. 455–458.

Famili, A., Shen, W.-M., Weber, R. & Simoudis, E. (1997). Data preprocessing and intelligent data analysis. *Intelligent Data Analysis 1*(1). `http://www.elsevier.com/locate/ida`.

Fayyad, U. M. (1994). Branching on attribute values in decision tree generation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 601–606. American Association for Artificial Intelligence.

Fayyad, U. M., Djorgovski, S. G. & Weir, N. (1996). From digitized images to on-line catalogs: Data mining a sky survey. *AI Magazine 17*(2), 51–66.

Fayyad, U. M. & Irani, K. B. (1992). The attribute-selection problem in decision tree generation. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 104–110. American Association for Artificial Intelligence.

Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (1996b). *From Data Mining to Knowledge Discovery: An Overview*, pp. 1–30. In Fayyad, Piatetsky-Shapiro, Smyth & Uthurusamy (1996).

Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (1996c). From data mining to knowledge discovery in databases. *AI Magazine 17*(3), 37–54.

Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (1996a). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM 39*(11), 27–34.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. & Uthurusamy, R. (Eds.) (1996). *Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA: American Association for Artificial Intelligence.

Fayyad, U. M. & Uthurusamy, R. (Eds.) (1995). *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Menlo Park, CA. American Association for Artificial Intelligence: American Association for Artificial Intelligence.

Felix, L. C. M., Rezende, S. O., Doi, C. Y., de Paula, M. F. & Romanato, M. J. (1998). $\mathcal{MLC}$++ biblioteca de aprendizado de máquina em $\mathcal{C}$++. Technical Report 72, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_72.ps.zip`.

Flach, P. A. (2000). On the state of the art in machine learning: a personal review. Technical Report CSTR-00-018, Department of Computer Science, University of Bristol.

Flach, P. A. & Lavrac, N. (2000). The role of feature construction in inductive rule learning. In L. D. Raedt & S. Kramer (Eds.), *Proceedings of the ICML2000 workshop on Attribute-Value and Relational Learning: crossing the boundaries*, Stanford, USA, pp. 1–11. 17th International Conference on Machine Learning.

Freedman, D., Pisani, R. & Purves, R. (Eds.) (1998). *Statistics* (Third ed.). W. W. Norton & Company.

Freitas, A. A. (1998a). A multi-criteria approach for the evaluation of rule interestingness. In *Proceedings of the International Conference on Data Mining*, Rio de Janeiro, RJ, pp. 7–20.

Freitas, A. A. (1998b). On objective measures of rule surprisingness. In *Principles of Data Mining & Knowledge Discovery: Proceedings of the Second European Symp. Lecture Notes in Artificial Intelligence*, Volume 1510, pp. 1–9.

Freitas, A. A. (1999). On rule interestingness measures. *Knowledge-Based Systems 12*(5–6), 309–315.

Freitas, A. A. & Lavington, S. H. (1998). *Mining Very Large Databases with Parallel Processing*. Kluwer Academic Publishers.

Freund, Y. & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pp. 23–37. Springer-Verlag.

Freund, Y. & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, Lake Tahoe, California, pp. 123–140. Morgan Kaufmann.

Gomes, A. K. (2001). Análise de regras utilizando medidas de avaliação e de interessabilidade do conhecimento simbólico. Exame de Qualificação de Mestrado, ICMC-USP.

Gorman, R. P. & Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks 1*, 75–89.

Han, J. & Cercone, N. (2000). RuleViz: a model for visualizing knowledge discovery process. See ACM (2000), pp. 244–253.

Heckerman, D. (1996). *Bayesian Networks for Knowledge Discovery*, pp. 273–306. In Fayyad, Piatetsky-Shapiro, Smyth & Uthurusamy (1996).

Horst, P. S. (1999). Avaliação do conhecimento adquirido por algoritmos de aprendizado de máquina utilizando exemplos. Dissertação de Mestrado, ICMC-USP.

Horst, P. S. & Monard, M. C. (2000). Um sistema computacional para avaliação de regras induzidas por algoritmos de aprendizado de máquina. See Monard & Sichman (2000), pp. 167–176.

Imamura, C. Y. (2000). Pré-processamento para extração de conhecimento de bases textuais. Exame de Qualificação de Mestrado, ICMC-USP.

Jarvis, J. & Tyson, A. (1981). FOCAS: Faint object classification and analysis system. *Astronomical Journal 86*(476).

John, G., Kohavi, R. & Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Tenth International Conference on Machine Learning*, San Francisco, CA, pp. 167–173. Morgan Kaufmann.

Kearns, M. J. & Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. Ellis Horwood.

Kira, K. & Rendell, L. A. (1992a). The feature selection problem: Traditional methods and a new algorithm. In *Tenth National Conference on Artificial Intelligence*, pp. 129–134. MIT Press.

Kira, K. & Rendell, L. A. (1992b). A practical approach to feature selection. In *Proceedings of the Ninth International Workshop on Machine Learning*, Aberdeen, Scotland, pp. 249–256. Morgan Kaufmann.

Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H. & Verkamo, A. I. (1994). Finding interesting rules from large sets of discovered association rules. In B. K. B. Nabil R. Adam & Y. Yesha (Eds.), *Third International Conference on Information and Knowledge Managmente*, Volume 30, Gaithersburg, Maryland, pp. 401–407. ACM Press.

Kohavi, R. & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence 97*(1–2), 273–324.

Kohavi, R. & Sommerfield, D. (1995). Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. See Fayyad & Uthurusamy (1995), pp. 192–197.

Kohavi, R., Sommerfield, D. & Dougherty, J. (1994). $\mathcal{MLC}$++: A machine learning library in $\mathcal{C}$++. In *Tools with Artificial Intelligence*, pp. 740–743. IEEE Computer Society Press.

Kohavi, R., Sommerfield, D. & Dougherty, J. (1996). Data mining using $\mathcal{MLC}$++: A machine learning library in $\mathcal{C}$++. In *Tools with Artificial Intelligence*, pp. 234–245. IEEE Computer Society Press.

Kononenko, I. (1993). Inductive and bayesian learning in medical diagnosis. *Applied Artificial Intelligence 7*, 317–337.

Kononenko, I. (1994). Estimating attributes: Analysis and extensios of relief. In *Proceedings of the 1994 European Conference on Machine Learning*, Amsterdam, pp. 171–182. Springer-Verlag.

Kowalsky, R. (1979). *Logic for Problem Solving*. North-Holland.

Kubat, M., Bratko, I. & Michalski, R. S. (1998). *A Review of Machine Learning Methods*, pp. 3–69. In Michalski, Bratko & Kubat (1998).

Kubat, M., Holte, R. C. & Matwin, S. (1997). Learning when negative examples abound: One-sided selection. In *Proceedings of the European Conference on Machine Learning (ECML)*, pp. 146–153. Springer-Verlag.

Kubat, M., Holte, R. C. & Matwin, S. (1998). Machine learning for detection of oil spills in satellite radar images. *Machine Learning 30*(2/3), 195–215.

Kubat, M. & Matwin, S. (1997). Addressing the curse of imbalanced training sets: One-sided sampling. In *Proceedings of the Fourteenth International Conference on Machine Learning*, San Francisco, pp. 179–186. Morgan Kaufmann.

Langley, P. (1983). *Rediscovering Chemistry with the* Bacon *System*, pp. 307–330. In Michalski, Carbonell & Mitchell (1983).

Langley, P. (1996). *Elements of Machine Learning*. Morgan Kaufmann Publishers, Inc.

Langley, P., Iba, W. & Thompson, K. (1992). An analysis of bayseian classifiers. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pp. 223–228. AAAI Press and MIT Press.

Langley, P. & Simon, H. A. (1995). Applications of machine learning and rule induction. *Communications of the ACM 38*, 55–64.

Lavrač, N., Flach, P. & Zupan, R. (1999). Rule evaluation measures: A unifying view. In S. Dzeroski & P. Flach (Eds.), *Proceedings of the Ninth International Workshop on Inductive Logic Programming (ILP-99)*, Volume 1634, pp. 74–185. Springer-Verlag. Lecture Notes in Artificial Intelligence.

Lee, H. D. (2000). Seleção e construção de features relevantes para o aprendizado de máquina. Dissertação de Mestrado, ICMC-USP.

Lee, H. D. & Monard, M. C. (2000). Applying knowledge-driven constructive induction: Some experimental results. Technical Report 101, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_101.ps.zip`.

Lee, H. D., Monard, M. C. & Baranauskas, J. A. (1999). Empirical comparison of wrapper and filter approaches for feature subset selection. Technical Report 94, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_94.ps.zip`.

Lee, H. D., Monard, M. C. & Baranauskas, J. A. (2000). A practical approach for knowledge-driven constructive induction. In *Proceedings of the Argentine Symposium on Artificial Intelligence (ASAI/JAIIO/SADIO)*, pp. 71–85.

Lee, H. D., Monard, M. C. & Esteves, S. C. (2000). Indução construtiva guiada pelo conhecimento: um estudo de caso do processamento sêmen diagnóstico. See Monard & Sichman (2000), pp. 157–166.

Lenat, D. B. (1983). *The Role of Heuristics in Learning by Discovery: Three Case Studies*, pp. 243–306. In Michalski, Carbonell & Mitchell (1983).

Liu, H. & Motoda, H. (Eds.) (1998). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers.

Margineantu, D. D. & Dietterich, T. G. (1997). Pruning adaptive boosting. In *Proceedings of the Fourteenth International Conference on Machine Learning*, San Francisco, pp. 211–218. Morgan Kaufmann.

Martins, C. A. (2001). Clustering conceitual em aprendizado de máquina. Exame de Qualificação de Doutorado, ICMC-USP.

Martins, C. A. & Monard, M. C. (2000). Interpretação de clusters utilizando aprendizado de máquina simbólico. See Vaz (2000). (CD-ROM).

Michalski, R. (1978). Pattern recognition as knowledge-guided computer induction. Technical Report 927, Department of Computer Science, University of Illinois, Urbana-Champaign, Ill.

Michalski, R. S. (1983a). *A Theory and Methodology of Inductive Learning*, pp. 83–134. In Michalski, Carbonell & Mitchell (1983).

Michalski, R. S. (1983b). A theory and methodology of inductive learning. *Artificial Intelligence 20*, 111–161.

Michalski, R. S., Bratko, I. & Kubat, M. (Eds.) (1998). *Machine Learning and Data Mining Methods and Applications*. West Sussex, England: John Wiley & Sons Ltd.

Michalski, R. S., Carbonell, J. G. & Mitchell, T. M. (Eds.) (1983). *Machine Learning: An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann.

Michalski, R. S. & Kaufman, K. A. (1998). *Data Mining and Knowledge Discovery: A Review of Issues and a Multistrategy Approach*, pp. 71–112. In Michalski, Bratko & Kubat (1998).

Michie, D. (1986). Current developments in expert systems. In *Proceedings of the Second Australian Conference on Applications of Expert Systems*, Sydney, Australia, pp. 163–182.

Michie, D. (1988). Machine learning in the next five years. In *Proceedings of the Third European Working Session on Learning EWSL-88*, Glasgow, London, Pitman, pp. 107–122. Pitman.

Michie, D., Spiegelhalter, D. J. & Taylor, C. C. (Eds.) (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood.

Milaré, C. R. (2000). Extracao de conhecimneto de redes neurais. Exame de Qualificação de Doutorado, ICMC-USP.

Minsky, M. L. & Papert, S. (1988). *Perceptrons : an Introduction to Computational Geometry*. The MIT Press.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence 18*, 203–226. Reprinted in Shavlik and Dietterich (eds.), 1990. Readings in Machine Learning, Morgan Kaufmann Publishers, Inc.

Mitchell, T. M. (1998). *Machine Learning*. McGraw–Hill.

Mitchell, T. M., Utgoff, P. E. & Banerji, R. (1983). *Leaning by Experimentation: Acquiring and Refining Problem-Solving Heuristics*, pp. 137–190. In Michalski, Carbonell & Mitchell (1983).

Monard, M. C. & Baranauskas, J. A. (2000). Aplicações de inteligência artificial: Uma visão geral. In *I Congresso de Lógica Aplicada à Tecnologia (LAPTEC)*, Rio de Janeiro, pp. 339–348.

Monard, M. C., Batista, G. E. A. P. A., Kawamoto, S. & Pugliesi, J. B. (1997). Uma introdução ao aprendizado simbólico de máquina. `ftp://labic.icmc.sc.usp.br/didatico/PostScript/ML.ps`.

Monard, M. C., Caulkins, C. W., Baranauskas, J. A., Oliveira, R. B. T. & Rezende, S. O. (1999). Data preparation, reduction and prediction in the context of data mining: A case study with insurance policies. Technical Report 81, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_81.ps.zip`.

Monard, M. C. & Sichman, J. S. (Eds.) (2000). *Proceedings of the IBERAMIA/SBIA*, Atibaia, SP.

Morik, K., Wrobel, S., Jörg-Uwe & Emde, W. (1993). *Knowledge Acquisition and Machine Learning: Theory, Methods, and Applications*. Harcourt Brace & Company, Publishers.

Moses, L. E. (Ed.) (1986). *Think and Explain with Statistics*. Addison–Wesley.

Muggleton, S. (1987). Duce, and oracle-based approach to constructive induction. In *Proceedings of IJCAI-87*, Milan, Italy, pp. 287–292. Morgan Kaufmann.

Muggleton, S. & Firth, J. (1999). Progol4.4: A tutorial introdution. `ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/progol4.4`.

Murthy, S. K., Kasif, S. & Salzberg, S. L. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research 2*(1), 1–32. `http://www.cs.jhu.edu/~salzberg/jair94.ps`.

MySQL (2000). The MySQL server. `http://www.mysql.com`.

Opitz, D. & Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research 11*, 169–198.

Pagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning 5*(1), 71–100.

Pazzani, M. J. (2000). Knowledge discovery from data? *IEEE Intelligent Systems 13*, 10–12. March/April 2000.

Piatetsky-Shapiro, G. (1989). Knowledge discovery in real databases: A report on the UJCAI-89 workshop. *AI Magazine 11*(5), 68–70.

Piatetsky-Shapiro, G. (1991). *Discovery Analysis and Presentation of Strong Rules*, Chapter 13. American Association for Artificial Intelligence.

Pila, A. D. (2001). Seleção de de atributos relevantes para aprendizado de máquina utilizando a abordagem de rough sets. Dissertação de Mestrado, ICMC-USP.

Prati, R. C., Baranauskas, J. A. & Monard, M. C. (1999). BibView: Um sistema para auxiliar a manutenção de registros para o BibTeX. Technical Report 95, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_95.ps.zip`.

Prati, R. C., Baranauskas, J. A. & Monard, M. C. (2001a). Extração de informações padronizadas para a avaliação de regras induzidas por algoritmos de aprendizado de máquina simbólico. Technical Report 145, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip`.

Prati, R. C., Baranauskas, J. A. & Monard, M. C. (2001b). Uma proposta de unificação da linguagem de representação de conceitos de algoritmos de aprendizado de máquina simbólicos. Technical Report 137, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip`.

Pugliesi, J. B. (2001). O pós-processamento em extração de conhecimento de bases de dados. Exame de Qualificação de Doutorado, ICMC-USP.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning 1*, 81–106. Reprinted in Shavlik and Dietterich (eds.), 1990. Readings in Machine Learning, Morgan Kaufmann Publishers, Inc.

Quinlan, J. R. (1990). Learning logical definition from relations. *Machine Learning 5*, 239–266.

Quinlan, J. R. (1993). C4.5*: Programs for Machine Learning*. Morgan Kaufmann. San Francisco, CA.

Quinlan, J. R. (1996). Bagging, boosting and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 725–730. American Association for Artificial Intelligence.

Rathjens, D. (1996). *MineSet™ User's Guide*. Silicon Graphics, Inc.

Rezende, S. O. & Pugliesi, J. B. (1998). Aquisição de conhecimento explícito ou manual. Technical Report 37, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_37.ps.zip`.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review 65*, 386–408.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*. The MIT Press.

Russel, S. & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Salzberg, S. L. (1995a). Locating protein coding regions in human dna using a decision tree algorithm. *Journal of Computational Biology 2*(3), 473–485.

Salzberg, S. L. (1995b). On comparing classifiers: A critique of current research and methods. Technical Report JHU-95/06, Department of Computer Science, Johns Hopkins University. `http://www.cs.jhu.edu/~salzberg/critique.ps`.

Salzberg, S. L., Chandar, R., Ford, H., Murthy, S. & White, R. (1995). Decision trees for automated identification of cosmic ray hits in Hubble space telescope images. *Publications of the Astronomical Society of the Pacific 107*, 1–10. `ftp://ftp.cs.jhu.edu/pub/salzberg/starcr.ps.gz`.

Sammut, C., Hurst, S., Kedzier, D. & Michie, D. (1992). Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, pp. 385–393. Morgan Kaufmann.

Schaffer, C. (1994). A conservation law for generalization performance. In W. W. Cohen & H. Hirsh (Eds.), *Proceedings of the Eleventh International Conference on Machine Learning*, New Brunswick, New Jersey, pp. 259–265. Morgan Kaufmann.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning 5*(2), 197–227.

Simon, H. (1983). *Why should machines learn?*, pp. 25–37. In Michalski, Carbonell & Mitchell (1983).

Teller, A. & Veloso, M. (1995). Program evolution for data mining. *International Journal of Expert Systems 8*(3), 213–236.

Todorovski, L., Flach, P. & Lavrac, N. (2000). A report on experiments with weighted relative accuracy in CN2. Technical Report CSTR-00-003, Department of Computer Science, University of Bristol. `http://www.compsci.bristol.ac.uk/Tools/Reports/Ps/`.

Todorovski, L., Flach, P. & Lavrač, N. (2000). Predictive performance of weighted relative accuracy. In D. A. Zighed, J. Komorowski, & J. Zytkow (Eds.), *4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2000)*, pp. 255–264. Springer-Verlag.

Vaz, L. E. (Ed.) (2000). *Proceedings 21st Iberian Latin American Congress on Computational Methods in Engineering (CILAMCE)*, Rio de Janeiro. LMC/EPUSP. (CD-ROM).

Wall, L., Christiansen, T. & Schwartz, R. L. (1996). *Programming Perl*. O'Reilly & Associates, Inc.

Weiss, S. M. & Indurkhya, N. (1998). *Predictive Data Mining: A Practical Guide*. San Francisco, CA: Morgan Kaufmann.

Weiss, S. M. & Kulikowski, C. A. (1991). *Computer Systems that Learn*. San Mateo, CA: Morgan Kaufmann.

Winston, P. H. (1970). Learning structural descriptions from examples. Technical Report AI-TR-231, MIT, Cambridge, Mass.

Wnek, E. B. J. & Michalski, R. S. (1993). Multistrategy constructive induction. In *Proceedings of the Second International Workshop on Machine Learning – ML93*, San Francisco, pp. 188–203. Morgan Kaufmann.

Wnek, J. & Michalski, R. S. (1994). Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning 14*(2), 139–168.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks 5*, 241–259.

Yang, L. (2000). Interactive exploration of very large relational datasets through 3D dynamic projections. See ACM (2000), pp. 236–243.

Yoon, J. P. & Kerschberg, L. (1993). A framework for knowledge discovery and evolution databases. *IEEE Transaction on Knowledge and Data Engineering 5*(6), 973–979.

Índice