**Um Ambiente de Avaliação de
Algoritmos de Aprendizado de Máquina
Utilizando Exemplos¹**

Gustavo Enrique de Almeida Prado Alves Batista

Orientação:

Profa. Dra. Maria Carolina Monard

Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos, da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Ciências – Área: Ciências de Computação e Matemática Computacional.

**USP - São Carlos
Setembro de 1997**

¹Trabalho realizado com auxílio do CNPq.

*À minha avó,
Margarida.*

*Aos meus pais,
Joselito e Margarida,*

pelo apoio, dedicação e incentivo.

Agradecimentos

À professora Maria Carolina Monard, pela orientação e, principalmente, pela amizade durante os últimos anos;

Ao amigo José Pacheco de Almeida Prado por ter me incentivado a seguir a carreira acadêmica;

Aos amigos Cláudia Milaré, Gustavo Rosalém e Ricardo Freitas;

Aos funcionários do Instituto de Ciências Matemáticas de São Carlos;

Ao CNPq pelo apoio financeiro;

Àqueles que de um modo ou de outro contribuíram para a realização deste trabalho.

Resumo

Um sistema de aprendizado supervisionado é um programa capaz de realizar decisões baseado na experiência contida em casos resolvidos com sucesso. As regras de classificação induzidas por um sistema de aprendizado podem ser analisadas segundo dois critérios: a complexidade dessas regras e o erro de classificação sobre um conjunto independente de exemplos.

Sistemas de aprendizado têm sido desenvolvidos na prática utilizando diferentes paradigmas incluindo estatística, redes neurais, bem como sistemas de aprendizado simbólico proposicionais e relacionais. Diversos métodos de aprendizado podem ser aplicados à mesma amostra de dados e alguns deles podem desempenhar melhor que outros. Para uma dada aplicação, não existem garantias que qualquer um desses métodos é necessariamente o melhor. Em outras palavras, não existe uma análise matemática que possa determinar se um algoritmo de aprendizado irá desempenhar melhor que outro. Desta forma, estudos experimentais são necessários.

Neste trabalho nos concentramos em uma tarefa de aprendizado conhecida como classificação ou predição, na qual o problema consiste na construção de um procedimento de classificação a partir de um conjunto de casos no qual as classes verdadeiras são conhecidas, chamado de aprendizado supervisionado. O maior objetivo de um classificador é ser capaz de prever com sucesso a respeito de novos casos. A performance de um classificador é medida em termos da taxa de erro. Técnicas experimentais para estimar a taxa de erro verdadeira não somente provêem uma base para comparar objetivamente as performances de diversos algoritmos de aprendizado no mesmo conjunto de exemplos, mas também podem ser uma ferramenta poderosa para projetar um classificador. As técnicas para estimar a taxa de erro são baseadas na teoria estatística de *resampling*.

Um ambiente chamado AMPSAM foi implementado para ajudar na aplicação dos métodos de *resampling* em conjuntos de exemplos do mundo real. AMPSAM foi projetado como uma coleção de programas independentes, os quais podem interagir entre si através de *scripts* pré-definidos ou de novos *scripts* criados pelo usuário. O ambiente utiliza um formato padrão para arquivos de exemplos o qual é independente da sintaxe de qualquer algoritmo. AMPSAM também inclui ferramentas para particionar conjuntos de exemplos em conjuntos de treinamento e teste utilizando diferentes métodos de *resampling*. Além do método *holdout*, que é o estimador de taxa de erro mais comum, AMPSAM suporta os métodos *n-fold cross-validation* — incluindo o *learning-one-out* — e o método *bootstrap*. As matrizes de confusão produzidas em cada iteração utilizando conjuntos de treinamento e teste podem ser fornecidas a um outro sistema implementado chamado SMEC. Este sistema calcula e mostra graficamente algumas das medidas descritivas mais importantes relacionadas com tendência central e dispersão dos dados.

Este trabalho também relata os resultados experimentais a respeito de medidas do erro de classificação de três classificadores proposicionais e relacionais bem conhecidos, utilizando ambos os sistemas implementados, em diversos conjuntos de exemplos freqüentemente utilizados em pesquisas de Aprendizado de Máquina.

Abstract

A learning system is a computer program that makes decisions based on the accumulative experience contained in successfully solved cases. The classification rules induced by a learning system are judged by two criteria: their classification error on an independent test set and their complexity.

Practical learning systems have been developed using different paradigms including statistics, neural nets, as well as propositional and relational symbolic machine learning. Several learning methods can be applied to the same sample data and some of them may do better than others. Still, for a given application, there is no guarantee that any of these methods will work or that any single method is necessarily the best one. In other words, there is not a mathematical analysis method that can determine whether a learning system algorithm will work well. Hence, experimental studies are required.

In this work we confine our attention to the learning task known as classification or prediction, where the problem concerns the construction of a classification procedure from a set of data for which the true classes are known, and is termed supervised learning. The overall objective of a classifier is to be able to predict successfully on new data.

Performance is measured in terms of the error rate. Error rate estimation techniques not only provide a basis for objectively comparing the error rate of several classifiers on the same data and then estimating their future performance on new data, but they can also be a powerful tool for designing a classifier. The techniques of error rate estimation are based on statistical resampling theory.

In this work, rules induced complexity of propositional and relational learning systems as well as several resampling methods to estimate the true error rate are discussed. An environment called AMPSAM has been implemented to aid in the application of resampling methods to real world data sets. AMPSAM consists of a collection of interdependent programs that can be bound together either by already defined or by new user defined scripts. The environment uses a common file format for data sets which is independent of any specific classifier scheme. It also includes facilities for splitting data sets up into test and training sets using different methods. Besides holdout, which is the most common accuracy estimation method, AMPSAM supports n-fold cross-validation — including leaving-one-out — and bootstrap.

The confusion matrices produced in each run using those test and training sets can be input to another system called SMEC. This system calculates and graphically displays some of the most important descriptive measures related to central tendency and dispersion of those data. This work also reports the results of experiments measuring the classification error of three well known propositional and relational classifiers, using the implemented systems, on several data sets commonly used in Machine Learning research.

Sumário

1	Introdução	1
2	Sistemas de Aprendizado	4
2.1	Considerações Iniciais	4
2.2	O Modelo de Classificação	5
2.3	Paradigmas de Aprendizado por Exemplos	7
2.3.1	Paradigma Simbólico	7
2.3.2	Paradigma Estatístico	8
2.3.3	Paradigma Instance-based	8
2.3.4	Paradigma Conexionista	9
2.3.5	Paradigma Genético	10
2.4	Considerações Finais	12
3	Técnicas para Medir a Performance de Sistemas de Aprendizado	13
3.1	Considerações Iniciais	13
3.2	A Taxa de Erro	13
3.3	Tipos de Erros	14
3.4	O Modelo Probably Approximately Correct (PAC)	18
3.5	Técnicas para Estimar a Taxa de Erro Verdadeira	25
3.5.1	Estimação da Taxa de Erro Através de Treinar-e-Testar	26

3.5.2	Métodos Resampling	29
3.6	Problemas Comuns em Classificação	33
3.7	Critérios de Avaliação de Sistemas de Aprendizado	34
3.8	Considerações Finais	35
4	Sistemas de Aprendizado Simbólico Proposicional	36
4.1	Considerações Iniciais	36
4.2	Linguagens de Descrição	37
4.3	Árvores de Decisão	39
4.3.1	Escolha do Atributo	42
4.3.2	Poda de Árvores de Decisão	46
4.3.3	Limitações dos Métodos de Indução de Árvores de Decisão	47
4.4	Indução de Regras de Decisão	49
4.4.1	Bias de Linguagem	51
4.4.2	Bias de Busca	52
4.4.3	Bias para Evitar Overfitting	55
4.5	Indução Construtiva	56
4.6	Considerações Finais	58
5	Programação Lógica Indutiva	60
5.1	Considerações Iniciais	60
5.2	Conceitos Básicos	61
5.2.1	Sintaxe da Notação Clausal	62
5.2.2	Semântica da Lógica Clausal	65
5.2.3	Prova por Resolução	69
5.3	Especificação do Problema de PLI	70
5.4	PLI como um Problema de Busca	71

5.5	A Relação θ – <i>subsoma</i>	73
5.6	Indução por Generalização	76
5.7	Indução por Especialização	82
5.8	Considerações Finais	86
6	O Ambiente Computacional AMPSAM	88
6.1	Considerações Iniciais	88
6.2	O Projeto do Ambiente AMPSAM	89
6.3	Uma Visão Geral do Ambiente AMPSAM	91
6.4	As Ferramentas de Particionamento de Exemplos	92
6.5	As Ferramentas de Manipulação Sintática	94
6.5.1	A Sintaxe Padrão para Arquivos de Exemplos	94
6.5.2	As Ferramentas para Conversão Sintática	98
6.6	Um Exemplo de Conversão Sintática	101
6.7	Comentários a Respeito do Utilitário Awk	105
6.8	Trabalhos Relacionados	106
6.9	Considerações Finais	107
7	Avaliação Experimental de alguns Algoritmos Simbólicos	108
7.1	Considerações Iniciais	108
7.2	Os Conjuntos de Exemplos Utilizados	109
7.3	Complexidade do Modelo e Parâmetros do Modelo	111
7.4	Resultados da Taxa de Erro Aparente	113
7.5	Como Foram Realizadas as Medidas	114
7.6	O Sistema SMEC	115
7.6.1	Descrição do Sistema SMEC	116
7.6.2	As Medidas Calculadas pelo Sistema SMEC	117

7.7	Resultados Experimentais	119
7.8	Problemas Comuns na Avaliação de Performance	125
7.8.1	Baixa Precisão Relacionada com Conjuntos de Exemplos Ruins .	126
7.8.2	Erros Cometidos no Processo de Avaliação	127
7.9	Qual Sistema de Aprendizado é Melhor?	129
7.10	Considerações Finais	132
8	Conclusões	134
A	Gráficos Gerados pelo Sistema SMEC	137
B	Resultados Complementares das Análises Experimentais	143
C	Aprendizado de Máquina na Web	146

Lista de Figuras

2.1	Sistema de Aprendizado	4
2.2	Sistema de Classificação	5
2.3	Representação da Classificação	6
2.4	Ciclo de Evolução	12
3.1	ϵ Círculo Delineando o Conjunto de Hipóteses PAC	20
3.2	Número de Exemplos, m , em Função de n e δ para $\epsilon = 0.01$ e 0.001	25
3.3	Conjuntos de Treinamento e Teste	26
3.4	Número de Casos de Teste e Qualidade da Predição [Weiss 91]	27
4.1	Exemplo de Árvore de Decisão Binária	39
4.2	Atributo Cabeça como Nó Raiz da Árvore de Decisão	41
4.3	Árvore de Decisão Induzida para o Exemplo dos Robôs	41
4.4	Divisão dos Exemplos do Nó Raiz Através do Atributo Cabeça	44
4.5	Divisão dos Exemplos do Nó Raiz Através do Atributo Corpo	45
4.6	Divisão dos Exemplos do Nó Raiz Através do Atributo Sorri?	45
4.7	O Atributo Cabeça para a Divisão do Nó Direiro	46
4.8	Exemplo de Interpretação Geométrica de uma Regra	48
4.9	Árvore de Decisão Correspondente à Expressão Lógica $(A \wedge B) \vee (C \wedge D)$	49
4.10	Algoritmo Separar-e-Conquistar	51

4.11	Pós-poda em um Conjunto de Regras de Decisão [Fürnkranz 96]	56
4.12	O Atributo Composto Mesma-Forma como Raiz da Árvore de Decisão .	58
5.1	Completeza e Consistência de uma Hipótese [Lavrač 94]	72
5.2	Um Exemplo de Reticulado.	77
5.3	Algoritmo para Calcular a lgg de Termos	77
5.4	Algoritmo para Calcular lgg de Literais	78
5.5	Algoritmo para Calcular lgg de Cláusulas	79
5.6	Árvore Genealógica	83
6.1	Uma Visão Geral do Sistema AMPSAM	92
6.2	Um exemplo de Arquivo de Declaração de Atributos na Sintaxe Padrão	96
6.3	Um Exemplo de Arquivo de Exemplos	97
6.4	Caracteres Separadores para Arquivos de Declaração de Atributos . . .	99
6.5	Arquivo de Declaração de Atributos em Nova Sintaxe	100
6.6	Caracteres Separadores para Arquivos de Exemplos	101
6.7	Arquivo de Exemplos em Nova Sintaxe	101
6.8	Conversão do Arquivo de Declaração de Atributos entre a Sintaxe Padrão (<i>a</i>) e a Sintaxe CN2 (<i>f</i>)	102
7.1	Proporção entre as Classes X e Y para os Conjuntos de Exemplos	110
7.2	Taxas de Erro Aparente (—) e do Conjunto de Teste (- -) para Árvores de Decisão de Diversos Tamanhos (Criado a Partir de Informações em [Weiss 96])	112
7.3	Taxas de Erro Médias para o Conjunto de Exemplos Hepatitis	120
7.4	Taxas de Erro Médias para o Conjunto de Exemplos Breast Cancer . . .	121
7.5	Histograma de Taxas de Erro para Hepatitis (Criado a Partir de In- formações em [Holte 93])	123
7.6	Histograma de Taxas de Erro para Breast Cancer (Criado a Partir de Informações em [Holte 93])	124

7.7	Taxas de Erro Médias para o Conjunto de Exemplos Pima	125
7.8	Taxas de Erro Médias para o Conjunto de Exemplos Bupa	126
7.9	Taxas de Erro Médias para o Conjunto de Exemplos Hungarian	127
7.10	Taxas de Erro Médias para o Conjunto de Exemplos Credit	128
7.11	Médias Aritméticas das Taxas de Erro Aparente e das Taxas de Erro do 10-CV para o CN2	129
A.1	Média Aritmética (o) e Média Geométrica (+) para cada Iteração e para todo o Conjunto de Exemplos	137
A.2	Média Aritmética e Mediana para todas as Iterações e Média Aritmética (o) para cada Iteração	138
A.3	Taxa de Erro para a Classe X – Média Aritmética e Mediana para todas as Iterações e Média (o) para cada Iteração	139
A.4	Taxa de Erro para a Classe Y – Média Aritmética e Mediana para todas as Iterações e Média (o) para cada Iteração	140
A.5	Matriz de Confusão para uma Iteração	140
A.6	Matriz de Confusão para todas as Iterações	141
A.7	Taxa de Erro Porcentual para cada Iteração em todo o Conjunto de Exemplos e para cada Classe	141
A.8	Outra Perspectiva da Figura A.7	142

Lista de Tabelas

3.1	Exemplo de Matriz de Confusão para Três Classes	15
3.2	Tipos de Erros para Classificação com Duas Classes	15
3.3	Medidas de Performance de Classificação mais Comuns	16
3.4	Exemplo de Matriz de Confusão	17
3.5	Exemplo de Matriz de Custo	17
3.6	Funções Booleanas Diferentes com $n = 3$ atributos	23
3.7	Número de Exemplos para Aprendizado PAC no Espaço de Todas as Funções Booleanas	24
3.8	Comparação entre Holdout e Random Subsampling	30
3.9	Comparação entre Métodos de Cross-validation	31
3.10	Características dos Estimadores Bootstrap	33
4.1	Exemplos de Robôs Amigos e Inimigos	40
4.2	Construção do Atributo Composto Mesma-Forma para o Exemplo dos Robôs Amigos de Inimigos	57
5.1	Exemplos de lgg de Termos	78
5.2	Exemplos de lgg de Literais	78
7.1	Características dos Conjuntos de Exemplos	110
7.2	Taxas de Erro Aparente para os Algoritmos CN2 e FOIL ([Monard 95]) e NewId (AMPSAM).	113

7.3	Matriz de Confusão e Taxas de Erro de Classificação	117
7.4	Taxas de Erro Médias para o Conjunto de Exemplos Hepatitis	119
7.5	Taxas de Erro Médias para o Conjunto de Exemplos Breast Cancer	121
7.6	Frequências de Desempenhos por Intervalo (Criado a Partir de Informações em [Holte 93]	122
7.7	Médias Aritméticas das Taxas de Erro para os Conjuntos de Exemplos Pima, Bupa, Hungarian e Credit	123
7.8	Requisitos Presentes em Publicações de Avaliação Experimental em Re- des Neurais [Flexer 96]	128
B.1	Taxas de Erro para o Conjunto de Exemplos Pima	143
B.2	Taxas de Erro para o Conjunto de Exemplos Bupa	144
B.3	Taxas de Erro para Conjunto de Exemplos Hungarian	144
B.4	Taxas de Erro para Conjunto de Exemplos Credit	145

Capítulo 1

Introdução

Em 1956, Bruner, Goodnow e Austin publicaram o livro *A Study of Thinking*, o qual se tornou um marco na psicologia e teria depois um grande impacto sobre Aprendizado de Máquina [Sammut 94]. Os experimentos realizados por Bruner, Goodnow e Austin foram direcionados para a compreensão da capacidade humana de categorizar e como as categorias são aprendidas.

Nós começamos com que parece ser um paradoxo. O mundo de experiências de qualquer homem normal é composto por uma imensa capacidade de discriminalizar diferentes objetos, eventos, pessoas, impressões ... Mas se nós usássemos toda a nossa capacidade para registrar as diferenças entre os objetos e para responder a cada evento encontrado como único, nós logo poderíamos ser sobrecarregados pela complexidade do nosso ambiente ... A solução do que parece um paradoxo ... é encontrada pela capacidade do homem de categorizar. Categorizar é tornar coisas similares equivalentes, agrupar objetos, eventos e pessoas à nossa volta em classes ... O processo de categorização envolve ... um ato de invenção ... Se nós aprendermos a classe “casa” como um conceito, novos exemplos poderão ser prontamente reconhecidos. A categoria se torna uma ferramenta para uso futuro. O aprendizado e a utilização de categorias representam uma das formas mais gerais e elementares de cognição pela qual o homem se ajusta ao seu ambiente.

Bruner, Goodnow e Austin assumiram que objetos e eventos poderiam ser descritos por um conjunto de atributos e estavam preocupados com como inferências poderiam ser delineadas dos atributos até as classes.

... quando alguém aprende a categorizar um subconjunto de exemplos de uma forma, esta pessoa está mais que simplesmente aprendendo a reco-

nhecer instâncias encontradas. Ele está também aprendendo uma regra que pode ser aplicada a novas instâncias. O conceito ou categoria é basicamente esta “regra de agrupamento” e com tais regras uma pessoa contrói, forma e obtém conceitos.

A noção de uma regra como uma representação abstrata de um conceito na mente humana se tornou questionável por psicologistas e ainda não há uma boa teoria para explicar como são armazenados os conceitos. A mesma questão sobre a natureza da representação também está presente em aprendizado de máquina.

Na realidade, existem muitas formas de inferência que podem ser aplicadas a um conhecimento, na tentativa de obter conceitos sobre esta informação. Por exemplo, um estudante pode tentar aprender o conceito de borboleta analisando descrições de borboletas, generalizando exemplos de borboletas específicas, construindo o conceito em um processo de observação e análise de diferentes tipos de insetos, ou ainda por outras formas. O tipo de inferência realizada pelo estudante sobre as informações fornecidas define a estratégia de aprendizado do conceito e pode constituir um bom critério para classificar os processos de aprendizado [Michalski 87].

Muitas estratégias básicas de aprendizado podem ser identificadas nas pesquisas de Aprendizado de Máquina. Este trabalho tem como principal foco o aprendizado indutivo por exemplos, no qual o sistema de aprendizado busca adquirir conceitos através da generalização de exemplos e, opcionalmente contra-exemplos, providos por um professor ou pelo ambiente.

A generalização de exemplos e contra-exemplos é um processo complexo cuja precisão dos resultados pode apresentar variações em diferentes aplicações, segundo diversos fatores. Uma vez que não existe uma análise teórica que possa determinar se um algoritmo de aprendizado irá desempenhar bem em uma determinada aplicação, estudos experimentais são necessários.

Métodos estatísticos são essenciais para uma análise experimental não tendenciosa. Entre os métodos estatísticos mais conhecidos estão os métodos de resampling, os quais são capazes de estimar a taxa de erro verdadeira de forma confiável mesmo para pequenas amostras de casos resolvidos. Os métodos de resampling são assim chamados pois consistem de repetidos experimentos de amostragem, treinamento e teste.

Pesquisar e implementar ferramentas, baseadas em métodos estatísticos, capazes de medir a performance de sistemas indutivos, de forma não tendenciosa, é um dos objetivos deste trabalho.

Este trabalho está organizado da seguinte forma:

O Capítulo 2 apresenta os principais paradigmas de aprendizado de máquina. São brevemente apresentados os paradigmas simbólico, conexionista, instance-based, estatístico e genético.

O Capítulo 3 apresenta a taxa de erro como um índice para medir a performance de sistemas de aprendizado. São discutidos alguns métodos estatísticos capazes de estimar a taxa de erro verdadeira com confiabilidade. São apresentados também alguns resultados teóricos provenientes do modelo Probably Approximately Correct — PAC.

O Capítulo 4 mostra com mais detalhes o paradigma de aprendizado de máquina simbólico proposicional. É dada ênfase ao aprendizado que utiliza árvores de decisão e regras de decisão como linguagens de descrição de hipóteses ou conceitos, uma vez que muitos dos principais algoritmos de aprendizado de máquina utilizam essas linguagens para descrever os conceitos aprendidos.

O Capítulo 5 apresenta os principais conceitos da Programação Lógica Indutiva. Esta abordagem de aprendizado simbólico busca superar algumas das principais limitações dos sistemas de aprendizado proposicionais através da utilização da lógica de primeira ordem como linguagem de descrição de exemplos, hipóteses e conhecimento de fundo.

O Capítulo 6 descreve o projeto do ambiente AMPSAM— Ambiente Computacional para Medir a Performance de Sistemas de Aprendizado de Máquina. Este ambiente busca tornar o processo de medir a performance de sistemas de aprendizado de máquina mais ágil e confiável.

O Capítulo 7 apresenta alguns resultados obtidos com experimentos realizados através do ambiente AMPSAM, envolvendo alguns dos mais conhecidos algoritmos de aprendizado simbólico e utilizando conjuntos de exemplos reais.

O Capítulo 8 apresenta as conclusões deste trabalho, evidenciando as linhas de pesquisa a serem continuadas.

Capítulo 2

Sistemas de Aprendizado

2.1 Considerações Iniciais

Sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas contidas em casos resolvidos com sucesso [Weiss 91]. Mais especificamente, este trabalho se concentra em uma das tarefas de aprendizado mais básica e proeminente: a *classificação* ou *predição*. Como ilustrado na Figura 2.1, o objetivo fundamental de um sistema de aprendizado está em extrair conhecimento na forma de, por exemplo, regras de decisão, de um conjunto de dados tal que essas regras possam ser aplicadas a novos dados. Um sistema de aprendizado típico é projetado para trabalhar sob algum paradigma de aprendizado, tal como o paradigma simbólico, estatístico, conexionista, etc. “Aprender” consiste em escolher ou adaptar parâmetros da representação do modelo, no paradigma escolhido, de forma que o modelo classifique corretamente os exemplos conhecidos bem como novos exemplos.

Figura 2.1: Sistema de Aprendizado

De certa forma, classificação pode ser resumida em encontrar uma função que mapeie

objetos com características comuns, na mesma classe. Tal problema é o mesmo a que Bruner, Goodnow e Austin [Bruner 56] se referiram e chamaram de categorização. A classificação pode possuir dois significados diferentes:

1. Dado um conjunto de observações, tentar estabelecer a existência de classes ou *clusters* nos dados;
2. Dado um conjunto de observações no qual as classes são conhecidas, o objetivo está em encontrar regras capazes de classificar novas observações entre as classes já existentes.

O primeiro tipo é conhecido como *Aprendizado Não Supervisionado* ou *Clustering*, o segundo é conhecido como *Aprendizado Supervisionado*. Este trabalho se concentrará em aprendizado supervisionado.

2.2 O Modelo de Classificação

Sob a óptica do problema de classificação, um sistema de aprendizado pode ser visto como um sistema de alto nível que ajuda a construir um sistema de tomada de decisões chamado de *classificador*. A forma mais simples de representar um classificador seria como uma caixa preta que produz uma decisão para cada padrão de dados que for apresentado a ela. A Figura 2.2 ilustra a estrutura simples de um sistema de classificação o qual aceita um padrão de dados como entrada e produz uma decisão como saída.

Figura 2.2: Sistema de Classificação

Um sistema de aprendizado tem disponível um conjunto finito de exemplos de casos resolvidos. Os dados de cada caso consistem em um padrão de observações e a classificação correta correspondente. O objetivo do sistema de aprendizado é moldar a estrutura do classificador para um problema específico, encontrando uma forma genérica de relatar qualquer padrão particular de observações em uma das classes especificadas.

A representação básica do problema de classificação é simples. Cada exemplo de um problema resolvido consiste de observações e a correta classe correspondente. Geralmente, uma simples classe será resultado de um padrão de classes observado. Porém, podem haver problemas nos quais múltiplas classes estão associadas a um padrão de observações, mas freqüentemente tais problemas podem ser decompostos em vários sub-problemas envolvendo classes individuais.

Desta forma, o conjunto de exemplos contém os dados que o sistema de aprendizado irá usar para encontrar as regras de decisão generalizadas para o classificador. A Figura 2.3 ilustra os elementos representacionais de uma coleção de casos resolvidos, e a relação entre o sistema de aprendizado e o classificador. Cada observação, do conjunto de observações, potencialmente relevante à solução do problema é chamada de *atributo*. Atributos também são chamados de *features*, variáveis, testes e medidas.

Figura 2.3: Representação da Classificação

Desta forma, o problema de classificação pode ser resumido da seguinte forma:

dados exemplos compostos por n atributos, $x = x_1, \dots, x_n$, e a classe associada a cada exemplo, y , a tarefa é encontrar uma função de mapeamento, f , tal que $y = f(x)$.

A diferença entre *regressão* e *classificação* é que em problemas de regressão a variável de saída y assume valores contínuos, enquanto que em problemas de classificação, y é estritamente categórica. Desta perspectiva, classificação pode ser pensada como uma sub-categoria de regressão [Weiss 95].

Como casos corretamente resolvidos serão utilizados na construção de um classificador específico, o padrão de valores de atributos para cada caso é associado com uma classificação correta. Desta forma, aprender neste tipo de sistema pode ser visto como um processo de generalização das associações observadas empiricamente, mas sob restrições impostas pelo paradigma de aprendizado escolhido.

2.3 Paradigmas de Aprendizado por Exemplos

Dentro da área de aprendizado de máquina já foram propostos vários paradigmas diferentes. Um requisito básico para todos os paradigmas de classificação, que são discutidos a seguir, é que todo o conhecimento a ser aprendido esteja na forma de exemplos e que cada exemplo esteja relacionado a uma classe corretamente classificada. Os paradigmas de aprendizado mais conhecidos são brevemente discutidos a seguir.

2.3.1 Paradigma Simbólico

Os sistemas de aprendizado simbólico buscam aprender construindo representações simbólicas de um conceito através da análise de exemplos e contra-exemplos deste conceito. As representações simbólicas estão tipicamente na forma de alguma expressão lógica, árvore de decisão, regras de produção ou rede semântica.

Atualmente, entre as representações simbólicas mais estudadas estão as árvores e regras de decisão. É atribuído a Morgan e Messenger [Morgan 73], o desenvolvimento original do programa para a indução de árvores de decisão. Métodos de indução de árvores de decisão a partir de dados empíricos, conhecido como *particionamento recursivo*, foi estudado por pesquisadores da área de Inteligência Artificial e Estatística. Os sistemas ID3 [Quinlan 86] e C4 [Quinlan 87b] para indução de árvores de decisão tiveram uma importante contribuição sobre a pesquisa em Inteligência Artificial. É interessante observar que sistemas de classificação e árvores de regressão [Breiman 84] foram desenvolvidos por estatísticos durante praticamente o mesmo período que o ID3, no final dos anos 70.

Os trabalhos com indução de regras de decisão surgiram com a simples tradução das árvores de decisão para regras, com a poda realizada sobre as regras; tal abordagem surgiu em [Quinlan 87a, Quinlan 87b]. Posteriormente, foram criados métodos que induziam regras diretamente a partir dos dados, um exemplo deste trabalho pode ser encontrado em [Michalski 86].

Aprendizado simbólico será discutido em maiores detalhes nos Capítulos 4 e 5.

2.3.2 Paradigma Estatístico

Pesquisadores em estatística têm criado diversos métodos de classificação, muitos deles semelhantes aos métodos empregados em aprendizado de máquina. Por exemplo, o método CART [Breiman 84], um sistema muito conhecido para montar árvores de decisão, foi desenvolvido por estatísticos. Como regra geral, técnicas estatísticas tendem a focar tarefas em que todos os atributos têm valores contínuos ou ordinais. Muitos deles também são paramétricos, assumindo alguma forma de modelo, e então encontrando valores apropriados para os parâmetros do modelo a partir de dados. Por exemplo, um classificador linear assume que classes podem ser expressas como combinação linear dos valores dos atributos, e então procura uma combinação linear particular que forneça a melhor aproximação sobre o conjunto de dados. Os classificadores estatísticos freqüentemente assumem que valores de atributos estão normalmente distribuídos, e então usam os dados fornecidos para determinar média, variância e co-variância da distribuição.

Alguns autores têm considerado redes neurais como métodos estatísticos paramétricos, uma vez que treinar uma rede neural geralmente significa encontrar valores apropriados para pesos e *bias* pré-determinados.

2.3.3 Paradigma Instance-based

Uma forma de classificar um caso é lembrar de um caso similar cuja classe é conhecida e assumir que o novo caso terá a mesma classe. Esta filosofia exemplifica os sistemas instance-based, que classificam casos nunca vistos através de casos similares conhecidos [Quinlan 88].

As características principais dos sistemas instance-based são

- Quais casos de treinamento devem ser lembrados? Se todos os casos forem memorizados, o classificador pode se tornar lento e difícil de manusear. O ideal é reter casos prototípicos que juntos resumam toda a informação importante; esta abordagem pode ser observada em livros médicos e legais. Em [Aha 91] estão descritas algumas estratégias para decidir quando um novo caso deve ser memorizado;
- Como deve ser medida a similaridade entre os casos? Se todos os atributos forem contínuos, pode-se calcular a distância entre dois casos como a raiz quadrada da soma dos quadrados das diferenças dos atributos. Quando alguns atributos não são ordinais, essa interpretação de distância se torna mais problemática. Além do mais, se existem muitos atributos irrelevantes, dois casos similares podem aparentar serem muito diferentes pois eles podem possuir valores diferentes em atributos

sem importância. Em [Stanfill 86] foi desenvolvido um método sensível ao contexto para alterar a escala dos atributos de forma que as medidas de distância fiquem mais robustas;

- Como um novo caso deve ser relacionado com casos armazenados? Existem duas alternativas que são usar um simples caso armazenado o qual é o mais próximo do novo caso, ou usar vários casos levando-se em consideração os diferentes graus de similaridade entre cada caso.

2.3.4 Paradigma Conexionista

Redes neurais são construções matemáticas relativamente simples que foram inspiradas no modelo biológico do sistema nervoso. Sua representação envolve unidades altamente interconectadas, no qual o nome *conexionismo* é utilizado para descrever a área de estudo.

A metáfora biológica com as conexões neurais do sistema nervoso tem interessado muitos pesquisadores, e tem fornecido muitas discussões sobre os méritos e as limitações desta abordagem de aprendizado. Em particular, as analogias com a biologia têm levado muitos pesquisadores a acreditar que as redes neurais possuem um grande potencial na resolução de problemas que requerem intenso processamento sensorial humano, tal como visão e reconhecimento de voz.

As pesquisas em redes neurais foram iniciadas com o trabalho pioneiro de McCulloch e Pitts em [McCulloch 43]. McCulloch era um psiquiatra e pesquisou por 20 anos uma forma de representar um evento no sistema nervoso. Pitts era um jovem pesquisador e começou a trabalhar com McCulloch em 1942. Praticamente 15 anos após a publicação de McCulloch e Pitts, Rosenblatt em [Rosenblatt 58] apresentou o Perceptron, cuja grande contribuição foi a prova do teorema de convergência. Mas no livro *Perceptrons*, [Minsky 69], Minsky e Papert demonstraram a existência de limites fundamentais nos perceptrons de uma camada. A pesquisa na área ficou praticamente estática até que Hopfield em [Hopfield 82] utilizou a idéia de uma função de energia para formular uma nova forma de compreender os cálculos realizados em redes recorrentes com conexões sinápticas simétricas.

Talvez mais que qualquer outra publicação, o artigo de Hopfield em 1982 e o livro de Rumelhart e McLelland [Rumelhart 86], foram as publicações que mais influenciaram para o resurgimento do interesse sobre redes neurais na década de 80. Redes neurais tiveram um longo caminho desde McCulloch e Pitts, e continuarão a crescer em teoria, projetos e aplicações [Haykin 94].

2.3.5 Paradigma Genético

Este formalismo de classificação é derivado do modelo evolucionário de aprendizado [Holland 86]. Um classificador genético consiste de uma população de elementos de classificação que competem para fazer a predição. Elementos que possuem uma performance fraca são descartados, enquanto os elementos mais fortes proliferam, produzindo variações de si mesmos. Este paradigma possui uma analogia direta com a teoria de Darwin, na qual sobrevivem os mais bem adaptados ao ambiente. Segundo [Freitas 92]

Um algoritmo genético é um procedimento iterativo que mantém uma população de indivíduos, cada um dos quais é um candidato à solução de algum problema específico. A cada iteração (denominada geração), os indivíduos da população atual são avaliados quanto à sua aptidão para a solução do problema. Com base nessas avaliações aplicam-se alguns operadores genéticos aos indivíduos, formando-se uma nova população, que substituirá a população atual. Isto é feito de modo que, quanto maior a aptidão de um indivíduo da população atual, maior a sua influência na formação dos indivíduos da nova geração. Assim, com o passar do tempo, a seleção natural tende a fazer com que a população seja formada por indivíduos cada vez melhores (soluções cada vez mais próximas da solução ótima para o problema). Como critério de parada do algoritmo genético, geralmente define-se um limite no número de gerações. Dentre os indivíduos da última geração, aqueles mais aptos representam a melhor solução encontrada pelo algoritmo. Pode-se também especificar que o algoritmo encerrará quando for gerado algum indivíduo que satisfaça alguma condição mínima de aptidão.

Alguns operadores genéticos básicos que aplicados à população geram novos indivíduos são discutidos a seguir. Esses operadores são denominados

1. Reprodução;
2. Cruzamento;
3. Mutação;
4. Inversão.

1. Reprodução. Este operador indica a quantidade de cópias que um indivíduo produz, proporcional à sua aptidão. Estas cópias serão submetidas à ação de outros operadores genéticos na geração de novos indivíduos.
2. Cruzamento. Este operador realiza a troca de material genético entre indivíduos. Por exemplo, considerando os seguintes indivíduos representados por uma cadeia de símbolos de comprimento 6

Indivíduo 1: $\boxed{x1 \ x2 \ x3 \ x4 \ x5 \ x6}$

Indivíduo 2: $\boxed{y1 \ y2 \ y3 \ y4 \ y5 \ y6}$

aplicando o operador de cruzamento na sexta posição, são gerados os seguintes novos indivíduos

Indivíduo 3: $\boxed{x1 \ x2 \ x3 \ x4 \ x5 \ \mathbf{y6}}$

Indivíduo 4: $\boxed{y1 \ y2 \ y3 \ y4 \ y5 \ \mathbf{x6}}$

3. **Mutação.** Como em Biologia, a mutação, geralmente, é importante para uma espécie sobreviver. Aplicada aos algoritmos genéticos, este operador modifica o valor de uma posição aleatória da cadeia de símbolos que representa o indivíduo. Por exemplo, dado um indivíduo representado por uma cadeia de símbolos binários

Indivíduo 1: $\boxed{0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0}$

aplicando o operador de mutação na quarta posição da cadeia é gerado o seguinte indivíduo

Indivíduo 2: $\boxed{0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0}$

A vantagem deste operador é que a probabilidade de pesquisar pela solução de um problema em qualquer região do espaço nunca será zero, virtude que os outros operadores não possuem.

4. **Inversão.** Este operador realiza a troca de símbolos na cadeia que representa o indivíduo. Símbolos em posições mais próximas têm maior probabilidade de serem rearranjados dentro da cadeia.

Por exemplo, dado o Indivíduo 1, o operador de inversão (posições 3 e 5) gera o Indivíduo 2

Indivíduo 1: $\boxed{x1 \ x2 \ \mathbf{x3} \ x4 \ \mathbf{x5} \ x6}$

Indivíduo 2: $\boxed{x1 \ x2 \ \mathbf{x5} \ x4 \ \mathbf{x3} \ x6}$

A Figura 2.4 mostra o processo cíclico no qual os operadores genéticos são aplicados.

Segundo [De Jong 88] os algoritmos genéticos são uma técnica utilizada para solucionar problemas de otimização e de aprendizado de máquina, utilizando simulações e regras probabilísticas (e não determinísticas). O sistema trabalha na forma de tentativa e erro com alguma forma de retroalimentação do erro, onde conseqüentemente “aprende” soluções melhores para os problemas. O algoritmo nem sempre encontra uma solução correta, mas procura adquirir novos conhecimentos e chegar o mais próximo possível da solução.

Figura 2.4: Ciclo de Evolução

2.4 Considerações Finais

Os sistemas de aprendizado podem ser vistos como programas capazes de criar classificadores a partir de conjuntos de exemplos. Na realidade, o principal objetivo de um sistema de aprendizado está em extrair conceitos expressos em alguma linguagem, por exemplo regras de decisão, capazes de serem aplicadas a novos casos. Os classificadores são sistemas que simplesmente utilizam tais regras para classificar casos nunca vistos.

É importante ressaltar que existem diversos paradigmas diferentes para extrair o conhecimento presente nos exemplos e transformá-los em regras. Por exemplo, existem paradigmas que procuram extrair o conhecimento através de representações simbólicas e outros utilizam cálculos estatísticos ou numéricos. Este capítulo discutiu brevemente os paradigmas mais conhecidos de aprendizado de máquina. No próximo capítulo será discutido com mais detalhes o paradigma simbólico de aprendizado de máquina.

Capítulo 3

Técnicas para Medir a Performance de Sistemas de Aprendizado

3.1 Considerações Iniciais

O principal objetivo a ser atingido por um sistema de aprendizado está em classificar com sucesso casos nunca vistos. Este objetivo se torna mais problemático quando estão disponíveis apenas pequenos conjuntos de exemplos para treinamento. Nestas situações não se sabe ao certo se tal número reduzido de exemplos é ou não suficiente para extrair conceitos genéricos. Este tipo de indagação, entre outros, motivou a pesquisa de métodos capazes de apresentar índices que estimam a performance de classificadores para exemplos não vistos.

Existem várias formas de estimar a performance de um sistema de aprendizado. Algumas formas são muito melhores que outras. Em termos estatísticos, certos métodos são considerados *tendenciosos*. Alguns tendem a apresentar estimativas de performance muito otimistas e outros estimativas muito pessimistas. Encontrar algum índice capaz de estimar verdadeiramente a performance de um sistema de aprendizado é a base para uma comparação adequada dos diversos sistemas de aprendizado.

3.2 A Taxa de Erro

Quando é fornecido um padrão de entrada a um classificador, este responde com a classe a que supostamente este padrão pertence. Algumas vezes o classificador responde corretamente, outras vezes não. Seguramente, a taxa de erro é a forma mais utilizada

para medir a performance de um classificador. Pode-se calcular a taxa de erro através da equação

$$taxa\ de\ erro = \frac{número\ de\ erros}{número\ de\ casos} \quad (3.1)$$

É conhecida por *taxa de erro aparente* de um classificador a taxa de erro calculada utilizando somente os exemplos de treinamento, ou seja, inicialmente o sistema é treinado com um conjunto de exemplos e posteriormente é verificado quantos erros o classificador cometeu classificando os mesmos exemplos com que foi treinado. No outro extremo, encontra-se a *taxa de erro verdadeira* que pode ser entendida como a taxa de erro obtida sobre um número muito grande de novos casos, selecionados independentemente dos casos usados para treinar o classificador. A taxa de erro verdadeira difere da taxa de erro aparente, pois a taxa de erro verdadeira exige um grande número de exemplos independentes dos exemplos de treinamento para testar o classificador.

A taxa de erro verdadeira fornece uma excelente medida da performance de um classificador. Entretanto, em casos reais, conjuntos com grande número de exemplos são raros, o que inviabiliza o seu cálculo na maioria das vezes. Resta, então, saber se é possível aproximar a taxa de erro verdadeira através de taxas de erro empíricas calculadas com pequenos conjuntos de exemplos.

Uma tentativa inicial seria tentar estimar a taxa de erro verdadeira através da taxa de erro aparente. Entretanto, é bastante comum que os classificadores possuam uma taxa de erro aparente muito inferior a taxa de erro verdadeira, pois pode ocorrer o processo de *overfitting* (overfitting é discutido na seção 3.6). Desta forma, se a taxa de erro aparente fosse utilizada, a estimativa seria pouco confiável e extremamente otimista se comparada com a taxa de erro verdadeira.

Através de técnicas estatísticas que apresentam de diferentes formas os exemplos ao classificador é possível obter aproximações da taxas de erro verdadeira bem mais confiáveis que a taxa de erro aparente. Tais técnicas são objetos de discussão da seção 3.5.

3.3 Tipos de Erros

Se todos os erros são de igual importância, a performance de um classificador pode ser calculada através da equação 3.1. Entretanto, distinções entre os tipos de erros são importantes para muitas aplicações. Por exemplo, o erro cometido no diagnóstico de uma pessoa como sendo sadia, quando na verdade esta pessoa está correndo risco de vida (conhecido como decisão *falsa negativa*) é muito mais grave que o tipo oposto de erro, ou seja, diagnosticar alguém doente quando, na verdade, esta pessoa está sadia (conhecido como decisão *falsa positiva*). Frequentemente, com testes futuros e o passar

do tempo o falso diagnóstico da pessoa sadia será corrigido e nenhum dano sério será causado à sua saúde. Entretanto, a pessoa doente que foi encaminhada para casa poderá adoecer, e no pior dos casos até mesmo morrer, o que pode tornar o erro muito sério.

Se é necessário fazer distinções entre os tipos de erros, então a *matriz de confusão* pode ser usada para separar os diferentes erros. A Tabela 3.1 mostra um exemplo deste tipo de matriz para três classes. O número de predições corretas para cada classe está representada ao longo da diagonal da matriz. Todos os outros números são erros de diferentes tipos. Por exemplo, na Tabela 3.1 a classe ♠ foi corretamente classificada 53 vezes, mas foi erroneamente classificada como classe ♦ 5 vezes.

	<i>Classes Preditas</i>		
<i>Classes Verdadeiras</i>	♣	♠	♦
♣	36	0	1
♠	0	53	5
♦	1	2	72

Tabela 3.1: Exemplo de Matriz de Confusão para Três Classes

Problemas com apenas duas classes são mais comuns, pois geralmente se está interessado em prever a ocorrência ou a não ocorrência de um simples evento ou hipótese. A Tabela 3.2 mostra os tipos de erros que podem ocorrer através de uma matriz de confusão para duas classes.

	<i>Predição Positiva</i>	<i>Predição Negativa</i>
<i>Classe Positiva</i>	Verdadeiro Positivo (<i>a</i>)	Falso Negativo (<i>b</i>)
<i>Classe Negativa</i>	Falso Positivo (<i>c</i>)	Verdadeiro Negativo (<i>d</i>)

Tabela 3.2: Tipos de Erros para Classificação com Duas Classes

É comum encontrar na literatura algumas medidas de precisão de performance para classificadores. A Tabela 3.3 mostra algumas medidas muito comuns; essas medidas se referem ao número de acertos do classificador, ao contrário da taxa de erro que mede o número de erros. Algumas medidas semelhantes a estas são utilizadas no Capítulo 7, entretanto essas medidas foram redefinidas para representar o número de erros ao invés do número de acertos do classificador. As variáveis da Tabela 3.2 representam o número de casos

- *a*: Classificados como positivos e pertencentes à classe positiva;
- *b*: Classificados como positivos mas pertencentes à classe negativa;
- *c*: Classificados como negativos mas pertencentes à classe positiva;
- *d*: Classificados como negativos e pertencentes à classe negativa;

- $a + b$: Pertencentes à classe positiva;
- $c + d$: Pertencentes à classe negativa;
- $a + c$: Classificados como positivos;
- $b + d$: Classificados como negativos.

<i>Sensibilidade</i>	$\frac{a}{a+b}$
<i>Especificidade</i>	$\frac{d}{c+d}$
<i>Valor Preditivo +</i>	$\frac{a}{a+c}$
<i>Valor Preditivo -</i>	$\frac{d}{b+d}$
<i>Precisão</i>	$\frac{a+d}{a+b+c+d}$

Tabela 3.3: Medidas de Performance de Classificação mais Comuns

As medidas da Tabela 3.3 podem ser interpretadas da seguinte forma

Sensibilidade: O percentual de casos da classe positiva corretamente classificados;

Especificidade: O percentual de casos da classe negativa corretamente classificados;

Valor Preditivo +: O percentual dos casos reconhecidos pelo classificador como pertencentes à classe positiva que realmente pertencem a esta classe;

Valor Preditivo -: O percentual dos casos reconhecidos pelo classificador como pertencentes à classe negativa que realmente pertencem a esta classe;

Precisão: O percentual de casos de ambas as classes corretamente classificados.

Quando é necessário distinguir entre os diferentes tipos de erros, o *custo de classificação incorreta* é uma alternativa natural à taxa de erro. Neste caso, ao invés de projetar um classificador que minimize a taxa de erro, o objetivo está em minimizar os custos de classificação incorreta. O custo de classificação incorreta é simplesmente uma penalização pela ocorrência de um erro. Por exemplo, no caso com duas classes, um custo pode ser associado ao erro falso positivo, e outro custo pode ser associado ao erro falso negativo. Um custo médio de classificação incorreta pode ser obtido multiplicando cada custo pela respectiva taxa de erro. Mais formalmente, para qualquer matriz de confusão, seja E_{ij} , $i \neq j$, o número de erros na matriz de confusão e C_{ij} , $i \neq j$, o custo para cada tipo de classificação incorreta, o custo total de classificação incorreta C_{total} é dado pela equação 3.2, onde o custo de classificação correta (C_{ij} , para $i = j$) é 0.

$$C_{total} = \sum_{i=1}^n \sum_{j=1}^n E_{ij} C_{ij} \quad (3.2)$$

Para exemplificar a utilização do custo para medir a performance de um classificador, será utilizado o exemplo do classificador para prever a ocorrência ou não de uma dada doença. A matriz de confusão gerada por um determinado classificador é apresentada na Tabela 3.4.

	<i>Classes Preditas</i>	
<i>Classes Verdadeiras</i>	Positivo	Negativo
Positivo	181	15
Negativo	1	176

Tabela 3.4: Exemplo de Matriz de Confusão

A matriz de confusão mostrada na Tabela 3.4 apresenta a seguinte taxa de erro

$$Taxa\ de\ Erro = \frac{15 + 1}{181 + 1 + 15 + 176} = 0.04\ ou\ 4\%$$

Como foi dito anteriormente, a decisão falsa negativa é considerada mais grave e, desta forma foi elaborada uma matriz de custo, Tabela 3.5, a qual fornece um custo cinco vezes maior à decisão falsa negativa sobre a decisão falsa positiva.

	<i>Classes Preditas</i>	
<i>Classes Verdadeiras</i>	Positivo	Negativo
Positivo	0	5
Negativo	1	0

Tabela 3.5: Exemplo de Matriz de Custo

Desta forma, considerando os valores apresentados nas Tabelas 3.4 e 3.5, o custo total (C_{total}) do classificador é

$$C_{total} = (181 \times 0) + (1 \times 1) + (15 \times 5) + (176 \times 0) = 76$$

e o custo médio por decisão ($C_{médio}$) é de

$$C_{médio} = \frac{76}{373} = 0.20\ ou\ 20\%$$

Vale enfatizar que apesar do classificador apresentar uma taxa de erro relativamente baixa (aproximadamente 4%), este classificador pode ser considerado não apropriado para esta aplicação. Uma vez estipulada a verdadeira gravidade de cada erro, o classificador não apresentou um custo médio ($C_{médio}$) satisfatório.

Até então foi considerada somente a possibilidade de penalizar as classificações incorretas, mas é possível também estabelecer ganhos para classificações corretas. Na *análise*

de risco ou *análise de decisão*, ambos custos e benefícios são usados para avaliar a performance do classificador.

Neste tipo de análise, os custos são chamados de *riscos*. Se os custos de classificação incorreta são associados a números negativos, e os ganhos para decisões corretas são associados a números positivos, então a equação 3.2 pode ser reescrita em termos de riscos, da seguinte forma

$$R_{total} = \sum_{i=1}^n \sum_{j=1}^n E_{ij} R_{ij} \quad (3.3)$$

Entretanto, no caso de aplicações reais, é muito difícil estabelecer os valores correspondentes a custos e riscos para avaliar a performance de um classificador. Por isso a taxa de erro, objeto de estudo deste trabalho, é mais utilizada como padrão para medir a performance de classificadores. Contudo, deve ser ressaltado que as técnicas descritas no restante deste capítulo para estimar a taxa de erro verdadeira podem ser facilmente modificadas para estimar custos e riscos verdadeiros, sempre que esta informação esteja disponível.

Na próxima seção são apresentados alguns conceitos básicos provenientes de um modelo teórico conhecido como *Probably Approximately Correct*.

3.4 O Modelo Probably Approximately Correct (PAC)

Como já mencionado, a performance de um classificador é geralmente medida através de sua taxa de erro. Se está disponível um número ilimitado de casos para treinar, a taxa de erro aparente será a taxa de erro verdadeira. Infelizmente, conjuntos reais de exemplos são finitos e geralmente pequenos. Desta forma, uma indagação importante a ser respondida é

Quantos casos são necessários para que a taxa de erro aparente se torne efetivamente a taxa de erro verdadeira?

Algumas respostas para indagações como esta são fornecidas pela área denominada Teoria Computacional de Aprendizado¹. A idéia geral é a seguinte [Russell 95]: para qualquer hipótese que estiver *muito errada* com alta probabilidade, após processar um pequeno número de exemplos, este fato será observado pois essa hipótese fará previsões incorretas. Assim, para qualquer hipótese que é consistente com um número suficientemente grande de exemplos de treinamento, é pouco provável que ela esteja *muito*

¹Computational Learning Theory

errada, ou seja, essa hipótese deve ser Provavelmente Aproximadamente Correta². Um ponto importante a ser considerado é a relação entre o conjunto de treinamento e o conjunto de teste, pois interessa que a hipótese proposta seja aproximadamente correta também no conjunto de teste, e não somente no conjunto de treinamento. Nesta teoria é assumido que tanto o conjunto de treinamento quanto o conjunto de teste são retirados aleatoriamente do mesmo conjunto de exemplos usando a mesma distribuição de probabilidade. Esta condição, denominada de *estacionária*, é muito importante em aprendizado-PAC pois permite realizar uma conexão entre o passado — conjunto de exemplos de treinamento utilizado pelo algoritmo para aprender uma hipótese — e o futuro, isto é, classificar corretamente novos exemplos. Ou seja,

o modelo PAC define aprendizado em termos do poder preditivo da hipótese encontrada pelo algoritmo de aprendizado.

Como mostrado a seguir, é possível aplicar ao algoritmo esta medida de sucesso porque é considerado que os exemplos de treinamento são retirados independentemente de uma distribuição de probabilidades fixa D e seu poder preditivo é também medido em relação a essa mesma distribuição D .

Com as considerações acima, é possível responder à seguinte pergunta

Quantos exemplos de treinamento são necessários para que a hipótese h encontrada pelo algoritmo seja PAC?

Sejam

- $\mathbf{X} = \{X_1, X_2, X_3, \dots\}$ o conjunto de todos os possíveis exemplos;
- D a distribuição da qual os exemplos são retirados;
- $\mathbf{H} = \{h_1, h_2, h_3, \dots\}$ o conjunto das possíveis hipóteses;
- m o número de exemplos no conjunto de treinamento.

Observar que $\mathbf{X}$ e $\mathbf{H}$ podem também ser finitos. Assumindo que a *verdadeira* função f_v é um elemento de $\mathbf{H}$, isto é, $f_v \in \mathbf{H}$, o erro em um exemplo $X_i \in \mathbf{X}$ da hipótese h em relação a verdadeira função f_v dada a distribuição D sobre os exemplos pode ser definido como a probabilidade de h ser diferente de f_v no exemplo $X_i \in \mathbf{X}$

$$erro(h) = P(h(X_i) \neq f_v(X_i) | X_i \text{ retirado de } D)$$

A hipótese h é denominada *aproximadamente correta* se

$$erro(h) \leq \epsilon$$

²Probably Approximately Correct – PAC

ϵ uma constante pequena. A ideia geral é mostrar que após o algoritmo ter visto m exemplos, todas as hipóteses consistentes serão, com alta probabilidade, aproximadamente corretas. A Figura 3.1 ilustra esta idéia. As hipóteses aproximadamente corretas são aquelas no espaço de hipóteses $\mathbf{H}$ que se encontram *perto*, a uma distância ϵ , da verdadeira hipótese f_v , isto é, dentro de um círculo — denominado ϵ -círculo — cujo centro encontra-se a verdadeira função f_v .

Figura 3.1: ϵ Círculo Delineando o Conjunto de Hipóteses PAC

Assim, o conjunto $\mathbf{H}$ de hipóteses pode ser considerado como o conjunto de hipóteses aproximadamente corretas, que são aquelas dentro do ϵ -círculo, e as hipóteses restantes que serão denominadas $\mathbf{H}_{ruim}$. Para todo $h_r \in \mathbf{H}_{ruim}$, como $erro(h_r) > \epsilon$, pois h_r encontra-se fora do ϵ -círculo, então a probabilidade de h_r cobrir qualquer exemplo é

$$P(h_r \text{ cobrir um exemplo}) \leq (1 - \epsilon)$$

Então a probabilidade de *uma* hipótese ruim $h_r \in \mathbf{H}_{ruim}$ ser consistente com os primeiros m exemplos é

$$P(h_r \text{ cobrir } m \text{ exemplos}) \leq (1 - \epsilon)^m$$

Porém, para $\mathbf{H}_{ruim}$ conter hipóteses consistentes pelo menos uma das hipóteses em $\mathbf{H}_{ruim}$ deve ser consistente. A probabilidade de que isto aconteça está limitada pela soma das probabilidades individuais.

$$\begin{aligned} P(\mathbf{H}_{ruim} \text{ conter uma hipótese consistente}) &\leq |\mathbf{H}_{ruim}|(1 - \epsilon)^m \\ &\leq |\mathbf{H}|(1 - \epsilon)^m \end{aligned}$$

Interessa então analisar as condições para reduzir a probabilidade deste evento tal que seja menor do que um número pequeno δ , ou seja

$$|\mathbf{H}|(1 - \epsilon)^m \leq \delta$$

Aplicando $\ln$ obtêm-se

$$\begin{aligned}
\ln(|\mathbf{H}|) + m \ln(1 - \epsilon) &\leq \ln(\delta) \\
m &\leq \frac{\ln(\delta) - \ln(|\mathbf{H}|)}{\ln(1 - \epsilon)} \\
m &\geq \frac{1}{\ln(1 - \epsilon)} (-\ln(\delta) + \ln(|\mathbf{H}|)) \\
m &\geq \frac{1}{\ln(1 - \epsilon)} (\ln \frac{1}{\delta} + \ln(|\mathbf{H}|))
\end{aligned}$$

utilizando a inequação $(1 - \epsilon) \leq e^{-\epsilon}$ obtêm-se

$$m \geq \frac{1}{\epsilon} (\ln \frac{1}{\delta} + \ln |\mathbf{H}|) \quad (3.4)$$

Esta inequação mostra que se um algoritmo de aprendizado de máquina encontrar uma hipótese que é consistente com este número m de exemplos, então com probabilidade no mínimo de $1 - \delta$ essa hipótese tem erro máximo de ϵ

ϵ é denominado *parâmetro de erro* e δ *parâmetro de confiança*. Ambos controlam os dois tipos de fracassos aos quais é susceptível o algoritmo de aprendizado no modelo PAC. O número requerido de exemplos m em função de ϵ e δ é denominado *complexidade da amostra* do espaço de estados.

O critério mínimo não trivial a ser exigido do algoritmo é

$$\begin{aligned}
0 < \epsilon &< \frac{1}{2} \\
0 < \delta &< \frac{1}{2}
\end{aligned}$$

Considerando a inequação (3.4), aparentemente, a única incógnita a ser determinada é o tamanho do espaço de hipóteses $|\mathbf{H}|$. No caso de $|\mathbf{H}|$ ter cardinalidade finita e ignorando a eficiência computacional do algoritmo de aprendizado, ou, em outras palavras, assumindo que o algoritmo de aprendizado possui infinito poder computacional para processar a amostra m de exemplos, então, a inequação (3.4) afirma que se o algoritmo utilizar uma amostra finita $m = O(\frac{1}{\epsilon} \log \frac{|\mathbf{H}|}{\delta})$ de exemplos e encontrar qualquer hipótese h consistente com esses exemplos³ então h verifica o modelo PAC.

Há uma diferença fundamental entre um conceito (que pode ser um conjunto ou uma função booleana, por exemplo) e sua representação (que é a codificação simbólica desse conjunto ou função). Por exemplo, considerando a classe de conceitos definidos pelas

³Observar que essa hipótese h pode até ser encontrada por busca exaustiva.

atribuições que satisfazem uma função booleana f_b , um conceito dessa classe tanto pode ser representado pela própria fórmula f_b , pela sua tabela verdade, por uma árvore de decisão, bem como por outra fórmula booleana f'_b equivalente a f_b . Entretanto, ainda que todas elas sejam representações do mesmo conceito, o tamanho da representação de cada uma delas pode ser muito diferente. Por exemplo, a função booleana de paridade⁴

$$f(x_1, x_2, \dots, x_n) = x_1 \oplus x_2 \oplus \dots \oplus x_n$$

retorna 1 se e somente se um número par de entradas for 1. Esta função pode ser computada por um circuito digital – $h_{circuito}$ – constituído de portas $\neg$, $\vee$ e $\wedge$ cujo tamanho é limitado por um polinômio de grau n . Entretanto, a mesma função representada na Forma Normal Disjuntiva – h_{FND} – ou por uma árvore de decisão – h_{ad} – requer tamanho exponencial em n . Ou seja, as três hipóteses $h_{circuito}$, h_{FND} e h_{ad} são representações do mesmo conceito, no entanto

$$\begin{aligned} tamanho(h_{circuito}) &= O(n) \\ tamanho(h_{FND}) &= O(2^n) \\ tamanho(h_{ad}) &= O(2^n) \end{aligned}$$

Problema análogo acontece com a função booleana majoritária⁵ que retorna 1 se a maioria das entradas for 1.

Em cada um desses exemplos foi fixado um esquema de representação (circuito digital, FND, árvore de decisão, etc ...) para codificar conceitos, mas o tamanho da codificação (por exemplo número de portas de um circuito digital, número de literais de uma conjunção de funções booleanas, número de nós de uma árvore de decisão, etc ...) pode ser muito diferente para esquemas diferentes. Em princípio, o modelo de aprendizado PAC somente leva em conta os exemplos e sua classificação. Ele não possui informação referente ao esquema de representação que está sendo utilizado pelo algoritmo. Entretanto, como mostrado pelos exemplos citados, o esquema de representação utilizado pelo algoritmo é muito importante já que o tempo de execução do algoritmo também depende desse esquema.

Uma pergunta que surge é a seguinte

Existe algum esquema de representação que é eficiente para todas as classes de funções?

A resposta é negativa. Por exemplo, considerando o conjunto de todas as funções booleanas com n atributos

$$F_n = \{f_{n_1}, f_{n_2}, f_{n_3}, \dots, f_{n_k}, \dots\}$$

⁴onde $\oplus$ denota a operação de ou-exclusivo.

⁵Majority function

definidas pela sua tabela verdade, então a cardinalidade $|F_n|$ desse conjunto é o número de tabelas verdade diferentes. Cada tabela verdade tem 2^n filas já que cada entrada está definida por n atributos. Assim, pode-se considerar que a coluna $n + 1$ que especifica o valor dessa função – classe – está representada por um número de 2^n bits que define a função. Portanto, independentemente do esquema de representação utilizado, serão necessários, para algumas funções (na realidade para a maioria delas) 2^n bits para representá-las. Dessa forma, conclui-se que existem 2^{2^n} — um número exponencial —, de funções booleanas diferentes com n atributos, ou seja

$$|F_n| = 2^{2^n}$$

Por exemplo, com somente $n = 3$ atributos booleanos existem $2^8 = 256$ funções booleanas diferentes — Tabela 3.6.

X	x_1	x_2	x_3	f_{3_1}	f_{3_2}	f_{3_3}	$\dots$	$f_{3_{10}}$	$\dots$	$f_{3_{256}}$
X₁	0	0	0	0	0	0		0		1
X₂	0	0	1	0	0	0		1		1
X₃	0	1	0	0	0	0		0		1
X₄	0	1	1	0	0	0		0		1
X₅	1	0	0	0	0	0		0		1
X₆	1	0	1	0	0	0		0		1
X₇	1	1	0	0	0	1		0		1
X₈	1	1	1	0	1	0		1		1

Tabela 3.6: Funções Booleanas Diferentes com $n = 3$ atributos

Somente duplicando o número de valores booleanos para $n = 6$, o número de funções booleanas diferentes cresce para $2^{64} = 1.84 * 10^{19}$

Assim, considerando o caso geral do conjunto de todas as funções booleanas com n atributos, e com base na inequação (3.4), a complexidade da amostra m é

$$m \geq \frac{1}{\epsilon} \left(\ln \frac{1}{\delta} + \ln 2^{2^n} \right)$$

$$m \geq \frac{1}{\epsilon} \left(\ln \frac{1}{\delta} + 2^n \ln 2 \right)$$

portanto, para ϵ e δ fixos tem-se

$$m = O(2^n)$$

Como o número de possíveis exemplos **X** é também 2^n , isto significa que qualquer algoritmo de aprendizado para o espaço de hipóteses de todas as funções booleanas não fará melhor que uma tabela *lookup* se simplesmente retornar uma hipótese que é consistente com todos os exemplos conhecidos. A Tabela 3.7 mostra o número de exemplos necessários em função de alguns valores de n , ϵ e δ para o espaço de hipóteses de todas as funções booleanas.

n	ϵ	δ	m
5	0.1	0.1	245
5	0.1	0.01	268
5	0.01	0.1	2450
5	0.01	0.01	2680
10	0.1	0.1	7123
10	0.1	0.01	7146
10	0.01	0.1	71230
10	0.01	0.01	71460

Tabela 3.7: Número de Exemplos para Aprendizado PAC no Espaço de Todas as Funções Booleanas

A Figura 3.2 mostra graficamente o número mínimo de exemplos, m , em função do número de atributos, n e δ para $\epsilon = 0.01$ e 0.001 .

Desta forma, o algoritmo não terá capacidade de aprender a menos que considere um espaço de hipótese mais restrito. Mas, se o espaço de hipótese for restringido, pode acontecer que a verdadeira função f_v não faça parte desse espaço. De qualquer forma, pode-se pensar que na maioria dos casos não é necessário utilizar todo o poder de expressão das funções booleanas, e limitar o aprendizado a linguagens de descrição mais restritas.

Por fim, deve-se ressaltar que a análise PAC é uma análise de pior caso. Para todas as possíveis distribuições resultantes de um conjunto de amostras, ela garante que os resultados de classificação serão corretos com uma pequena margem de erro. Embora este tipo de análise forneça limites teóricos interessantes para taxas de erro, mesmo para classificadores simples os resultados indicam que um grande número de casos é necessário para garantir a performance.

Baseados nestes resultados teóricos, pode-se ficar desencorajado em tentar estimar a taxa de erro verdadeira para um classificador. Mas resultados melhores podem ser obtidos para problemas reais, uma vez que dada uma amostra de uma simples população, a tarefa está em estimar a taxa de erro para aquela população, não para todas as possíveis populações. Este tipo de análise requer muito menos casos, uma vez que uma única distribuição da população é considerada. Além disso, em vez de todos os casos serem usados para estimar a taxa de erro verdadeira, os casos podem ser particionados em dois grupos, alguns utilizados para projetar o classificador, outros para testar o classificador. Apesar de que esta forma de análise não fornece garantias de performance para todas as possíveis distribuições, ela fornece uma estimativa da taxa de erro verdadeira para a população sendo considerada. Ao contrário da análise PAC, o número de casos necessários, neste outro tipo de análise, para o conjunto de teste é surpreendentemente pequeno.

Figura 3.2: Número de Exemplos, m , em Função de n e δ para $\epsilon = 0.01$ e 0.001 .

Na próxima seção serão apresentados alguns métodos estatísticos capazes de estimar a taxa de erro verdadeira, de forma confiável, para problemas reais os quais envolvem pequenos conjuntos de exemplos. Estes métodos são baseados no princípio de *treinar-e-testar*.

3.5 Técnicas para Estimar a Taxa de Erro Verdadeira

Uma vez que a taxa de erro aparente não é uma boa estimativa para a taxa de erro verdadeira, alguma alternativa deve ser encontrada. O termo *estimativa honesta de taxa de erro* é muitas vezes utilizado, mas não deve ser mal interpretado, pois se refere a estimadores imprecisos e não desonestos. Algumas vezes, taxas de erro aparente têm sido usadas para estimar a performance de classificadores, mas freqüentemente esses fatos estão relacionados com a falta de familiaridade com métodos estatísticos adequados para estimação de erros.

Um dos requisitos para estimar a taxa de erro verdadeira é manter a amostra de exemplos em ordem aleatória. Isto significa que a amostra de exemplos não deve ser pre-selecionada, de forma a evitar que seja feita qualquer decisão sobre exemplos mais e menos representativos.

3.5.1 Estimação da Taxa de Erro Através de Treinar-e-Testar

O princípio básico que envolve os métodos de treinar-e-testar está em dividir a amostra de exemplos em dois grupos mutuamente exclusivos, como ilustrado na Figura 3.3. O primeiro, chamado de *conjunto de treinamento*, é usado para treinar o sistema de aprendizado. O segundo, chamado de *conjunto de teste*, é usado exclusivamente para medir a taxa de erro do classificador, chamada de *taxa de erro do conjunto de teste*. Se o conjunto de teste for “escondido” do classificador e mostrado somente depois do treinamento, então tem-se uma correspondência procedural direta à tarefa de determinar a taxa de erro em novos casos.

Figura 3.3: Conjuntos de Treinamento e Teste

Os dois conjuntos de casos devem ser amostras aleatórias de alguma população. Em adição, os casos dos dois conjuntos de amostras devem ser independentes. Independente significa que não há relações entre os conjuntos com exceção que ambos são amostras de uma mesma população.

Na seção 3.4, uma questão muito abrangente foi levantada a respeito do número de casos que devem pertencer a uma amostra para garantir uma performance equivalente no futuro. Se nenhuma suposição for feita a respeito da verdadeira distribuição da população, os resultados não são muito satisfatórios pois é necessário um número muito grande de casos. Entretanto, se conjuntos independentes de treinamento e teste são utilizados, excelentes resultados práticos são conhecidos. Com esta suposição, surgem as seguintes questões

Quantos casos de teste são necessários para uma estimação de erro precisa?

A mesma questão pode ser reescrita como

Quantos casos de teste são necessários para que a taxa de erro do conjunto de teste seja essencialmente a taxa de erro verdadeira?

Surpreendentemente, pode-se obter excelentes aproximações da taxa de erro verdadeira, mesmo com amostras de exemplos pequenas. A Figura 3.4 apresenta um gráfico que mostra a relação entre a taxa de erro do conjunto do teste e a mais alta possível taxa de erro verdadeira, para conjuntos de teste de vários tamanhos. Esses valores possuem 95% de confiabilidade, isto é, não há mais de 5% de probabilidade que a taxa de erro exceda os valores apresentados. Por exemplo, para um conjunto de teste de tamanho 50, se taxa de erro do conjunto do teste apresentar um valor de 0%, então há uma grande probabilidade que a taxa de erro verdadeira seja no máximo de 10%. Para 1000 casos de teste e taxa de erro do conjunto de teste de 0%, certamente a taxa de erro verdadeira ficará abaixo de 1%.

Figura 3.4: Número de Casos de Teste e Qualidade da Predição [Weiss 91]

Portanto, a qualidade da estimativa por taxa de erro do conjunto de teste é diretamente dependente do número de casos do conjunto de teste. Quando o tamanho do conjunto de teste atinge 1000 casos, a estimativa já é bastante precisa. Com 5000 casos, a taxa de erro do conjunto de teste é virtualmente idêntica à taxa de erro verdadeira.

O gráfico da Figura 3.4 é proveniente de resultados estatísticos conhecidos. No caso do problema de classificação, um classificador é construído para responder com a classe a que um caso pertence. Quando se está medindo a taxa de erro sobre um conjunto de exemplos, a classe respondida pelo classificador é comparada com a classe à qual o caso realmente pertence. Se o classificador acertou então houve sucesso na classificação,

caso contrário a classificação falhou. Este tipo de experimento no qual os elementos de uma população podem assumir dois resultados possíveis, freqüentemente chamados de sucesso e falha, são chamados de *ensaios de Bernoulli*. Considerando n ensaios de Bernoulli, tal que a probabilidade de cada ensaio ser um sucesso (ou fracasso) é p , então a variável aleatória X representando o número de sucessos (ou de fracassos) nos n ensaios tem uma distribuição binomial.

O gráfico da Figura 3.4 apresenta resultados em termos de taxa de erro, portanto deve-se considerar p como sendo a probabilidade do classificador errar na classificação, ou seja, p é taxa de erro verdadeira. É possível encontrar com 95% de certeza o intervalo de confiança para p a partir do número de casos da amostra n e da taxa de erro no conjunto de teste E_t . Como $n \geq 30$ pode-se utilizar a distribuição normal como uma boa aproximação da distribuição binomial. Desta forma, pode-se calcular o intervalo de confiança através da equação

$$d = 1.96 \sqrt{\frac{E_t(1 - E_t)}{n}}$$

Desta forma é garantido com 95% de confiabilidade que a taxa de erro verdadeira p está contida no intervalo

$$(E_t + d, E_t - d)$$

Como dito anteriormente, o gráfico da Figura 3.4 mostra a maior taxa de erro possível para conjuntos de teste de vários tamanhos. Desta forma, os valores representados no gráfico são os limites superiores do intervalo de confiança. Por exemplo, o gráfico mostra que para um conjunto de teste de 100 casos e uma taxa de erro do conjunto de teste de 0.3, a maior taxa de erro verdadeira possível com 95% de confiança é de 0.4. Através das equações pode-se confirmar esses valores, pois

$$d = 1.96 \sqrt{\frac{0.3(1 - 0.3)}{100}} = 0.08$$

Portanto o limite superior do intervalo de confiança é de $0.30 + 0.08 = 0.38 (\cong 0.4)$.

Na prática o número de exemplos disponível é freqüentemente restrito. Desta forma, não só um número suficiente de casos de teste, mas também a forma de como devem ser escolhidos os casos para treinamento e teste é de essencial importância. Uma pergunta que ainda falta ser respondida é

Quantos casos deve conter cada conjunto de teste e treinamento?

Existem várias formas conhecidas para dividir a amostra em conjuntos de treinamento e teste. O *Método Holdout* ou *Método H* divide fixamente a amostra de exemplos,

geralmente em $\frac{2}{3}$ para treinamento e $\frac{1}{3}$ para teste. Quando o número de casos para teste é grande (excede 1000 casos) costuma-se aumentar a fração do conjunto de treinamento.

Deve ser salientado que o método holdout apresenta uma taxa de erro do conjunto de teste que aproxima a taxa de erro verdadeira para grandes amostras de exemplos. Entretanto, para pequenas amostras de exemplos, os conjuntos de treinamento e teste ficam com tamanhos muito reduzidos, a aproximação se torna pessimista e o método holdout não apresenta resultados satisfatórios. Desta forma, quando o conjunto de exemplos disponível não é grande, podem ser aplicadas variações dos métodos de treinar-e-testar conhecidas por métodos *resampling*. Esses métodos serão discutidos a seguir.

3.5.2 Métodos Resampling

O método holdout é a técnica mais simples de estimativa de erro “honesta” conhecida e consiste em realizar somente um experimento treinar-e-testar. Entretanto uma única partição pode levar a resultados imprecisos, principalmente para pequenas e médias amostras de exemplos. Os métodos de resampling são variações dos métodos de treinar-e-testar nos quais são realizados vários experimentos de treinar-e-testar com diferentes partições de exemplos. Como resultado de cada experimento treinar-e-testar é obtida uma taxa de erro, e a média dessas taxas de erro é a estimativa para a taxa de erro verdadeira. Procura-se encontrar um estimador não *tendencioso*, ou seja que não estime a taxa de erro de forma otimista nem pessimista. O conceito chave de um estimador não tendencioso é a convergência para o valor verdadeiro quando aplicado a uma amostra de exemplos suficientemente grande. O estimador pode variar de amostra para amostra, mas sobre todas as amostras a média é correta. Um exemplo de estimador não tendencioso é a estimação da taxa de erro sobre um conjunto de teste. Enquanto que a estimação para um conjunto de teste pode diferir do valor verdadeiro, o valor médio das estimativas para todos os possíveis conjuntos de teste é o mesmo que o valor verdadeiro [Weiss 96].

A seguir são descritos os seguintes métodos de resampling

- Random subsampling
- Cross-validation
- Bootstrapping

Random Subsampling Quando múltiplos experimentos de treinar-e-testar são realizados, um novo classificador é projetado com cada conjunto de treinamento. A taxa de erro estimada é a média das taxas de erros dos classificadores calculada a partir de conjuntos de teste independentes e aleatórios. O método que realiza tal procedimento

é conhecido como *Random Subsampling* e pode produzir melhores resultados que uma simples partição treinar-e-testar.

A Tabela 3.8 mostra uma comparação entre os métodos Holdout e random subsampling para uma amostra de n casos. O método random subsampling soluciona um problema típico do método holdout, a escolha de uma partição dos exemplos que não é representativa aos conceitos. Com o método random subsampling este problema é contornado pois são criadas várias partições e o sistema é treinado e testado a cada iteração com uma partição diferente. No caso de uma partição ser pouco representativa, tal problema pode ser amenizado pois a taxa de erro final é calculada através das médias das taxas de erro de todas as partições.

	<i>Holdout</i>	<i>Random Subsampling</i>
<i>Casos Treinamento</i>	j	j
<i>Casos de Teste</i>	$n - j$	$n - j$
<i>Iterações</i>	1	num. iterações $\ll n$

Tabela 3.8: Comparação entre Holdout e Random Subsampling

Cross-validation Cross-validation não é um único método, mas sim uma classe de métodos para estimar a taxa de erro verdadeira. Também são conhecidos como *k-fold cross-validation*, onde k representa o número de partições geradas aleatoriamente a partir da amostra de exemplos para treinar e testar o sistema. Nestes métodos, a amostra de exemplos é dividida em k partições mutuamente exclusivas. A cada iteração, uma partição diferente é usada para testar o sistema de aprendizado e todas as outras $k - 1$ partições são utilizadas para treinar o sistema. A taxa de erro é a média das taxas de erro calculadas para as k iterações. Por exemplo, no caso do 10-fold cross-validation todo o conjunto de exemplos é dividido em 10 partições diferentes, aleatórias e mutuamente exclusivas. São realizadas 10 iterações, a cada iteração uma partição diferente é utilizada para testar e as outras 9 partições restantes são utilizadas para treinar o sistema. A estimativa da taxa de erro verdadeira é resultado da média das taxas de erro das 10 iterações.

O método *Leaving-one-out* é um caso especial da classe de métodos cross-validation. Neste método, para uma amostra de n exemplos, são realizadas n iterações. A cada iteração um exemplo diferente é retirado do conjunto de exemplos e utilizado para testar o classificador. Os outros $n - 1$ exemplos são usados para treinar o sistema. A taxa de erro é calculada dividindo-se o número de erros observados por n .

O leaving-one-out é muito utilizado pois sua estimativa de taxa de erro verdadeira é praticamente não tendenciosa, isto é, quando usados vários conjuntos de exemplos, a média das estimativas tenderá à taxa de erro verdadeira.

Por ser computacionalmente caro, o leaving-one-out é geralmente utilizado somente em pequenos conjuntos de exemplos. Para conjuntos maiores, o k-fold cross-validation, com

$k = 10$, apresenta uma aproximação quase tão apurada quanto o leaving-one-out, a um custo muito inferior. A Tabela 3.9 mostra uma comparação entre estes dois métodos de cross-validation.

	<i>Leaving-One-Out</i>	<i>10-fold Cross-Validation</i>
<i>Casos Treinamento</i>	$n - 1$	90%
<i>Casos de Teste</i>	1	10%
<i>Iterações</i>	n	10

Tabela 3.9: Comparação entre Métodos de Cross-validation

Uma das vantagens de utilização dos métodos de cross-validation é que todos os casos da amostra de exemplos são usados para testar e todos os casos são usados para treinar o sistema.

Bootstrapping O problema de encontrar o melhor estimador para pequenas amostras é particularmente intrigante uma vez que pequenas amostras são bastante comuns. Tradicionalmente são consideradas pequenas amostras aquelas que possuem 30 ou menos casos. Por muitos anos, o leaving-one-out foi a técnica recomendada para avaliar a performance de classificadores para pequenas amostras. Uma vez que o leaving-one-out é virtualmente não tendencioso, ele pode ser aplicado a amostras muito maiores, apresentando resultados precisos.

Para amostras pequenas, o bootstrapping, um novo método de resampling, tem mostrado resultados promissores como estimador de taxa de erro. Esta é uma área de pesquisa ativa em estatística aplicada.

Embora o leaving-one-out seja um estimador quase não tendencioso da taxa de erro verdadeira, ainda existem dificuldades com esta técnica. Tendência e variância de um estimador de taxa de erro contribuem para a imprecisão da taxa de erro estimada. Enquanto que o leaving-one-out é praticamente não tendencioso, sua variância para pequenas amostras é alta. Lembre-se que não tendencioso significa que o estimador, após várias aplicações, tende para a taxa de erro verdadeira. Porém a variância do leaving-one-out é alta, o que pode ser comparado a um bêbado tentando andar sobre uma linha reta. Esta pessoa pode apresentar uma média exatamente sobre a linha, mesmo quando constantemente cambaleia para a direita e esquerda da linha.

O efeito da variância tende a dominar em pequenas amostras. Desta forma, um estimador com baixa variância, mesmo que tendencioso, é potencialmente superior ao leaving-one-out para pequenas amostras.

Existem vários estimadores bootstrap, mas dois estimadores em particular têm mostrado os melhores resultados, o *$\epsilon 0$ bootstrap* e o *.632 bootstrap*. Para o estimador *$\epsilon 0$* , o conjunto de treinamento consiste de n casos amostrados com reposição a partir de uma amostra de tamanho n . Amostrados com reposição significa que os exemplos de

treinamento são retirados do conjunto inicial de exemplos mas cópias dos elementos retirados permanecem no conjunto inicial, de forma que um mesmo elemento pode ser amostrado repetidas vezes.

A fim de exemplificar, considerando-se existir uma amostra de casos contendo apenas sete casos

Amostra de casos:

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Para construir o conjunto de treinamento deve-se sortear um caso aleatoriamente da amostra de casos. Considerando que o caso número 6 é sorteado, ele é copiado para o conjunto de treinamento. Os casos devem ser copiados e não retirados da amostra inicial, desta forma a amostra inicial deve sempre conter os mesmos casos. Novamente, outros casos são sorteados aleatoriamente e copiados, existindo assim a possibilidade de um mesmo caso ser copiado mais de uma vez para o conjunto de treinamento. Os casos são copiados até que o conjunto de treinamento possua o mesmo número de casos que a amostra de casos inicial, ou seja, sete casos. Por exemplo, o conjunto de treinamento poderia conter

Conjunto de treinamento:

1	2	3	3	3	6	6
---	---	---	---	---	---	---

O conjunto de teste é composto por todos os casos que não pertencem ao conjunto de treinamento, neste exemplo este conjunto seria composto pelos seguintes casos

Conjunto de teste:

4	5	7
---	---	---

Esta técnica possui uma fração média de casos não repetidos para o conjunto de treinamento de 0.632, e uma fração média de 0.368 para casos do conjunto de teste. O estimador .632 bootstrap é a combinação linear simples de

$$.368 * app + .632 * e0$$

Onde *app* é a taxa de erro aparente para todos os casos (treinamento e teste). Deve ser notado que *e0* pode ser aproximado através da repetição do método 2-fold cross-validation, isto é, a divisão meio a meio da amostra inicial para treinamento e teste.

A taxa de erro final é calculada através da média das taxas de erro das várias iterações. Cerca de 200 iterações são consideradas necessárias para obter uma boa estimativa com o bootstrap. Portanto, este método pode ser considerado mais caro computacionalmente que o método leaving-one-out. A Tabela 3.10 resume as características básicas do método bootstrap.

	<i>Bootstrap</i>
<i>Casos Treinamento</i>	n (com j casos não repetidos)
<i>Casos de Teste</i>	$n - j$
<i>Iterações</i>	200

Tabela 3.10: Características dos Estimadores Bootstrap

3.6 Problemas Comuns em Classificação

Um classificador que classifica bem os exemplos que foram utilizados no treinamento, mas que não classifica bem novos exemplos não possui utilidade. Como já mencionado, usar somente a taxa de erro aparente para estimar a performance pode levar a resultados desastrosos em novos casos.

Se a taxa de erro aparente fosse um bom estimador para a taxa de erro verdadeira, o problema de classificação estaria resolvido, pois através da técnica de tabela lookup pode-se obter um classificador com taxa de erro aparente zero. Na técnica de tabela lookup os próprios exemplos se tornam o classificador e, através de uma simples busca na tabela pode-se encontrar facilmente a resposta para casos conhecidos. Infelizmente, a probabilidade de encontrar um novo caso na tabela é pequena, pois o número de combinações possíveis de atributos pode ser imensa.

A natureza deste problema, que pode ser mais facilmente visualizado com o caso da tabela lookup, é chamado de *overspecialization* ou *overfitting* do classificador sobre os dados. Durante o processo de treinamento, existe um esforço muito grande do sistema de aprendizado para “acomodar” os exemplos de treinamento. Tal esforço pode levar a uma extração de regras pouco genéricas para novos casos, ocorrendo assim o processo de overfitting. A tabela lookup é um caso extremo, onde as regras só podem ser aplicadas com segurança aos casos conhecidos. Neste caso utilizar a taxa de erro aparente para estimar a performance pode levar a um falso sentimento de segurança.

Felizmente, existem técnicas bem mais confiáveis capazes de estimar a taxa de erro verdadeira, como por exemplo, as técnicas de resampling descritas anteriormente.

Um dos fatores mais relevantes para que um sistema de aprendizado consiga extrair boas regras de decisão está na capacidade de predição dos atributos da amostra de exemplos. Isto ocorre pois o sistema de aprendizado depende exclusivamente dos exemplos e da qualidade dos atributos para aprender. Por exemplo, na tentativa de prever se uma pessoa possui câncer de pulmão, um atributo que poderia ser utilizado é se a pessoa é fumante. Com certeza este atributo possui maior poder de predição que o simples acaso. Por outro lado, se fosse utilizada a altura da pessoa, este atributo poderia se mostrar muito menos preditivo, ou até mesmo nulo para o processo de aprendizado. É importante que existam atributos preditivos e que o sistema de aprendizado saiba escolher entre todos os atributos quais são aqueles potencialmente mais preditivos.

São classificados como *ruído* aqueles atributos que não possuem poder de predição, ou seja, contribuem somente com o acaso. Alguns atributos considerados a princípio apenas ruído podem se tornar altamente preditivos quando combinados com outros atributos, num processo conhecido como indução construtiva, tratada na seção 4.5, pg. 56.

Por esses fatores, os atributos devem ser selecionados com critério. Os responsáveis por selecionar quais atributos devem ser observados ou medidos devem ser especialistas no domínio. Sobretudo, os atributos selecionados devem ser corretamente observados e as classes corretamente classificadas. Voltando ao exemplo do câncer, se um atributo escolhido foi o peso, este deve ser corretamente medido e anotado. É importante também que a classe esteja corretamente classificada, ou seja, que não existam diagnósticos incorretos da doença.

Felizmente, problemas na coleta dos dados e de qualidade dos atributos podem ser identificados facilmente. Quando problemas deste tipo ocorrem, os classificadores podem apresentar taxas de erro aparente muito baixas, entretanto não costumam apresentar uma boa performance para novos casos. Utilizando métodos de treinar-e-testar ou re-sampling, pode-se diagnosticar esses problemas através da estimativa da taxa de erro verdadeira.

3.7 Critérios de Avaliação de Sistemas de Aprendizado

A taxa de erro é um índice que mede mais diretamente a performance de um sistema de aprendizado. Entretanto, a qualidade das regras geradas por sistema de aprendizado pode ser analisada segundo outros critérios. Alguns destes critérios podem ser facilmente medidos como a precisão e a velocidade de classificação, outros como a compreensibilidade são muito mais complexos de serem avaliados e dimensionados. Alguns critérios que podem ser utilizados para medir a qualidade das regras geradas por um sistema de aprendizado são descritos em [Henery 94].

- **Precisão.** É o grau de confiabilidade das regras. Geralmente é representado pela taxa de erro, embora em alguns casos existam erros mais sérios que outros, e em certos casos pode ser importante controlar a taxa de erro em algumas classes chaves;
- **Velocidade.** Em algumas circunstâncias a velocidade de um classificador é o aspecto mais importante. Um classificador que é 90% preciso pode ser preferido à outro que é 95% preciso se ele for 100 vezes mais rápido nos testes. Tais considerações podem ser importantes para aplicações de tempo real, isto é, aplicações que possuem sérias restrições de tempo nas suas tarefas;
- **Compreensibilidade.** Se é um operador humano quem deve aplicar o procedimento

de classificação, o procedimento deve ser facilmente compreensível ou senão enganos podem ser feitos na aplicação das regras. Um exemplo é o caso Three-Mile Island, onde os dispositivos automáticos corretamente recomendaram o desligamento, mas esta recomendação não foi executada pelos operadores humanos que a consideraram infundada. Uma história similar se aplica ao desastre de Chernobyl;

- Tempo de Aprendizado. Especialmente em um ambiente que muda rapidamente, pode ser necessário aprender uma regra de classificação rapidamente, ou fazer ajustes a uma regra existente em tempo real. “Rapidamente” pode implicar também que seja necessário somente um pequeno número de observações para estabelecer a regra.

3.8 Considerações Finais

O objetivo geral de um sistema de classificação está em aprender a partir de amostras de exemplos e generalizar para novos casos nunca vistos. A performance dos classificadores é facilmente medida em termos da taxa de erro, que pode ser calculada através da razão do número de erros pelo número total de casos da amostra. Enquanto que, freqüentemente, é muito fácil projetar um classificador que cometa poucos erros na amostra de treinamento, é muito mais difícil obter esta performance para novos casos. Em situações em que o classificador comete muitos erros para novos casos é dito que o classificador se especializou ou super-ajustou aos dados de treinamento. Para evitar esta especialização, o sistema de aprendizado deveria ter seus resultados analisados por uma técnica capaz de estimar a taxa de erro verdadeira, a fim de prever qual será a sua performance para novos casos. Técnicas de estimativa de taxa de erro provêm a base para a comparação da performance de sistemas de aprendizado sobre uma mesma amostra.

As técnicas de estimação da taxa de erro são extremamente simples no conceito e muito precisas na prática. Tais técnicas estão baseada na teoria estatística de resampling. Utilizando as amostras iniciais, estas técnicas repetidamente simulam a apresentação de novos casos. O que diferencia as diversas técnicas apresentadas é a forma de como são divididos os conjuntos de treinamento e teste.

O método holdout pode ser utilizado para grandes amostras de exemplos, mas existe o risco de ser escolhida uma partição não representativa. Os métodos de resampling corrigem este problema apresentando várias vezes os dados ao sistema de aprendizado. Entre todas as técnicas, o leaving-one-out se destaca por ser praticamente não tendencioso, mas por ser computacionalmente caro, este método não é muito utilizado para conjuntos maiores, nos quais o 10-fold cross-validation é mais apropriado. Para conjuntos muito pequenos, a variância do leaving-one-out é muito grande e outro método deve ser escolhido. O candidato mais forte é o bootstrap que é menos preciso que o leaving-one-out, mas sua variância é baixa para pequenas amostras.

Capítulo 4

Sistemas de Aprendizado Simbólico Proposicional

4.1 Considerações Iniciais

Diversos paradigmas de aprendizado indutivo por exemplos foram discutidos na seção 2.3 pg. 7. Alguns desses paradigmas realizam o aprendizado através de métodos numéricos ou estatísticos, os quais são de difícil compreensão para a maioria das pessoas não familiarizadas com a área de aprendizado de máquina. Desta forma, mesmo quando os métodos empregados são simples, como um discriminador linear, esses sistemas podem ser vistos como “caixas pretas” por seus usuários, uma vez que essas pessoas não sabem ao certo por que motivos um caso foi classificado em uma determinada classe. Essa falta de compreensão do funcionamento do método utilizado pode levar a diversos problemas e resultar na má utilização do sistema.

Problemas como este fizeram com que alguns pesquisadores procurassem por sistemas de aprendizado capazes de induzir conceitos que fossem representados por uma linguagem facilmente compreensível por um ser humano. A esses sistemas deu-se o nome de *sistemas de aprendizado simbólico indutivo*.

Uma forma de representar os conceitos de forma compreensível é através da lógica de regras de decisão associativa, uma vez que regras da forma

Se X é verdade e Y é falso, conclua classe C

podem ser facilmente entendidas e aceitas por humanos.

Estas regras são simples combinações de proposições que podem ser avaliadas em verdadeiro ou falso. Cada regra de decisão é associada com uma classe particular, e quando

uma regra é satisfeita, ou seja, avaliada como verdadeira, esta regra indica uma classe em particular.

Enquanto que a forma mais simples de uma regra é a conjunção de proposições (operador lógico “E”¹), tipicamente mais de uma regra poderá ser necessária para cobrir uma certa classe. Quando um conjunto de regras é necessário para classificar um objeto, logicamente as regras formam um conjunto disjuntivo (operador lógico “OU”²). O uso de ambos os juntores $\wedge$ e $\vee$ em uma mesma regra pode causar resultados ambíguos. O formato que foi descrito há pouco é a forma mais comum de prevenir ambiguidades, limitando as regras a conjunção de proposições (termos), mas permitindo disjunções em um conjunto de regras (expressões). Conjuntos de regras neste formato estão na *forma normal disjuntiva*. Por exemplo, para que $X \vee (Y \wedge Z)$ seja verdadeiro, X deve ser verdadeiro ou ambos Y e Z devem ser verdadeiros.

É importante dizer que existem outras formas além de regras associativas para representar conceitos aprendidos de forma compreensível aos humanos, como por exemplo redes semânticas. Entretanto, este capítulo se limita a discutir apenas alguns métodos para a indução de regras associativas e de árvores de decisão, uma vez que muitos dos recentes sistemas de aprendizado simbólico utilizam esses métodos. Na seção 4.3 é discutida a indução de árvores de decisão, as quais podem ser convertidas em regras associativas. Na seção 4.4 é discutida a indução de regras de decisão, dando-se ênfase ao método de separar-e-conquistar.

Na próxima seção são discutidas as *linguagens de descrição* e como estas linguagens podem influenciar no processo de aprendizagem de um sistema simbólico.

4.2 Linguagens de Descrição

Basicamente, um sistema de aprendizado indutivo supervisionado aprende através de exemplos de casos solucionados. Para problemas mais complexos, pode também ser utilizado um conhecimento prévio a respeito do problema. Tal conhecimento prévio utilizado para subsidiar o processo de aprendizado é freqüentemente conhecido como *teoria de domínio* ou *conhecimento de fundo*.

Em um sistema de aprendizado é necessário utilizar linguagens para representar ambos, exemplos e teoria de domínio, além dos conceitos induzidos. Estas linguagens são comumente referenciadas como *linguagens de descrição*, mais especificamente chamadas de

- *Linguagem de descrição de exemplos ou instâncias* ($\mathcal{L}_E$)

¹representado pelo símbolo lógico $\wedge$

²representado pelo símbolo lógico $\vee$

- *Linguagem de descrição de hipóteses ou conceitos* ($\mathcal{L}_{\mathcal{H}}$)
- *Linguagem de descrição de teoria de domínio ou conhecimento de fundo* ($\mathcal{L}_{\mathcal{D}}$)

A linguagem de descrição de exemplos ou instâncias descreve os exemplos utilizados pelo programa para aprender conceitos, a linguagem de descrição de conceitos ou hipóteses descreve o estado interno de um programa de aprendizado, correspondente à sua teoria dos conceitos ou padrões que existem nos dados e, por último, a linguagem de descrição de teoria de domínio ou conhecimento de fundo descreve todo o conhecimento prévio que o programa possui a respeito do problema.

Um programa é limitado aos conceitos que pode aprender pela capacidade representacional de suas linguagens de descrição [Mitchell 80]. Por exemplo, se uma linguagem do tipo atributo-valor for utilizada para representar os exemplos para um programa de indução, os valores de certos atributos claramente estabelecem limites sobre os tipos de padrões que o sistema pode aprender. O programa é dito *influenciado* pela linguagem de descrição de exemplos ou instâncias. A linguagem de descrição de conceitos ou hipóteses também estabelece limites sobre o que pode ou não ser aprendido. Por exemplo, na linguagem do tipo atributo-valor, relações entre objetos são difíceis de representar. Uma linguagem mais expressiva, tal como uma linguagem lógica de primeira ordem, pode facilmente ser utilizada para representar relações³.

Por exemplo, sejam A e B dois atributos inteiros e deseja-se expressar o conceito: *A possui o mesmo valor que B*. Utilizando uma linguagem proposicional, este conceito só pode ser expresso enumerando todos os possíveis valores para os quais A é igual a B . Por exemplo

$$\begin{aligned} &(A = 1 \wedge B = 1) \vee \\ &(A = 2 \wedge B = 2) \vee \\ &(A = 3 \wedge B = 3) \vee \\ &\dots \end{aligned}$$

Em uma linguagem lógica de primeira ordem é possível representar relações entre atributos, como por exemplo a relação de igualdade. Desta forma, o mesmo conceito pode ser definido da seguinte forma

$$A = B$$

Infelizmente, o poder representacional das linguagens de descrição tem o seu preço. O aprendizado pode ser visto como uma busca no espaço de todas as sentenças da

³Sistemas de aprendizado que utilizam linguagem lógica de primeira ordem para representar hipóteses são discutidos no Capítulo 5.

linguagem, por uma sentença que melhor descreve os dados. Quanto mais rica for a linguagem de representação, maior será o espaço de busca. Quando um espaço de busca é pequeno, é possível usar métodos de busca do tipo “força bruta”. Entretanto, se o espaço de busca é muito grande, algum conhecimento adicional é requerido para reduzir este espaço de busca.

Os sistemas que utilizam a linguagem atributo-valor para descrever os exemplos, e eventualmente, contra-exemplos de treinamento, são chamados de sistemas de aprendizado simbólico proposicional, pois a linguagem atributo-valor possui a mesma força expressiva que uma linguagem lógica proposicional. Estes sistemas de aprendizado freqüentemente utilizam árvores de decisão (seção 4.3) ou regras (seção 4.4) para descrever os conceitos induzidos.

Assim, apesar de ser amplamente difundida, a linguagem atributo-valor impõe sérias restrições aos conceitos que podem ser aprendidos. Uma tentativa de superar algumas das restrições impostas por esta linguagem é conhecida como *indução construtiva* e, é discutida na seção 4.5.

4.3 Árvores de Decisão

Uma árvore de decisão é uma estrutura que consiste de nós e ramos. O nó inicial é freqüentemente chamado de nó raiz. Cada nó representa um teste ou uma decisão. No caso de uma árvore binária, como ilustra a Figura 4.1, os testes podem ser avaliados em verdadeiro ou falso. Dependendo se o resultado do teste é verdadeiro ou falso, a árvore irá se ramificar para a direita ou para a esquerda em direção a outro nó. Finalmente, quando um nó terminal (também chamado de nó folha) é encontrado, uma decisão pode ser feita à respeito da classificação.

Figura 4.1: Exemplo de Árvore de Decisão Binária

Na Figura 4.1, o nó 1 é o nó raiz e o nó 2 é o único nó não terminal além do nó raiz. Os nós sombreados são nós terminais. Quando um caso é apresentado para classificação, o teste representado pelo nó 1 é realizado primeiro, se não for satisfeito (falso) o sistema

responde com a classe $C1$. Entretanto se o teste for satisfeito (verdadeiro), outro teste deverá ser avaliado no nó 2. Dependendo da avaliação realizada no segundo teste, o sistema poderá responder com a classe $C2$, se o teste for falso, ou com a classe $C1$, se o teste for avaliado como verdadeiro.

O processo para construir a estrutura de uma árvore de decisão a partir de dados é conhecido como *indução de árvores de decisão*. Uma árvore de decisão pode ser induzida selecionando algum atributo inicial, dividindo os casos segundo este atributo em conjuntos disjuntos, e então repetindo esse procedimento para todos os nós subsequentes. Os nós se tornam terminais e portanto não são mais divididos quando todos os exemplos deste nó pertencem à mesma classe. Alternativamente, esta última condição pode ser relaxada, tal que um nó pode se tornar terminal mesmo se todos os casos não pertecerem a mesma classe. Neste caso, um número mínimo de casos de outras classes pode ser aceitável e o nó terminal é associado à classe de maior frequência.

A forma mais simples de dividir os nós em grupos disjuntos é particionar os dados por valores distintos de um atributo. Para ilustrar o processo de construção de uma árvore de decisão, considere um conjunto de exemplos para reconhecer se um robô é amigo ou inimigo, dados os formatos do sua cabeça e corpo e se o robô sorri ou não. A Tabela 4.1 mostra os casos desse conjunto de exemplos.

Número Exemplo	Cabeça	Corpo	Sorri?	Classe
1	quadrada	quadrado	sim	amigo
2	quadrada	triangular	não	inimigo
3	redonda	redondo	sim	amigo
4	triangular	redondo	não	inimigo
5	triangular	triangular	não	amigo
6	redonda	quadrado	não	inimigo

Tabela 4.1: Exemplos de Robôs Amigos e Inimigos

Como já foi dito anteriormente, o processo de indução inicializa com a seleção de um atributo para ser o nó raiz. No caso deste exemplo existem três atributos chamados de *cabeça*, *corpo* e *sorri?*. A seleção adequada dos atributos tem uma influência direta no tamanho final da árvore. Algumas medidas de teoria da informação podem ajudar na seleção dos atributos, uma das medidas mais utilizada é conhecida como *entropia* e será discutida mais adiante. Por hora, é selecionado o primeiro atributo, cabeça, para ser o nó raiz da árvore de decisão. A Figura 4.2 ilustra esta árvore.

Com a seleção do atributo *cabeça* para ser o nó raiz da árvore de decisão, os exemplos da Tabela 4.1 podem ser particionados conforme este atributo. Como ilustra a Figura 4.2, os exemplos de número 1 e 2 possuem cabeça quadrada e, desta forma, compõem o nó $N2$ da árvore, como estes dois exemplos pertencem a classes diferentes, este nó deverá ser particionado novamente. Os exemplos 3 e 6 possuem cabeça redonda e compõem o nó $N3$, e o nó $N4$ possui os exemplos 4 e 5. Os nós $N3$ e $N4$ também possuem exemplos de classes diferentes e devem ser particionados novamente. É necessário selecionar um

Figura 4.2: Atributo Cabeça como Nó Raiz da Árvore de Decisão

outro atributo para particionar os exemplos pertencentes aos nós $N2$, $N3$ e $N4$. Uma vez que, por hora, está-se selecionando os atributos de forma aleatória, é utilizado o segundo atributo, corpo, para particionar os exemplos desses nós.

A Figura 4.3 mostra a árvore de decisão com os exemplos dos nós particionados conforme o segundo atributo, corpo, com isso foram criados os nós de $N5$ a $N10$. Cada um deles possui apenas exemplos de uma classe (na realidade, cada um desses nós possui apenas um exemplo) e, desta forma, não precisam ser mais particionados e devem ser convertidos em nós folhas. Como não existem mais nós para serem particionados, o processo de indução da árvore de decisão é encerrado. Observa-se que o terceiro atributo nem mesmo foi utilizado na indução da árvore de decisão.

Figura 4.3: Árvore de Decisão Induzida para o Exemplo dos Robôs

Para qualquer árvore de decisão, todos os caminhos que partem do nó raiz e levam a um nó terminal correspondem a uma regra de decisão que é uma conjunção ($\wedge$) dos vários testes. Se existem caminhos múltiplos para uma mesma classe, então os caminhos representam disjunções ($\vee$). Deve ser ressaltado que em qualquer árvore de decisão todos os caminhos são mutuamente exclusivos. Para qualquer novo caso, um e somente um caminho na árvore poderá ser satisfeito.

A árvore de decisão criada no exemplo anterior, ilustrada pela Figura 4.3, corresponde as seguintes regras de decisão para as classes amigo e inimigo

Se (cabeça = quadrada) $\wedge$ (corpo = quadrado) $\vee$
(cabeça = redonda) $\wedge$ (corpo = redondo) $\vee$
(cabeça = triangular) $\wedge$ (corpo = triangular)
então conclua classe amigo.

Se (cabeça = quadrada) $\wedge$ (corpo = triangular) $\vee$
(cabeça = redonda) $\wedge$ (corpo = quadrado) $\vee$
(cabeça = triangular) $\wedge$ (corpo = redondo)
então conclua classe inimigo.

Freqüentemente, a divisão por valores de atributos pode ser feita de forma que os nós terminais consistam somente de membros de uma única classe, a não ser que, por exemplo, haja ambigüidade nos dados. Isto é, a não ser que um caso idêntico esteja associado a duas ou mais classes diferentes. Quando se está trabalhando com conjuntos de casos reais, o pior caso, na construção de árvores de decisão, ocorre quando cada caminho a um nó terminal corresponde exatamente a um dos casos no conjunto de exemplos. Entretanto, mesmo se os atributos forem escolhidos aleatoriamente para a divisão, a árvore resultante provavelmente possuirá muito menos nós terminais que casos na amostra de treinamento. Se fossem utilizados todos os valores dos atributos para separar os casos, seria garantido uma taxa de erro aparente praticamente nula. Entretanto, esse processo poderia causar um overfitting nos exemplos (seção 3.6 pg. 33). Desta forma, não se deve ficar impressionado com uma taxa de erro aparente praticamente nula, pois a taxa de erro verdadeira pode ser, eventualmente, muito mais alta, como será visto no Capítulo 7.

Até agora, na construção de árvores de decisão, os atributos foram escolhidos de forma aleatória. Na próxima seção é discutido como os atributos podem ser selecionados de forma que a árvore de decisão gerada seja mais simples e possivelmente mais preditiva na classificação de novos casos.

4.3.1 Escolha do Atributo

Como já foi dito, uma árvore de decisão pode ser induzida através de um processo repetitivo de seleção dos atributos e divisão dos casos. Esses atributos podem ser selecionados aleatoriamente, os casos divididos segundo os valores dos atributos e, eventualmente, a árvore será formada tal que todos os nós terminais conterão membros de somente uma classe, ou a maioria dos membros de uma classe caso exista ambigüidade nos dados, ou um valor mínimo de casos incorretamente classificados seja aceito.

A seleção meramente aleatória de atributos pode levar a árvores maiores. Em conjun-

tos de exemplos reais, a maioria dos atributos tem baixo poder preditivo e é necessário alguma sorte para escolher aleatoriamente os atributos mais preditivos. A seleção aleatória pode levar a uma perda de tempo dividindo os exemplos em função de atributos sem representatividade. Além do mais, árvores menores costumam apresentar melhores performances na classificação de novos casos. Por fim, uma vez que os conjuntos de exemplos são finitos, os casos podem se esgotar mesmo antes dos atributos preditivos serem escolhidos. Por exemplo, em uma árvore binária balanceada pode-se consumir 1000 casos ao atingir a profundidade de dez nós.

Após cada ramo ser construído, uma decisão deve ser feita a respeito do próximo nó a ser dividido. Uma abordagem básica para selecionar um atributo está em examinar todos os atributos e avaliar a probabilidade de cada um de melhorar a performance da árvore. Experiências com problemas reais têm mostrado que várias funções de avaliação de divisão podem fornecer bons resultados na escolha de bons atributos e na geração de árvores menores. Tais funções de avaliação trabalham somente quanto à divisão do nó atual, sem verificar o efeito dessa divisão nos nós sucessores (método é dito ser *míope* ou *guloso*). Portanto, a função de avaliação é uma heurística que tenta fazer bons julgamentos com informações incompletas.

As funções mais utilizadas de avaliação de divisão de nós trabalham reduzindo o grau de aleatoriedade ou “impureza” do nó atual. Por exemplo, durante a indução da árvore representada pela Figura 4.3 pg. 41, o sistema de aprendizado inicia analisando o nó raiz. Neste exemplo o nó raiz possui 6 casos, com 4 casos da classe amigo e 2 casos da classe inimigo. Este nó é muito impuro e não pode ser um nó terminal. A tarefa imediata é encontrar algum teste que possa dividir o nó raiz tal que se obtenha o melhor resultado. Se for encontrado algum atributo que divida o nó raiz em dois grupos tal que os 4 casos da classe amigo e os 2 casos da classe inimigo fiquem em ramos diferente, então a aleatoriedade ou impureza pode ser reduzida para 0.

As funções mais populares de avaliação são a função de *entropia* e a função *gini*. Uma vez que estas funções fornecem um índice de impureza, quanto menor for este índice, melhor será o resultado. A equação 4.1 define a função de entropia e a equação 4.2 a função gini, onde p_j é a probabilidade da classe j , a qual é aproximada pela porcentagem de casos pertencentes a classe j .

$$Entropia = - \sum_j p_j \log(p_j) \quad (4.1)$$

$$Gini = 1 - \sum_j p_j^2 \quad (4.2)$$

No exemplo de indução da árvore de decisão da Tabela 4.1 pg. 40, deve-se selecionar algum atributo para ser o nó raiz da árvore, segundo alguma função de avaliação. Para este exemplo é utilizada a função entropia. O nó raiz inicialmente possui 6 casos, com 4

casos na classe amigo e 2 casos na classe inimigo. O nó raiz possui exemplos de ambas as classes e deve-se procurar por algum atributo que divida os exemplos em ramos com o mínimo de impureza possível. A *impureza total* de uma certa divisão pode ser calculada como mostra equação 4.3, onde

- n é o nó sendo dividido;
- p_k são as probabilidades de cada ramo criado pela inclusão do atributo;
- $i(n_k)$ são as impurezas dos nós ligados aos novos ramos gerados.

A probabilidade de um novo ramo gerado é aproximada pela porcentagem de casos do nó n que irá para esse ramos.

$$Ti(n) = \sum_k p_k i(n_k) \quad (4.3)$$

Voltando ao exemplo anterior, deseja-se encontrar um atributo que forneça a maior redução de impureza. Os atributos possíveis são cabeça, corpo e sorri?. No caso do atributo cabeça os exemplos são divididos em 3 ramos cada um com dois exemplos, como mostra a Figura 4.4.

Figura 4.4: Divisão dos Exemplos do Nó Raiz Através do Atributo Cabeça

A impureza total para a divisão realizada através do atributo cabeça é

$$Ti(raiz) = \frac{2}{6} \times 1.00 + \frac{2}{6} \times 1.00 + \frac{2}{6} \times 1.00 = 1.00$$

O próximo atributo a ser verificado é o atributo corpo. Este divide os exemplos em três grupos como mostra a Figura 4.5.

A impureza total para a divisão realizada através do atributo corpo é

$$Ti(raiz) = \frac{2}{6} \times 1.00 + \frac{2}{6} \times 1.00 + \frac{2}{6} \times 1.00 = 1.00$$

Figura 4.5: Divisão dos Exemplos do Nó Raiz Através do Atributo Corpo

Figura 4.6: Divisão dos Exemplos do Nó Raiz Através do Atributo Sorri?

O último atributo, *sorri?*, divide o conjunto de exemplos em somente dois grupos como mostra a Figura 4.6

A impureza total da divisão realizada através do atributo *sorri?* é

$$Ti(raiz) = \frac{2}{6} \times 0 + \frac{4}{6} \times 0.81 = 0.54$$

Desta forma, o atributo *sorri?* deve ser escolhido, pois este divide o nó raiz com a menor entropia. Entretanto, a árvore não está totalmente induzida pois o nó a direita não pode ser convertido em um nó terminal. Portanto, deve-se escolher um outro atributo que seja capaz de dividir esse nó na busca de minimizar a impureza total desse nó.

O atributo *cabeça* pode dividir esse nó com a seguinte entropia total, como ilustra a Figura 4.7.

$$Ti(n) = \frac{2}{4} \times 0 + \frac{1}{4} \times 0 + \frac{1}{4} \times 0 = 0$$

A entropia total é igual a zero, o que significa que este atributo pode dividir os exemplos presentes no nó atual de forma que cada novo nó possua somente exemplos de uma classe, finalizando o processo de indução da árvore de decisão.

Através da seleção de atributos que fornecem a divisão de exemplos com menor impureza total é possível induzir uma árvore mais simples e, provavelmente, mais preditiva na classificação. Entretanto, muitas vezes esta abordagem não é suficiente e melhores

Figura 4.7: O Atributo Cabeça para a Divisão do Nó Direiro

resultados podem ser obtidos se ramos “fracos”, ou seja, ramos que contribuem muito pouco na classificação, fossem cortados da árvore. Este processo é chamado de *poda* de árvores de decisão e é discutido em maiores detalhes na seção seguinte.

4.3.2 Poda de Árvores de Decisão

Um problema comum em conjuntos de exemplos reais é a existência de erros nos valores dos atributos e das classes. Nesses casos é dito haver *ruído* nos dados [Bratko 90]. Ruído nos dados faz com que o aprendizado se torne mais complexo. De uma forma geral, o requisito de aprender um conceito que cobre todos os exemplos corretamente é abandonado e torna-se permitido aprender descrições que classificam incorretamente alguns exemplos. Esta concessão tem como objetivo classificar incorretamente os exemplos que possuem erros, desta forma esses exemplos estão sendo corretamente ignorados.

Uma série de procedimentos podem ser aplicados a uma árvore de decisão com o objetivo de simplificá-la, tornando o conceito aprendido mais simples e preditivo, a esses procedimentos dá-se o nome de *poda* em árvores de decisão [Castiñeira 91].

Uma forma de poda de árvores de decisão é assumir que um nó pode se tornar folha sempre que uma certa porcentagem dos exemplos pertencerem a uma mesma classe. Por exemplo, em um problema que envolve duas classes e que existe ruído nos dados, um nó pode se tornar terminal se este possuir 50 exemplos de uma classe e apenas 1 exemplo de outra classe. Com isso foi aplicado um *critério de parada* o qual decidiu não mais expandir o nó da árvore de decisão. Este tipo de poda, o qual é aplicado durante a indução da árvore de decisão, e envolve parar a indução da árvore de forma precoce, é conhecido como *pré-poda*.

Uma outra abordagem de poda consiste em analisar os ramos mais fracos de uma árvore de decisão, após a árvore ter sido totalmente induzida. Esta abordagem é conhecida como *pós-poda* e, com algumas variações, pode ser descrita da seguinte forma: um con-

junto de exemplos é fornecido à árvore de decisão para ser classificado. Esses exemplos podem ser os mesmos utilizados no treinamento, mas melhores resultados podem ser obtidos caso esses exemplos façam parte de um conjunto previamente separado. A partir dos nós folhas, cada subárvore formada por um nó não terminal ligado diretamente a nós terminais é analisada. Caso a taxa de erro no conjunto de exemplos melhore com a substituição da subárvore pelo melhor nó folha, então esta substituição é realizada, ou seja, a subárvore é podada. Este processo continua até que nenhuma melhoria possa ser feita com a poda de subárvores. Existem algumas variações de como as subárvores podem ser percorridas, mas de uma forma geral, busca-se diminuir o erro de classificação no conjunto de exemplos fornecidos.

Um dos métodos mais conhecidos de poda em árvores de decisão, chamado *Redução de Erro*, foi proposto em [Quinlan 87b] e consiste de verificar se a poda de uma subárvore pode diminuir o número de casos classificados incorretamente. Em [Mingers 89] são analisados alguns dos métodos de poda mais conhecidos em árvores de decisão.

4.3.3 Limitações dos Métodos de Indução de Árvores de Decisão

Com uma amostra grande o bastante, árvores de decisão podem teoricamente cobrir qualquer amostra. Entretanto, em situações reais com amostras finitas, as limitações presentes na estrutura de árvore podem causar alguns problemas práticos. Dois exemplos muito conhecidos podem exemplificar estas limitações.

Geometricamente, um sistema de aprendizado que encontra regras na forma normal disjuntiva está procurando estabelecer limites para as classes através de linhas paralelas aos eixos, formando regiões retangulares. Por exemplo, considerando variáveis contínuas em duas dimensões, o retângulo formado por $(X > 2 \wedge Y > 1) \wedge (X < 4 \wedge Y < 4)$ está ilustrado na Figura 4.8. Nessa figura, dois conceitos hipotéticos ($C1$ e $C2$) estão geometricamente representados por duas elipses. Um sistema de aprendizado simbólico buscará aprender os conceitos através de aproximações das elipses com regiões retangulares. Encontrando retângulos de dimensões apropriadas, um sistema de aprendizado pode cobrir qualquer classe. A eficiência com que um sistema de aprendizado consegue cobrir os dados com regiões retangulares irá determinar a performance de tal sistema.

Entretanto, muitas relações lineares não são paralelas aos eixos X e Y . Por exemplo, a função linear $x - y = 0$, representada por uma reta diagonal de 45° que passa pela origem e divide duas classes não pode ser facilmente aproximada por regiões retangulares. Uma árvore de decisão pura teria que utilizar várias regiões retangulares unidas na forma de escada, para aproximar tal trivial função linear. Este tipo de limitação tem criado esforços para tentar inserir relações lineares na representação de árvore de decisão. Uma forma de realizar este melhoramento na representação seria considerar como válidas divisões de nós que são compostas por combinações lineares de atributos. Desta forma, retas paralelas e diagonais poderiam ser utilizadas para compor superfícies complexas.

Figura 4.8: Exemplo de Interpretação Geométrica de uma Regra

Como já mencionado, uma árvore de decisão representa um conjunto de regras de decisão, onde existe uma regra para cada nó terminal da árvore. O conjunto de regras de decisão pode ser expresso na forma normal disjuntiva. Entretanto, o conjunto de regras potenciais que árvores de decisão podem representar é claramente limitado. Uma vez que as árvores começam no nó raiz, todas as regras precisam começar com o mesmo atributo. O mais importante, as regras necessariamente são mutuamente exclusivas. Isto pode levar a ineficiências e replicações de subárvores.

A Figura 4.9 apresenta uma árvore que corresponde a expressão booleana $(A \wedge B) \vee (C \wedge D)$. É associado o valor 1 (ramo direito da árvore) quando a expressão é satisfeita e o valor 0 caso contrário. Pode ser observado que a árvore tem que repetir uma subárvore pela obrigatoriedade de regras mutuamente exclusivas. Este problema é chamado de *problema da subárvore repetida*⁴ [Pagallo 90]. Uma solução para este tipo de problema é permitir combinações booleanas durante as divisões dos nós. Em vez de dividir nós simples, a árvore pode também examinar divisões de combinações de atributos. O problema de encontrar a melhor combinação de atributos para cada nó é extremamente complexo, mas heurísticas podem ajudar a melhorar a performance de uma árvore permitindo a inserção de combinações de atributos na árvore. Um dos processos para solucionar este problema é através de indução construtiva (seção 4.5).

Um outro exemplo de replicação de subárvores pode ser encontrado em [Cendrowska 87], no qual é mostrado que a árvore de decisão mínima para aprender o conceito x definido como

Se $a = 3 \wedge b = 3$ então conclua classe x
 Se $c = 3 \wedge d = 3$ então conclua classe x

⁴Do inglês *replicated subtree problem*

Figura 4.9: Árvore de Decisão Correspondente à Expressão Lógica $(A \wedge B) \vee (C \wedge D)$

possui 10 nós não terminais e 21 nós folhas, assumindo que os atributos a , b , c e d podem assumir valores no conjunto $\{1, 2, 3\}$.

4.4 Indução de Regras de Decisão

Regras de decisão ou regras de produção são potencialmente mais poderosas que árvores de decisão. Embora todas as árvores de decisão possam ser expressas na forma de regras de decisão, o relaxamento nas restrições de exclusão mútua pode potencialmente levar as regras a cobrir as classes de forma mais eficiente e compacta.

Embora o relaxamento da exclusão mútua pareça ser uma pequena modificação, desenvolver um sistema de aprendizado com este tipo de linguagem de descrição se torna muito mais complexo. A maior dificuldade na construção de um sistema de aprendizado com conjunto de regras mutuamente não exclusivas está na falta de capacidade de se prever qual será o resultado da interação das regras, mesmo para os casos do conjunto de treinamento. Deve ser observado que em uma árvore as regras são gradualmente compostas através de particionamentos repetidos dos dados. A árvore cresce de forma gradual com um controle simples dos erros cometidos como resultado de cada particionamento. Em qualquer estágio da construção da árvore, cada caminho na árvore corresponde a uma regra. O número de erros para uma classe é simplesmente a soma dos erros para cada regra desta classe. Portanto as árvores operam em um ambiente altamente controlado durante a fase de aprendizado.

Certamente se existisse um ambiente semelhante a uma árvore de decisão, discutido anteriormente, onde fosse possível montar o conjunto de regras mutuamente não exclusivas, facilitaria em muito o trabalho de montar um sistema de aprendizado que

utiliza regras como linguagem de descrição de conceitos. Uma forma de resolver tal problema seria tentar encontrar algumas boas regras que cobrissem partes do problema e, posteriormente, agrupá-las na forma normal disjuntiva, ou seja, “agrupá-las” com o operador $\vee$.

Essa é a forma como a estratégia de separar-e-conquistar funciona, esta estratégia tem a sua origem na família de algoritmos de aprendizado de máquina AQ [Michalski 69] sob o nome de estratégia *covering*. O termo separar-e-conquistar foi primeiramente utilizado por [Pagallo 90] pela forma de desenvolver a teoria que caracteriza esta estratégia de aprendizado: aprender uma regra que cobre uma parte dos exemplos dados, remover os exemplos cobertos do conjunto de treinamento e, recursivamente aprender outra regra que cubra os exemplos restantes até que todos os exemplos de treinamento sejam processados.

Um exemplo de um algoritmo de aprendizado muito simples sob a estratégia separar-e-conquistar, ilustrado na Figura 4.10, é o seguinte: dado um conjunto de exemplos (exemplos positivos) e contra-exemplos (exemplos negativos) de um conceito, o algoritmo inicia com uma hipótese vazia, e adiciona a esta hipótese regras que cubram os exemplos positivos. O aprendizado de uma regra inicia com uma regra que seja a mais genérica possível, como por exemplo

Se *verdade* então conclua classe X.

Esta regra diz que a classe X é verdadeira para qualquer exemplo, seja ele positivo ou negativo. Esta regra é especializada retirando a condição *verdade* e adicionando novas condições à regra. Estas condições são geralmente testes sobre a presença de certos valores para vários atributos presentes nos exemplos do conjunto de treinamento. Para que se atinja o objetivo de encontrar regras que não cubram exemplos negativos, o algoritmo seleciona um teste que otimize a pureza da regra, isto é, um teste que maximize a porcentagem de exemplos positivos entre todos os exemplos cobertos. Quando é encontrada uma regra que cobre somente exemplos positivos, estes exemplos são removidos do conjunto de treinamento e outra regra é aprendida para cobrir os exemplos que ainda restam no conjunto de treinamento. Este algoritmo é repetido até que não restem mais exemplos positivos. Desta forma, ao chegar ao fim de sua execução, este algoritmo terá criado um hipótese que cobre todos os exemplos positivos (a hipótese é *completa*) e nenhum exemplo negativos (a hipótese é *consistente*).

Apesar de que os sistemas de aprendizado que utilizam a estratégia separar-e-conquistar compartilham uma estrutura básica semelhante ao algoritmo descrito anteriormente, alguns sistemas possuem algumas características adicionais. Pode-se analisar os sistemas de separar-e-conquistar segundo as seguintes dimensões

- **Bias⁵ de Linguagem:** são as linguagens de descrição (seção 4.2) utilizadas pelo

⁵Em [Mitchell 80] *bias* é definido como qualquer base para escolher uma generalização ao invés de

```

procedure Separar-e-Conquistar(Exemplos)
  Hipotese =  $\emptyset$ 
  while Positivos(Exemplos)  $\neq \emptyset$ 
    MelhorRegra =  $\{true\}$ 
    Regra = MelhorRegra
    while Negativos(Cobertos)  $\neq \emptyset$ 
      for Condicao  $\in$  Condicoes
        Refinamento = Regra  $\cup$  Condicao
        if Pureza(Refinamento, Exemplos) > Pureza(MelhorRegra, Exemplos)
          MelhorRegra = Refinamento
      Regra = MelhorRegra
    Hipotese = Hipotese  $\cup$  Regra
    Exemplos = Exemplos - Cobertos
  return(Hipotese)

```

Figura 4.10: Algoritmo Separar-e-Conquistar

algoritmo de aprendizado para descrever exemplos, hipótese e teoria de domínio;

- **Bias de Busca:** é a forma como o espaço de hipóteses é percorrido. Isto inclui o algoritmo de busca (hill-climbing, best-first search, etc.), a estratégia de busca (top-down e bottom-up) e a função heurística utilizada para avaliar as hipóteses encontradas;
- **Bias para Evitar Overfitting:** é a heurística utilizada para evitar overfitting em conjuntos de exemplos com ruído. Muitas vezes é preferível regras simples às regras complexas, uma vez que regras simples são provavelmente mais preditivas em novos exemplos.

4.4.1 Bias de Linguagem

Como já foi dito anteriormente, as linguagens de descrição são importantes pois determinam quais conceitos podem ser aprendidos por um sistema de aprendizado. Certos conceitos podem não ser expressáveis ou podem ter uma representação muito complicada em certas linguagens de descrição de hipóteses. Por outro lado, o espaço de busca para um algoritmo de aprendizado é definido pela linguagem de descrição de hipóteses e, quanto mais poderosa for esta linguagem, maior será o espaço de busca.

A grande maioria dos sistemas de aprendizado proposicionais utilizam conjuntos de regras na forma normal disjuntiva para expressar os conceitos aprendidos. Muitas vezes, esses sistemas são utilizados em problemas que envolvem apenas duas classes. Nesses casos, são aprendidas regras que classificam apenas uma classe e, quando um exemplo

outra que não seja estritamente consistente com os exemplos de treinamento observados.

não é coberto por nenhuma destas regras, este exemplo é classificado como pertencente à outra classe⁶. Muitos sistemas de aprendizado simbólico relacional utilizam esta estratégia para aprender conceitos que envolvem duas classes.

Para problemas que envolvem múltiplas classes é necessário impor uma ordem às regras pertencentes à hipótese induzida, uma vez que alguns exemplos podem ser cobertos por duas ou mais regras que realizam diferentes predições. Sendo assim, um exemplo pode ser classificado em diferentes classes dependendo da ordem das regras. Teorias que impõem uma ordem de avaliação em suas regras são freqüentemente chamadas de *listas de decisão*⁷ [Rivest 87].

Recentemente foram propostos sistemas de aprendizado capazes de aprender conceitos expressos em linguagem lógica de primeira ordem. Estes sistemas são capazes de aprender conceitos mais complexos que os conceitos expressáveis em linguagem proposicional e, também, são capazes de utilizar naturalmente teoria de domínio no processo de aprendizado. Esses sistemas são discutidos mais detalhadamente no Capítulo 5.

Existem ainda sistemas de aprendizado que procuram aprender conceitos capazes de prever sobre classes contínuas — problemas de regressão — esses sistemas geralmente utilizam *regras de regressão* para expressar suas hipóteses. Entretanto esses sistemas estão fora do escopo deste trabalho. Em [Fürnkranz 96] é realizada uma discussão detalhada a respeito de bias de linguagem e outras características dos métodos de indução de regras através de separar-e-conquistar.

4.4.2 Bias de Busca

O bias de busca pode ser analisado segundo três aspectos principais

- Algoritmo de Busca;
- Estratégia de Busca;
- Função Heurística;

discutidos a seguir.

Algoritmo de Busca Entre os algoritmos de busca utilizados por sistemas de aprendizado com a estratégia separar-e-conquistar, o mais simples é o algoritmo hill-climbing. Os algoritmos de separar-e-conquistar consistem de encontrar regras que cubram o

⁶A classe coberta pelas regras induzidas é freqüentemente chamada de classe positiva e, a outra classe de classe negativa. O processo de classificar como exemplo negativo um exemplo que não é coberto pelas regras é conhecido por negação por falha (*negation as failure*).

⁷Do inglês *decision lists*.

máximo de exemplos positivos sem cobrir exemplos negativos. Como já foi dito anteriormente, o processo pode, por exemplo, iniciar com uma regra genérica que cobre exemplos positivos e negativos. Através de especializações, esta regra pode ser restringida a cobrir somente exemplos positivos⁸. Desta forma, dada uma regra genérica, um conjunto de restrições pode ser aplicado a esta regra, dando origem a uma série de novas regras. Se estas novas regras ainda cobrirem exemplos negativos, novas restrições podem ser aplicadas até que se encontre uma regra que cubra somente exemplos positivos.

O algoritmo hill-climbing seleciona, de todas as regras geradas pela aplicação de restrições, apenas a melhor regra, avaliada por uma função heurística. Somente esta regra é especializada nas iterações seguintes do algoritmo com aplicações de novas restrições. O algoritmo hill-climbing possui deficiências pois é possível, durante o processo de construção de uma regra, separar uma regra que a princípio pareça ser uma boa regra, mas com a inserção de novas condições mostra-se ser, por exemplo, uma regra pouco preditiva. Como este algoritmo não mantém instâncias de outras regras que também estavam sendo construídas, não é possível desistir de uma decisão. Por este motivo, o algoritmo hill-climbing é dito ser *míope* uma vez que busca encontrar o ponto de máximo global através da seleção do melhor refinamento local.

O algoritmo beam search busca aliviar a miopia do método hill-climbing através da seleção de não apenas a melhor regra, mas também de outras k melhores regras. Este algoritmo mantém a eficiência do algoritmo hill-climbing (reduzida por um fator constante), entretanto este algoritmo é capaz de produzir melhores resultados, e por este motivo é muito utilizado em algoritmos do tipo separar-e-conquistar tais como AQ [Michalski 86] e CN2 [Clark 89].

Um outro algoritmo de busca que pode ser utilizado é conhecido como best-first search, no qual todas as regras geradas são utilizadas na próxima iteração. Este algoritmo pode ser entendido como se fosse o método beam search com $k = \infty$. O maior problema desta abordagem é o custo computacional.

Estratégia de Busca Entre as estratégias de busca, a mais comum em algoritmos separar-e-conquistar é a estratégia top-down. Esta estratégia foi utilizada no algoritmo de separar-e-conquistar descrito anteriormente. A busca inicia com a regra mais genérica possível e esta regra é repetidamente especializada através de, por exemplo, a inserção de condições na cláusula. A busca termina quando é encontrada uma regra que cubra somente exemplos positivos e não cubra exemplos negativos.

Uma outra estratégia é conhecida como bottom-up e não é frequentemente aplicada em algoritmos separar-e-conquistar. Nesta estratégia a busca parte de uma regra específica (tipicamente um exemplo do conjunto de treinamento) e são aplicados operadores

⁸Este processo é conhecido como busca top-down, é possível utilizar uma outra estratégia de busca conhecida como bottom-up, descrita a seguir.

de generalização com o intuito de aumentar o número de casos cobertos por esta regra. Um exemplo de sistema de aprendizado bottom-up é o SIA [Venturini 93], o qual utiliza algoritmos genéticos para encontrar generalizações de um conjunto selecionado de exemplos positivos. A primeira geração é formada por generalizações aleatórias destes exemplos, e as gerações seguintes são criadas através da aplicação aleatória de operadores de generalização e da troca aleatória de condições entre regras. Este processo é repetido até que a melhor regra se mantenha estável por algumas gerações.

Função Heurística Por fim, a função heurística é utilizada para avaliar a qualidade das regras geradas. Esta função guia o processo de busca levando o algoritmo de busca à região correta na árvore de busca. Existem muitas funções heurísticas que podem ser utilizadas em algoritmos do tipo separar-e-conquistar. As funções descritas a seguir utilizam as seguintes variáveis

- P : O número total de exemplos positivos
- N : O número total de exemplos negativos
- p : O número de exemplos positivos cobertos por uma regra r
- n : O número de exemplos negativos cobertos por uma regra r

Alguns exemplos de funções são

1. Precisão:

$$Pr(r) = \frac{p + (N - n)}{P + N}$$

Esta medida fornece a precisão de uma regra, r , computando a porcentagem de exemplos corretamente classificados, isto é, o número de exemplos positivos cobertos pela regra mais o número de exemplos negativos não cobertos. Uma vez que P e N são constantes, maximizar esta função significa maximizar $p - n$.

2. Pureza:

$$P(r) = \frac{p}{p + n}$$

Esta medida fornece a porcentagem de exemplos positivos calculada sobre o número total de exemplos cobertos pela regra. Esta medida possui valor ótimo quando nenhum exemplo negativo é coberto, entretanto não é dado um abono para regras que cobrem muitos exemplos positivos.

3. Estimador Laplace:

$$LAP(r) = \frac{p + 1}{p + n + 2}$$

Esta medida penaliza regras que cobrem poucos exemplos. Se uma regra cobre nenhum exemplo, esta medida fornece o valor $\frac{1}{2}$. Por outro lado quanto mais

casos forem cobertos, mais esta função converge para a medida *pureza* $(\frac{p}{p+n})$. Por causa da simplicidade dessa medida, ela é muito utilizada em sistemas separar-e-conquistar.

4.4.3 Bias para Evitar Overfitting

Assim como em árvores de decisão, os algoritmos que aprendem regras necessitam de um mecanismo de poda, uma vez que na presença de ruído, os algoritmos de aprendizado que encontram hipóteses completas e consistentes são obrigados a inserir novas restrições às regras aprendidas para que sejam excluídos exemplos negativos que, na realidade, são positivos. E, também, estes algoritmos têm que adicionar novas regras à hipótese para que exemplos positivos, que na realidade são exemplos negativos, sejam cobertos. Desta forma, hipóteses geradas a partir de conjuntos de exemplos com ruído podem ser muito complicadas e provavelmente apresentarão baixa precisão de classificação para novos exemplos.

Uma forma de resolver este problema é procurar aumentar a capacidade de predição considerando não somente hipóteses completas e consistentes, mas aproximações mais simples dessas hipóteses. Uma hipótese simples que cobre a maioria dos exemplos positivos e não cobre a maioria dos exemplos negativos é, provavelmente, muito mais preditiva que uma hipótese completa e consistente mas muito complexa.

Assim como em árvores de decisão, existem duas abordagens para evitar o overfitting, a primeira busca manter a hipótese simples durante o processo de indução e é conhecida como *pré-poda*. A segunda busca simplificar a hipótese após o final do processo de indução e é conhecida como *pós-poda*.

Entre as técnicas de pré-poda mais conhecidas está o *critério mínimo de pureza*. Segundo este critério uma regra pode ser adicionada à hipótese sempre que uma certa porcentagem de exemplos positivos cobertos por esta regra é atingida. Por exemplo, uma regra pode ser aceita se 80% dos exemplos cobertos por ela são positivos.

É possível simplificar uma hipótese após a sua construção, através de pós-poda. Uma das técnicas mais conhecidas de pós-poda procura remover condições ou até mesmo regras inteiras que sejam desnecessárias à hipótese. A idéia é verificar se a remoção de uma condição, ou mesmo de uma regra, leva a uma degradação da qualidade da hipótese, sendo que a qualidade da hipótese é freqüentemente medida através de sua taxa de erro.

Como já foi dito anteriormente, a indução de árvores de decisão é um processo comportado, uma vez que os exemplos são particionados em nós mutuamente exclusivos. Desta forma, a poda de um nó em uma árvore de decisão não influencia nos demais nós. Entretanto, a indução de regras de decisão não cria regras mutuamente exclusivas, isto é, o mesmo exemplo pode ser coberto por duas ou mais regras. Por este motivo,

a poda de regras de decisão apresenta um problema fundamental que foi apontado por [Fürnkranz 94]. Quando uma condição é podada de uma regra, esta regra é generalizada, passando a cobrir mais exemplos positivos e negativos. A Figura 4.11 ilustra um conjunto de três regras antes e depois da aplicação da pós-poda. Inicialmente a hipótese era composta por três regras sendo que cada regra cobria um conjunto de exemplos denominados de A , B e C . Após a poda das regras a primeira regra foi generalizada e passou a cobrir também todos os exemplos cobertos pela terceira regra (conjunto C) e parte dos exemplos cobertos pela segunda regra ($B1 \subset B$). Enquanto que a terceira regra pode ser simplesmente removida da hipótese, a segunda regra se tornou em um problema bem mais complicado. A segunda regra naturalmente cobre os exemplos do conjunto $B2$, uma vez que esta regra foi criada para cobrir o conjunto B o qual contém o conjunto $B2$. Entretanto, certamente uma outra regra seria mais apropriada para discriminar entre os exemplos do conjunto $B2$ e os exemplos negativos.

Figura 4.11: Pós-poda em um Conjunto de Regras de Decisão [Fürnkranz 96]

Uma forma de tentar contornar este problema é podar cada regra logo após a sua construção. Com isso, os exemplos cobertos pela regra podada podem ser retirados do conjunto de treinamento e não influenciarão na construção das regras seguintes.

Na próxima seção é discutida a construção indutiva, que é uma forma de tentar estender a linguagem de representação através da criação de novos atributos.

4.5 Indução Construtiva

O conjunto de treinamento fornecido a um sistema de aprendizado é composto por diversos atributos e por uma classe a que cada exemplo pertence. Se estes atributos possuírem um alto poder preditivo o sistema poderá classificar novos casos cometendo poucos erros. Mas se os atributos possuírem pouco poder preditivo (ruído), pouco o sistema poderá extrair dos exemplos.

Alguns atributos que não são representativos a princípio, isto é, são apenas ruído, podem se tornar muito preditivos se combinados com outros atributos, em um processo conhecido como indução construtiva ou aprendizado indutivo. Neste processo, os atri-

butos iniciais do conjunto de treinamento, chamados de *atributos primitivos* ou *features primitivas*, são convenientemente compostos uns com os outros de forma a criar novos atributos chamados de *atributos compostos* ou *features compostas*.

Existem basicamente duas formas de conduzir o processo de indução contrutiva. Em uma destas formas, o usuário é responsável por controlar e direcionar o processo de composição dos atributos. Em uma outra forma, todo o processo é automaticamente conduzido pelo sistema.

Para exemplificar o uso da indução construtiva é utilizado o exemplo dos robôs amigos e inimigos da Tabela 4.1 pg. 40. Esse exemplo consiste de casos de robôs cujas características são o formato da cabeça (atributo cabeça), do corpo (atributo corpo) e se o robô sorri ou não (atributo sorri?). Como foi mostrado na seção 4.3.1, os atributos cabeça e corpo possuem uma entropia maior que o atributo sorri?, para a divisão dos exemplos no nó raiz. Certamente, esses atributos possuem um valor preditivo menor que o do atributo sorri?. Mas é possível utilizar estes dois atributos para compor um novo atributo como mostra a Tabela 4.2. Este novo atributo é chamado de *mesma-forma* e assume o valor *sim* quando o corpo e a cabeça possuem a mesma forma e o valor *não* caso contrário.

Número Exemplo	Cabeça	Corpo	Sorri?	Mesma-Forma	Classe
1	quadrada	quadrado	sim	sim	amigo
2	quadrada	triangular	não	não	inimigo
3	redondar	redondo	sim	sim	amigo
4	triangula	redondo	não	não	inimigo
5	triangular	triangular	não	sim	amigo
6	redonda	quadrado	sim	não	inimigo

Tabela 4.2: Construção do Atributo Composto Mesma-Forma para o Exemplo dos Robôs Amigos de Inimigos

O atributo composto mesma-forma é muito mais preditivo que os atributos primitivos cabeça e corpo separados, uma vez que sempre que este atributo assume o valor *sim* o robô é amigo e quando assume o valor *não* o robô é inimigo. Esse atributo possui entropia 0 se utilizado para dividir os exemplos do nó raiz e, com a utilização desse atributo é possível criar uma árvore com somente um nó raiz e dois nós folhas, como mostra a Figura 4.12

Para criar o atributo composto mesma-forma foi utilizado o operador de igualdade entre atributos, já que o atributo composto mesma-forma possui o valor *sim* quando cabeça e corpo são iguais e o valor *não* quando são diferentes. É possível criar mais atributos compostos através da aplicação de outros operadores. Estretanto, o processo de construção de atributos é difícil devido ao tamanho do espaço definido pelos atributos candidatos. O número de atributos compostos potenciais que podem ser construídos é uma função combinatória do número de atributos que já existentes, multiplicado pelo número de operadores construtivos adotados. Em [Nicoletti 94] é discutida a

Figura 4.12: O Atributo Composto Mesma-Forma como Raiz da Árvore de Decisão

complexidade deste processo e alguns algoritmos para a criação automática de atributos compostos são propostos.

Apesar do processo de indução construtiva ampliar alguns limites do aprendizado proposicional, a linguagem atributo-valor ainda é restrita aos conhecimentos que podem ser aprendidos. Como foi dito, o poder de expressão de uma linguagem atributo-valor equivale a uma linguagem lógica proposicional, desta forma, somente os conceitos que podem ser expressos em lógica proposicional podem ser aprendidos por um sistema proposicional. Esta forte restrição não permite representar objetos estruturados nem relações entre objetos e seus componentes. Desta forma, para algumas aplicações que requerem um poder maior de expressão que a linguagem atributo-valor oferece, pode-se utilizar uma linguagem de descrição como a lógica de primeira ordem, como será discutido no Capítulo 5.

4.6 Considerações Finais

O poder de expressão de um conjunto de regras de decisão é superior ao poder das árvores de decisão, pois as regras de decisão ampliam alguns limites impostos pelas árvores. Isto faz com que os sistemas de aprendizado que empregam regras de decisão sejam potencialmente superiores aos sistemas que utilizam árvores. Como já foi dito, quanto mais compacta e eficiente é a solução, melhor será o desempenho do sistema para novos casos. O maior poder de expressão das regras pode levar a soluções mais compactas e a um menor número de regras e atributos. Entretanto, até agora não foi comprovado que um método seja superior ao outro.

Neste capítulo foi apresentado o paradigma simbólico proposicional de aprendizado de máquina. Sistemas de aprendizado simbólico têm como uma das principais características o tipo de linguagem de descrição utilizada para descrever os exemplos, conceitos e conhecimento de fundo.

Os sistemas de aprendizado simbólico que utilizam a linguagem atributo-valor para descrever os exemplos a serem aprendidos são conhecidos como sistemas de aprendizado simbólico proposicional. Estes sistemas freqüentemente utilizam árvores ou regras de

decisão para representar os conceitos ou hipóteses induzidas⁹. O poder de expressão de um conjunto de regras de decisão é superior ao poder das árvores de decisão, pois as regras de decisão ampliam alguns limites impostos pelas árvores. Isto faz com que os sistemas de aprendizado que empregam regras de decisão sejam potencialmente superiores aos sistemas que utilizam árvores. Como já foi dito, quanto mais compacta e eficiente é a solução, melhor será o desempenho do sistema para novos casos. O maior poder de expressão das regras pode levar a soluções mais compactas e a um menor número de regras e atributos, por exemplo. Entretanto, até agora não foi comprovado que um método seja superior ao outro. Os sistemas de aprendizado simbólico proposicional são limitados quanto aos conceitos que podem ser aprendidos, pois a linguagem atributo-valor possui um poder representacional muito restrito. Uma tentativa de ampliar os limites da linguagem atributo-valor é conhecido como aprendizado construtivo, e procura criar novos atributos através da composição de atributos primitivos.

O próximo capítulo aborda a Programação Lógica Indutiva. Esta abordagem de aprendizado utiliza a lógica de primeira ordem como linguagem de descrição de exemplos, hipóteses e conhecimento de fundo. Através da utilização da lógica de primeira ordem como linguagem de descrição é possível superar algumas limitações dos sistemas de aprendizado proposicionais, entretanto este aumento na expressividade faz também com que o aprendizado se torne mais complexo.

⁹Implementações Prolog de diversos algoritmos de Aprendizado de Máquina simbólico proposicionais podem ser encontrados em [Monard 97a].

Capítulo 5

Programação Lógica Indutiva

5.1 Considerações Iniciais

Programação Lógica Indutiva (PLI) tem sido definida como sendo a intersecção de aprendizado indutivo e programação lógica, uma vez que PLI emprega técnicas de ambas as áreas.

Do aprendizado de máquina indutivo, PLI herda os objetivos, ou seja, desenvolver ferramentas e técnicas para induzir hipóteses a partir de exemplos e sintetizar novos conhecimentos a partir de experiência [Muggleton 94, Lavrač 95]. Através do uso da lógica de primeira ordem como linguagem de descrição de instâncias, hipóteses e conhecimento de fundo, PLI pode superar as duas principais limitações dos métodos tradicionais de aprendizado de máquina simbólico

1. O uso de linguagens de descrição limitadas — comparáveis à lógica proposicional.
2. Dificuldades do uso de conhecimento de fundo no processo de aprendizado.

A primeira limitação é importante pois muitos problemas não podem ser expressos em lógica proposicional, mas podem ser expressos em lógica de primeira ordem, ou alguma variação da lógica de primeira ordem. Um domínio onde este problema se torna óbvio é a síntese de programas lógicos a partir de exemplos. O uso de conhecimento de fundo é também crucial uma vez que, na maioria dos casos, ele é considerado essencial para a obtenção de um comportamento inteligente.

Da lógica computacional, PLI herda não somente seu formalismo representacional, mas também sua orientação semântica, e várias técnicas bem estabelecidas. Em contraste com outras abordagens de aprendizado indutivo, Programação Lógica Indutiva está

interessada nas propriedades das regras de inferência, na convergência de algoritmos, e na complexidade computacional dos procedimentos.

Programação Lógica Indutiva estende a teoria e prática da lógica computacional através da investigação da indução, ao invés da dedução, como modo básico de inferência. Da mesma forma que a teoria da lógica computacional descreve inferências dedutivas a partir de fórmulas lógicas, PLI estuda a inferência indutiva de programas lógicos a partir de exemplos e conhecimento de fundo.

5.2 Conceitos Básicos

Nesta seção são enfocados alguns conceitos básicos de Lógica de Primeira Ordem. Uma vez que se está preocupado em aplicar estes conceitos lógicos à Programação Lógica Indutiva, este trabalho se restringe à discussão de uma notação específica da Lógica de Primeira Ordem: a Notação Clausal. A Notação Clausal é um formalismo especialmente apropriado à Programação Lógica.

Neste texto os conceitos da Lógica Matemática são abordados de uma forma mais informal, utilizando exemplos para apresentar as principais definições. Estas definições provêm de textos tradicionais de Programação Lógica, tal como [Casanova 87].

Raciocínios lógicos são realizados diariamente. Por exemplo, dada a seguinte declaração

A: Todo mundo que gosta de Inteligência Artificial, gosta de lógica.

e também

B: Maria gosta de Inteligência Artificial.

Intuitivamente, pode-se concluir que

C: Maria gosta de Lógica.

Declarações como estas em linguagem natural podem ser convertidas em sentenças lógicas. Para isso, é necessário definir uma sintaxe, a qual consiste de um alfabeto, ou seja, os símbolos válidos, e um conjunto de regras sintáticas. Estas regras sintáticas definem quais são as sentenças válidas formadas a partir dos símbolos presentes no alfabeto.

Através da semântica é possível associar um “significado” às sentenças lógicas. No caso da Lógica de Primeira Ordem com Notação Clausal, o significado de uma sentença é

definido através da especificação das condições em que esta sentença é associada aos valores verdade **verdadeiro** ou **falso**.

No exemplo anterior é ilustrado um raciocínio intuitivo no qual se conclui que C é verdade dados A e B . Em Lógica é possível realizar “raciocínios” semelhantes, por exemplo é possível verificar através de Lógica de Primeira Ordem que C é *consequência lógica* de A e B , ou $A \wedge B \models C$.

Prova automática de teoremas especifica como podem ser obtidas novas sentenças a partir de outras conhecidas verdadeiras, através de manipulações simbólicas.

As seções seguintes discutem respectivamente a sintaxe, semântica e a prova automática de teoremas para a Lógica de Primeira Ordem com Notação Clausal.

5.2.1 Sintaxe da Notação Clausal

A Lógica de Primeira Ordem com Notação Clausal é especialmente útil para resolver problemas que envolvem objetos e relações. Por exemplo, a declaração

gosta(maria, ia)

pode ser lida como “Maria gosta de IA”. Nomes de objetos como *maria* e *ia* são chamados de *constantes*, neste texto é utilizada a convenção originária da linguagem de programação lógica Prolog de escrever constantes como números, palavras simples iniciando com um caracter minúsculo, ou na forma de uma sequência de palavras entre aspas simples (por exemplo ‘Isto é uma constante’).

Além de relações entre objetos, a Lógica de Primeira Ordem com Notação Clausal permite fazer referência a grupos arbitrários de objetos, como por exemplo

gosta(X, ia)

Esta declaração pode ser lida como “Todo mundo gosta de IA”. Objetos arbitrários são denotados por *variáveis*, na convenção utilizada, variáveis são palavras simples iniciadas por um caracter maiúsculo.

Objetos estruturados, ou simplesmente *estruturas*, são objetos que possuem vários componentes. Sendo que os próprios componentes podem ser estruturas (na realidade, os componentes de uma estrutura são *termos*, termo é definido logo em seguida). Por exemplo, uma data pode ser vista como uma estrutura com três componentes: dia, mês e ano. Embora seja composta por vários componentes, uma estrutura é tratada como um objeto simples. Para que os componentes da estrutura sejam compostos em um único objeto, deve ser escolhido um nome para a estrutura, este nome é chamado de

funtor ou *símbolo funcional*. Para o exemplo da data, um possível funtor é a palavra *data*, desta forma a data 1 de Janeiro de 1980 pode ser escrita da seguinte forma:

data(1, janeiro, 1980)

Constantes, variáveis e estruturas são objetos afins, e são chamados de *termos*. Desta forma, termos são definidos da seguinte maneira

- Uma constante é um termo.
- Uma variável é um termo.
- Uma estrutura é um termo, sendo que os componentes da estrutura também são termos.

Segundo esta definição, são exemplos de termos

X
maria
data(1, janeiro, 1980)
data(dia(1), mes(janeiro), ano(1980))
data(D, M, 1980)

São chamados de *termos ground* aqueles termos que não possuem variáveis, do exemplo anterior são termos ground apenas

maria
data(1, janeiro, 1980)
data(dia(1), mes(janeiro), ano(1980))

Relações entre objetos são denotadas por *predicados* ou *símbolos predicativos*, e é chamado de *átomo* a composição do *predicado* com os objetos envolvidos na relação. Um exemplo de átomo é

gosta(maria, ia)

no qual *gosta* é o predicado e *maria* e *ia* são os objetos envolvidos na relação. Em um átomo, os objetos são termos (constantes, variáveis ou estruturas) comumente chamados de *argumentos*. O número de argumentos define a *aridade* de um predicado. Para o exemplo anterior, o átomo *gosta* possui dois argumentos *maria* e *ia*, sendo assim um átomo de aridade 2. Por fim, um átomo ground é um átomo que não possui variáveis.

Um literal é um átomo ou a negação de um átomo. No alfabeto utilizado a negação é representada pelo símbolo “ $\neg$ ”. Um *literal positivo* é simplesmente um átomo, e um *literal negativo* é a negação de um átomo. São exemplos de literais

$$\begin{array}{ll} \text{gosta}(\text{maria}, \text{ia}) & \text{(Literal positivo)} \\ \neg \text{gosta}(\text{maria}, \text{ia}) & \text{(Literal negativo)} \end{array}$$

Átomos podem ser compostos através de símbolos com significados especiais presentes no alfabeto da linguagem lógica, a estas composições dá-se o nome de *fórmula*. Uma fórmula é definida recursivamente da seguinte forma

1. Um átomo é uma fórmula.
2. Se F e G são fórmulas, então $(\neg F)$, $(F;G)$, (F,G) , $(F \leftarrow G)$ também são fórmulas. Na qual “ $;$ ” representa uma disjunção ($\vee$), “ $,$ ” representa uma conjunção ($\wedge$) e “ $\leftarrow$ ” representa uma implicação (se G então F).
3. Fórmulas são geradas somente com a aplicação finita de 1 e 2.

Algumas fórmulas definem relações condicionais entre objetos, estas fórmulas são chamadas de *cláusulas* e possuem o seguinte formato

$$A_1; A_2; \dots; A_n \leftarrow B_1, B_2, \dots, B_m$$

na qual A_i e B_i são átomos. São exemplos de cláusulas

$$\begin{array}{l} \text{gosta}(\text{maria}, \text{logica}) \leftarrow \text{gosta}(\text{maria}, \text{ia}) \\ \text{gosta}(\text{X}, \text{logica}) \leftarrow \text{gosta}(\text{X}, \text{ia}) \\ \text{casado}(\text{X}); \text{solteiro}(\text{X}) \leftarrow \text{homem}(\text{X}), \text{adulto}(\text{X}) \end{array}$$

A disjunção de átomos $A_1; A_2; \dots; A_n$ é chamada de *cabeça* da cláusula e a conjunção de átomos $B_1, B_2, \dots, B_m$ é chamada de *corpo* da cláusula.

Dependendo do número de literais presentes na cabeça ou no corpo de uma cláusula, esta cláusula recebe nomes especiais.

Uma *cláusula definida* é uma cláusula que possui somente um literal na cabeça, ou seja

$$A \leftarrow B_1, \dots, B_m$$

Uma *cláusula unitária* ou *fato* é uma cláusula definida com um corpo vazio, ou seja

$$A \leftarrow$$

Neste trabalho será adotada a notação simplificada

$$A$$

Uma *cláusula objetivo* é uma cláusula que possui cabeça vazia, ou seja

$$\leftarrow B_1, \dots, B_m$$

Por fim uma *cláusula vazia* é uma cláusula que possui cabeça e corpo vazios e é representada pelo símbolo $\square$.

5.2.2 Semântica da Lógica Clausal

O *Universo de Herbrand*, H , de um conjunto de cláusulas, S , é o conjunto de termos ground ocorrendo em S . Por exemplo, para o seguinte conjunto de cláusulas

$$\begin{aligned} \text{gosta}(X, \text{logica}) &\leftarrow \text{gosta}(X, \text{ia}) \\ \text{gosta}(\text{maria}, \text{ia}) \end{aligned}$$

Tem-se o Universo de Herbrand

$$H = \{\text{logica}, \text{ia}, \text{maria}\}$$

Caso o conjunto de cláusulas possua uma estrutura, então deve-se encontrar todos os termos ground possíveis de serem contruídos a partir das estrutura e constantes presentes no conjunto de cláusulas. Se for inserido o funtor *aluno* nas cláusulas do exemplo anterior, tem-se o seguinte conjunto de cláusulas

$$\begin{aligned} \text{gosta}(\text{aluno}(X), \text{logica}) &\leftarrow \text{gosta}(\text{aluno}(X), \text{ia}) \\ \text{gosta}(\text{aluno}(\text{maria}), \text{ia}) \end{aligned}$$

Segundo estas cláusulas, tem-se o seguinte Universo de Herbrand

$$H = \{\text{logica}, \text{ia}, \text{maria}, \text{aluno}(\text{logica}), \text{aluno}(\text{ia}), \text{aluno}(\text{maria}), \text{aluno}(\text{aluno}(\text{logica})), \text{aluno}(\text{aluno}(\text{ia})), \dots\}$$

Como mostra este exemplo, a existência de estruturas em um conjunto de cláusulas faz com que o Universo de Herbrand se torne infinito, na realidade, três situações podem ocorrer na criação do Universo de Herbrand de um conjunto de cláusulas S

1. Se S possui constantes, mas não possui estruturas, então o Universo de Herbrand será o conjunto finito das constantes que ocorrem em S .
2. Se S possui constantes e estruturas, então o Universo de Herbrand será infinito e contável.
3. Se S não possui constantes então escolhe-se uma constante arbitrariamente, por exemplo a . Dependendo da presença ou não de estruturas em S , o Universo de Herbrand será infinito ou finito, respectivamente.

Uma definição mais formal para Universo de Herbrand, pode ser dada em termos de *conjuntos de constantes de nível i* .

Seja S um conjunto de cláusulas e H_0 o conjunto de constantes que ocorrem em S

- Se nenhuma constante ocorre em S então defina $H_0 = \{a\}$.
- O conjunto de *constantes de nível i* , H_i , é definido como sendo $H_i = H_{i-1} \cup T$, para $i \in \mathbb{N}$.

Onde T é o conjunto de todas as estruturas $f(t_1, \dots, t_n)$ ocorrendo em S e, t_j , $j = 1, \dots, n$ são os elementos de H_{i-1} .

O Universo de Herbrand é definido como sendo o maior conjunto de constantes de nível i , ou seja, H_∞ .

A *Base de Herbrand* de um conjunto de cláusulas, S , é o conjunto de átomos ground que podem ser construídos utilizando os predicados que ocorrem em S e os termos ground do Universo de Herbrand de S .

Voltando ao exemplo, as cláusulas

$$\begin{array}{l} \text{gosta}(X, \text{logica}) \leftarrow \text{gosta}(X, \text{ia}) \\ \text{gosta}(\text{maria}, \text{ia}) \end{array}$$

tem como Universo de Herbrand

$$H = \{\text{logica}, \text{ia}, \text{maria}\}$$

e Base de Herbrand

$$B_S = \{\text{gosta}(\text{logica}, \text{logica}), \text{gosta}(\text{ia}, \text{logica}), \text{gosta}(\text{maria}, \text{logica}), \text{gosta}(\text{logica}, \text{ia}), \\ \text{gosta}(\text{ia}, \text{ia}), \text{gosta}(\text{maria}, \text{ia})\}$$

A Base de Herbrand de um conjunto de cláusulas representa tudo o que pode ser dito a respeito dos indivíduos (constantes) que ocorrem no conjunto de cláusulas.

Uma *Interpretação de Herbrand*, ou simplesmente *Interpretação*, para um conjunto de cláusulas S é um mapeamento da Base de Herbrand de S no conjunto de valores verdade $\{\mathbf{verdadeiro}, \mathbf{falso}\}$. Por exemplo, o mapeamento

$$\begin{aligned} &\{\text{gosta}(\text{logica}, \text{logica}) \rightarrow \mathbf{verdadeiro}, \\ &\text{gosta}(\text{ia}, \text{logica}) \rightarrow \mathbf{falso}, \\ &\text{gosta}(\text{maria}, \text{logica}) \rightarrow \mathbf{verdadeiro}, \\ &\text{gosta}(\text{logica}, \text{ia}) \rightarrow \mathbf{verdadeiro}, \\ &\text{gosta}(\text{ia}, \text{ia}) \rightarrow \mathbf{falso}, \\ &\text{gosta}(\text{maria}, \text{ia}) \rightarrow \mathbf{falso}\} \end{aligned}$$

é uma interpretação para o exemplo anterior. Pode-se abreviar este mapeamento listando somente os elementos da Base de Herbrand associados ao valor **verdadeiro**. Desta forma, a Interpretação de Herbrand anterior pode ser escrita como

$$\{\text{gosta}(\text{logica}, \text{logica}), \text{gosta}(\text{maria}, \text{logica}), \text{gosta}(\text{logica}, \text{ia})\}$$

Uma vez que uma Interpretação de Herbrand associa valores verdade aos elementos da Base de Herbrand de um conjunto de cláusulas, então a Interpretação também associa valores verdade para cada cláusula do conjunto. Para avaliar o valor verdade de uma cláusula em uma dada interpretação, deve-se ter em mente que a cabeça de uma cláusula é composta pela disjunção de átomos e o corpo por uma conjunção de átomos. Conseqüentemente, a cabeça de uma cláusula é **verdadeira** se ao menos um átomo for **verdadeiro**, e o corpo da cláusula é **verdadeiro** se todos os átomos forem **verdadeiros**. Por sua vez, o valor verdade de uma cláusula é determinado pelo valor verdade de sua cabeça e corpo. Existem quatro possibilidades

- O corpo é **verdadeiro** e a cabeça é **verdadeira** então a cláusula é **verdadeira**;
- O corpo é **verdadeiro** e a cabeça é **falsa** então a cláusula é **falsa**;
- O corpo é **falso** e a cabeça é **verdadeira** então a cláusula é **verdadeira**;
- O corpo é **falso** e a cabeça é **falsa** então a cláusula é **falsa**.

Se uma cláusula é verdadeira sob uma determinada interpretação diz-se então que a interpretação é um *modelo* para a cláusula. Da mesma forma, uma interpretação é um modelo para um conjunto de cláusulas se todas as cláusulas deste conjunto são verdadeiras sob esta interpretação.

Para verificar se uma cláusula é verdadeira ou não, é também necessário lidar com as variáveis presentes na cláusula. Para tanto realizam-se substituições as quais são mapeamentos de variáveis para termos. Por exemplo a substituição

$$\{\text{maria}/X\}$$

quando aplicada à cláusula, C ,

$$\text{gosta}(X, \text{logica}) \leftarrow \text{gosta}(X, \text{ia})$$

resulta na cláusula

$$\text{gosta}(\text{maria}, \text{logica}) \leftarrow \text{gosta}(\text{maria}, \text{ia})$$

Neste exemplo, a cláusula resultante é ground, por isso esta cláusula é chamada de *instância ground* de C e a substituição $\{\text{maria}/X\}$ é chamada de *substituição ground*.

Uma interpretação é um modelo para uma cláusula não ground se esta é um modelo para todas as instâncias ground da cláusula, sendo que estas instâncias ground são construídas através de substituições entre as variáveis e os elementos da Base de Herbrand. Como a Base de Herbrand possui apenas átomos ground as substituições são todas ground.

Por exemplo, seja um conjunto de cláusulas S

$$\begin{aligned} \text{gosta}(X, \text{logica}) &\leftarrow \text{gosta}(X, \text{ia}) \\ \text{gosta}(\text{maria}, \text{ia}) & \end{aligned}$$

possui o seguinte Universo de Herbrand

$$H = \{\text{logica}, \text{ia}, \text{maria}\}$$

e a seguinte Base de Herbrand

$$B_S = \{\text{gosta}(\text{logica}, \text{logica}), \text{gosta}(\text{ia}, \text{logica}), \text{gosta}(\text{maria}, \text{logica}), \text{gosta}(\text{logica}, \text{ia}), \\ \text{gosta}(\text{ia}, \text{ia}), \text{gosta}(\text{maria}, \text{ia})\}$$

As instâncias ground de S são

$$\begin{aligned} \text{gosta}(\text{logica}, \text{logica}) &\leftarrow \text{gosta}(\text{logica}, \text{ia}) & (\{\text{logica}/X\}) \\ \text{gosta}(\text{ia}, \text{logica}) &\leftarrow \text{gosta}(\text{ia}, \text{ia}) & (\{\text{ia}/X\}) \\ \text{gosta}(\text{maria}, \text{logica}) &\leftarrow \text{gosta}(\text{maria}, \text{ia}) & (\{\text{maria}/X\}) \\ \text{gosta}(\text{maria}, \text{ia}) & \end{aligned}$$

Por fim, os modelos de S são

$\{\text{gosta}(\text{maria}, \text{ia}), \text{gosta}(\text{maria}, \text{logica})\}$
 $\{\text{gosta}(\text{maria}, \text{ia}), \text{gosta}(\text{maria}, \text{logica}), \text{gosta}(\text{ia}, \text{logica})\}$
 $\{\text{gosta}(\text{maria}, \text{ia}), \text{gosta}(\text{maria}, \text{logica}), \text{gosta}(\text{logica}, \text{logica})\}$
 $\{\text{gosta}(\text{maria}, \text{ia}), \text{gosta}(\text{maria}, \text{logica}), \text{gosta}(\text{ia}, \text{ia}), \text{gosta}(\text{ia}, \text{logica})\}$
 $\{\text{gosta}(\text{maria}, \text{ia}), \text{gosta}(\text{maria}, \text{logica}), \text{gosta}(\text{logica}, \text{ia}), \text{gosta}(\text{logica}, \text{logica})\}$

Pode ser verificado que as cláusulas $\text{gosta}(\text{maria}, \text{ia})$ e $\text{gosta}(\text{maria}, \text{logica})$ estão presentes em todos os modelos, diz-se então que estas cláusulas são *consequência lógica* de S . Em geral, uma cláusula C é consequência lógica de S se todo modelo do conjunto de cláusulas também é um modelo de C , e escreve-se $S \models C$.

5.2.3 Prova por Resolução

Teoria de prova fornece uma forma mais eficiente de verificar se uma cláusula, C , é consequência lógica de um conjunto de cláusulas S ($S \models C$). Para tanto são utilizadas regras de inferência, as quais são operações meramente sintáticas. Se C pode ser derivada de S através de aplicações de regras de inferência, então diz-se que C pode ser provada a partir de S . Para lógica clausal existe uma regra de inferência muito poderosa chamada de *resolução*.

Uma versão da resolução para cláusulas definidas é

$$\frac{A \leftarrow B_1, \dots, B_j, \dots, B_n \quad B_j \leftarrow D_1, \dots, D_k}{A \leftarrow B_1, \dots, B_{j-1}, D_1, \dots, D_k, B_{j+1}, \dots, B_n}$$

Por exemplo dadas as seguintes cláusulas

$\text{gosta}(\text{maria}, \text{joao}) \leftarrow \text{amigo}(\text{maria}, \text{jose}), \text{amigo}(\text{jose}, \text{joao})$
 $\text{amigo}(\text{maria}, \text{jose}) \leftarrow \text{amigo}(\text{jose}, \text{carlos})$

Com uma aplicação de resolução é possível provar que

$\text{gosta}(\text{maria}, \text{joao}) \leftarrow \text{gosta}(\text{jose}, \text{carlos}), \text{gosta}(\text{jose}, \text{joao})$

Quando as cláusulas possuem variáveis, é necessário substituir as variáveis, em um processo conhecido como *unificação*. Por exemplo, dadas as cláusulas

$\text{gosta}(\text{X}, \text{logica}) \leftarrow \text{gosta}(\text{X}, \text{ia})$
 $\text{gosta}(\text{maria}, \text{ia})$

É possível transformar o átomo $\text{gosta}(\text{X}, \text{ia})$ em $\text{gosta}(\text{maria}, \text{ia})$ se for aplicada a substituição $\{\text{maria}/\text{X}\}$ à primeira cláusula, resultando

```
gosta(maria, logica) ← gosta(maria, ia)
gosta(maria, ia)
```

Aplicando resolução fica provado que

```
gosta(maria, logica)
```

5.3 Especificação do Problema de PLI

De uma forma geral, PLI pode ser descrita a partir de uma teoria de conhecimento de fundo inicial K a alguma evidência $E = E^+ \cup E^-$ (exemplos). E^+ representa as evidências positivas e E^- as evidências negativas. O objetivo está em induzir a hipótese H que junto com K *explica* os exemplos E . No caso geral, K , E e H podem ser qualquer conjunto de cláusulas. Entretanto, geralmente a teoria de conhecimento de fundo, as evidências e as hipóteses têm que satisfazer algumas restrições sintáticas, que são chamadas de *bias* B . Bias define o espaço de fórmulas bem formadas usadas para representar hipóteses e pode ser considerada como parte do conhecimento de fundo.

Programação lógica indutiva pode ser definida como segue

Dado:

- um conjunto de exemplos E ;
- uma teoria de conhecimento de fundo K ;
- um bias de linguagem B que define o espaço de hipóteses;

Encontrar:

- uma hipótese H que satisfaz o bias da linguagem e explica os exemplos E com respeito à teoria de conhecimento de fundo K .

Na qual H é dito explicar E com respeito a K se e somente se

$$K \wedge H \models E^+ \quad (\text{completude})$$

e

$$K \wedge H \wedge E^- \not\models \square \quad (\text{consistência})$$

Na maioria dos trabalhos em PLI, H e K consistem de cláusulas definidas e E consiste de fatos verdadeiros e falsos, isto é, de exemplos positivos e negativos.

A Figura 5.1 mostra graficamente os conceitos de completude e consistência de uma hipótese. Esta figura utiliza a função cobre, a qual pode ser definida da seguinte forma

$$\begin{aligned} \text{cobre}(H, e) &= \text{verdade} && \text{se } e \text{ é coberto por } H \\ \text{cobre}(H, e) &= \text{falso} && \text{caso contrário} \end{aligned}$$

no qual H é a hipótese e e um exemplo positivo ou negativo.

A função cobre pode ser redefinida para conjuntos de exemplos da seguinte forma

$$\text{cobre}(H, E) = \{e \in E \mid \text{cobre}(H, e) = \text{verdade}\}$$

Dependendo de como a hipótese¹ cobre os exemplos positivos e negativos, quatro situações podem acontecer

1. H completa e consistente, cobre todos os exemplos positivos e nenhum exemplo negativo;
2. H incompleta e consistente, não cobre todos os exemplos positivos e não cobre exemplos negativos;
3. H completa e inconsistente, cobre todos os exemplos positivos e cobre alguns exemplos negativos;
4. H incompleta e inconsistente, não cobre todos os exemplos positivos e cobre alguns exemplos negativos.

5.4 PLI como um Problema de Busca

Da mesma forma que a maioria dos problemas em Inteligência Artificial, PLI pode ser visto com um problema de busca no qual o espaço de busca é formado pelas soluções candidatas. Para solucionar este problema pode-se tentar utilizar um método de força bruta no qual todas as soluções candidatas são geradas e avaliadas. Entretanto este método é computacionalmente muito caro para ser aplicado na prática. Desta forma é necessário estruturar o espaço de busca de forma que ramos infrutíferos da árvore de busca possam ser podados.

¹A função cobre pode ser redefinida como $\text{cobre}(H, K, E) = \text{cobre}(H \cup K, E)$ de maneira a considerar a teoria de domínio.

H : completa, consistente

H : incompleta, consistente

H : completa, inconsistente

H : incompleta, inconsistente

Figura 5.1: Completeza e Consistência de uma Hipótese [Lavrač 94]

PLI utiliza os conceitos de generalização e especialização para estruturar o espaço de busca. Uma hipótese H_1 é mais geral que uma hipótese H_2 se e somente se $H_1 \models H_2$. H_2 é dita ser mais especializada que H_1 .

Deve ser observado que generalização e especialização permitem podar parte do espaço de busca pois

- Se $K \wedge H_1$ não cobrem algum exemplo positivo, ou seja $K \wedge H_1 \not\models p$, onde p é uma evidência positiva ($p \in E^+$), então nenhuma das especializações de H_1 serão capazes de cobrir este exemplo. Desta forma, as especializações de H_1 podem ser podadas do espaço de busca.
- Se $K \wedge H_2$ cobrem algum exemplo negativo, ou seja $K \wedge H_2 \models n$, onde n é uma evidência negativa ($n \in E^-$), então todas as generalizações de H_2 também serão inconsistentes e podem ser podadas do espaço de busca.

A princípio, a relação de generalização entre cláusulas definida através de $\models$ parece ser bem apropriada a PLI. Mas, infelizmente, esta definição de semântica generalizada introduz dois problemas [Flach 94]. O primeiro é que não parece haver um procedimento simples para generalizar cláusulas, e o segundo problema é que as generalizações menos gerais sob consequência lógica não são sempre únicas.

Plotkin desenvolveu uma relação alternativa de generalidade chamada de $\theta - subsoma$. Esta relação fornece um procedimento muito simples para a determinação de cláusulas genéricas e determina também generalizações menos gerais únicas.

5.5 A Relação $\theta - subsoma$

O relação $\theta - subsoma$ foi proposta em [Plotkin 70]. Segundo Plotkin seu trabalho foi

... originado com uma sugestão de R.J. Popplestone, uma vez que unificação é útil em dedução automática pelo método de resolução, o seu oposto pode ser útil para indução. O oposto do unificador mais geral de dois literais é chamado de generalização menos geral.

A generalização menos geral de dois termos foi pensada por Plotkin como o oposto do unificador mais geral, por isso a generalização menos geral também é chamada de *anti-unificação*.

Para definir a generalização menos geral, foi antes definida uma relação de generalidade, a $\theta - subsoma$. No contexto da definição de $\theta - subsoma$, uma cláusula é tratada como um conjunto de literais relacionados entre si via disjunção.

Por definição, uma cláusula C_1 θ – *subsoma* uma cláusula C_2 se e somente se existir uma substituição θ tal que $C_1\theta \subseteq C_2$.

A relação θ – *subsoma* é uma relação de generalidade, uma vez que

- Se $C_1 \theta$ – *subsoma* C_2 então a cláusula C_1 é pelo menos tão geral quanto a cláusula C_2 .
- Se $C_1 \theta$ – *subsoma* C_2 e C_2 não θ – *subsoma* C_1 então C_1 é mais geral que C_2 . E pode-se dizer que C_1 é uma generalização de C_2 e que C_2 é uma especialização de C_1 .

O exemplo a seguir mostra duas cláusulas, C_1 e C_2 , e verifica se C_1 é uma generalização de C_2 sob a relação θ – *subsoma*.

$$\begin{aligned} C_1: & \text{cena}(X) \leftarrow \text{sobre}(X,Y,Z), \text{circulo}(Y) \\ C_2: & \text{cena}(\text{quadro1}) \leftarrow \text{sobre}(\text{quadro1}, \text{anel}, E), \text{circulo}(\text{anel}), \text{plano}(E) \end{aligned}$$

podem ser traduzidas para conjuntos de literais relacionados entre si via disjunção². Estes conjuntos são chamados de SC_1 e SC_2

$$\begin{aligned} SC_1: & \{\text{cena}(X), \neg \text{sobre}(X,Y,Z), \neg \text{circulo}(Y)\} \text{ e} \\ SC_2: & \{\text{cena}(\text{quadro1}), \neg \text{sobre}(\text{quadro1}, \text{anel}, E), \neg \text{circulo}(\text{anel}), \neg \text{plano}(E)\} \end{aligned}$$

Para verificar se $C_1 \theta$ – *subsoma* C_2 deve-se encontrar uma substituição, se houver alguma, que faça com que o conjunto formado com os literais da cláusula C_1 seja um subconjunto do conjunto formado pelos literais da cláusula C_2 . Para tanto, este exemplo analisa cada literal do conjunto SC_1 e verifica se estes literais estão presentes no conjunto SC_2 . O literal $\text{cena}(X)$ de SC_1 está contido no conjunto SC_2 caso for realizada a substituição $\{\text{quadro1}/X\}$. Desta forma esta substituição é aplicada a SC_1 , resultando na cláusula

$$\{\text{cena}(\text{quadro}), \neg \text{sobre}(\text{quadro1}, Y, Z), \neg \text{circulo}(Y)\}$$

O literal $\text{sobre}(\text{quadro1}, Y, Z)$ está contido no conjunto SC_2 caso for realizada as substituições $\{\text{anel}/Y, E/Z\}$. Desta forma aplica-se esta substituição resultando na cláusula

$$\{\text{cena}(\text{quadro}), \neg \text{sobre}(\text{quadro1}, \text{anel}, E), \neg \text{circulo}(Y)\}$$

²Observar que: $\text{cena}(X) \leftarrow \text{sobre}(X,Y,Z), \text{circulo}(Y) \equiv \text{cena}(X) \vee \neg \text{sobre}(X,Y,Z) \vee \neg \text{circulo}(Y)$ e $\text{cena}(\text{quadro1}) \leftarrow \text{sobre}(\text{quadro1}, \text{anel}, E), \text{circulo}(\text{anel}), \text{plano}(E) \equiv \text{cena}(\text{quadro1}) \vee \neg \text{sobre}(\text{quadro1}, \text{anel}, E) \vee \neg \text{circulo}(\text{anel}) \vee \neg \text{plano}(E)$.

No geral foram realizadas as seguintes substituições até o momento $\{\text{quadro1}/X, \text{anel}/Y, E/Z\}$. Resta ainda o seguinte literal a ser analisado

$$SC_1: \{\neg \text{circulo}(\text{anel})\}$$

o qual já faz parte do conjunto SC_2 .

Por fim, pode-se dizer que $C_1 \theta - \text{subsoma } C_2$ com as substituições $\theta = \{\text{quadro1}/X, \text{anel}/Y, E/Z\}$

Da mesma forma que pode-se verificar se $C_1 \theta - \text{subsoma } C_2$ é também possível verificar se $C_1 \theta - \text{subsoma}$ outras duas cláusulas C_2 e C_3 . Por exemplo, dadas as seguintes cláusulas

$$\begin{aligned} &\text{gosta}(\text{aluno}(\text{maria}), \text{professor}(\text{joao})) \\ &\text{gosta}(\text{aluno}(\text{joana}), \text{professor}(\text{jose})) \end{aligned}$$

As seguintes cláusulas $\theta - \text{subsoma}$ as cláusulas anteriores

$$\begin{aligned} G_1: &\text{gosta}(\text{aluno}(X), \text{professor}(Y)) && \{\text{maria}/X, \text{joao}/Y\} \\ G_2: &\text{gosta}(X, \text{professor}(Y)) && \{\text{aluno}(\text{maria})/X, \text{joao}/Y\} \\ G_3: &\text{gosta}(\text{aluno}(X), Y) && \{\text{maria}/X, \text{professor}(\text{joao})/Y\} \\ G_4: &\text{gosta}(X, Y) && \{\text{aluno}(\text{maria})/X, \text{professor}(\text{joao})/Y\} \end{aligned}$$

Estas cláusulas possuem os seguintes significados em linguagem natural, respectivamente

“Todo aluno gosta de todos os professores.”
 “Todo mundo gosta de todos os professores.”
 “Todo aluno gosta de todo mundo.”
 “Todo mundo gosta de todo mundo.”

Como pode ser observado as declarações anteriores vão gradativamente se tornando mais genéricas, sendo que a primeira declaração é a menos genérica e a última é a mais genérica. A primeira generalização é chamada de *generalização menos geral* ou simplesmente *lgg*³ sob $\theta - \text{subsoma}$.

Apesar de existir um conjunto de cláusulas que $\theta - \text{subsoma}$ outras duas cláusulas, existe apenas uma cláusula que é a menor generalização geral sob $\theta - \text{subsoma}$ [Muggleton 94].

³Do Inglês *Least General Generalization*, decidiu-se manter a sigla em Inglês pois esta é universalmente conhecida.

Pode-se definir a menor generalização geral de duas cláusulas C_2 e C_3 ($lgg(C_2, C_3)$) como sendo uma cláusula C_1 a qual $\theta - subsoma$ C_2 e C_3 tal que quaisquer outras cláusulas que $\theta - subsoma$ C_2 e C_3 também $\theta - subsoma$ C_1 .

Através do exemplo anterior pode-se verificar que G_1 é a lgg, pois

$$\begin{aligned} G_2 &\theta - subsoma G_1, \text{ com a substituição } \theta = \{\text{aluno}(X)/X\} \\ G_3 &\theta - subsoma G_1, \text{ com a substituição } \theta = \{\text{professor}(Y)/Y\} \\ G_4 &\theta - subsoma G_1, \text{ com a substituição } \theta = \{\text{aluno}(X)/X, \text{professor}(Y)/Y\} \end{aligned}$$

Por fim, as seguintes propriedades são válidas para a operação $\theta - subsoma$

1. Se $C_1 \theta - subsoma C_2$ então $C_1 \models C_2$, o inverso nem sempre é verdade.
2. A relação $\theta - subsoma$ de duas cláusulas induz a um reticulado, ou seja, para quaisquer duas cláusulas existe um *menor limite superior* (generalização menos geral) e um *maior limite inferior* e ambos são únicos.

Nesta seção foi apresentada a relação de generalização $\theta - subsoma$ e a menor generalização geral sob esta relação. Na próxima seção é discutida a indução por generalização, sendo que esta forma de indução está baseada na construção de rlgs (uma extensão das lggs).

5.6 Indução por Generalização

A menor generalização geral (lgg) de duas cláusulas C_1 e C_2 é a generalização menos geral comparada as outras generalizações destas cláusulas. Isto é importante, uma vez que forma uma base para generalizações cuidadosas de um algoritmo. Estas generalizações cuidadosas assumem que se C_1 e C_2 são verdade, então é muito provável que a $lgg(C_1, C_2)$ seja também verdade [Nicoletti 93].

Desta forma, pode-se definir esta operação da seguinte forma: a menor generalização geral de duas cláusulas C_1 e C_2 — $lgg(C_1, C_2)$ — é o menor limite superior do reticulado de cláusulas induzido pela operação $\theta - subsoma$. Para exemplificar, a Figura 5.2 mostra o reticulado para as cláusulas

gosta(aluno(maria), professor(joao))
gosta(aluno(joana), professor(jose))

do exemplo anterior.

Uma característica que faz com que as lgg sob $\theta - subsoma$ sejam especialmente adequadas para PLI, é porque elas podem ser calculadas facilmente. A seguir são apresentados

Figura 5.2: Um Exemplo de Reticulado.

três algoritmos para calcular o lgg de termos, literais e cláusulas [Lavrač 94]. O Algoritmo para calcular a lgg de termos é mostrado na Figura 5.3.

```

Procedure lggtermos( $T_1$ : Termo;  $T_2$ : Termo)
  if  $T_1 = T_2$ 
  then  $T_1$ 
  else
    if  $T_1 = f(u_1, \dots, u_k)$  and  $T_2 = f(v_1, \dots, v_k)$ 
    then  $f(\text{lggtermos}(u_1, v_1), \dots, \text{lggtermos}(u_k, v_k))$ 
    else  $V$ , na qual a mesma variável  $V$  é usada
      sempre como a lgg destes termos
  
```

Figura 5.3: Algoritmo para Calcular a lgg de Termos

Exemplos de lgg de alguns termos são mostrados na Tabela 5.1.

Nos dois últimos exemplos da Tabela 5.1 são utilizadas listas, as quais podem ser consideradas termos, uma vez que estas podem ser escritas utilizando o funtor “.”. Desta forma esses exemplos podem ser entendidos como

$$\begin{aligned} \text{lgg}(. (1, . (2, . (3, . ()))) , . (1, . (4, . (5, . ())))) &= . (1, . (A, . (B, . ()))) \\ \text{lgg}(. (r, . (s, . (t, . (u, . ())))) , . (r, . (s, . (t, . ())))) &= . (r, . (A, . (B, C))) \end{aligned}$$

A listagem do algoritmo responsável por calcular lgg de literais é apresentada na Figura 5.4.

T_1	T_2	$\text{lgg}(T_1, T_2)$
$f(r, s, t, u)$	$f(v, x, y, z)$	$f(A, B, C, D)$
$f(r, s, t, s)$	$f(v, x, x, x)$	$f(A, B, C, B)$
$f(r, s, t, u)$	$g(r, s, t, u)$	A
$[1, 2, 3]$	$[1, 4, 5]$	$[1, A, B]$
$[r, s, t, u]$	$[r, t, s]$	$[r, A, B \mid C]$

Tabela 5.1: Exemplos de lgg de Termos

Procedure $\text{lggliterais}(L_1: \text{Literal}; L_2: \text{Literal})$
if $L_1 = \text{sinal } p(t_1, \dots, t_n)$ **and** $L_2 = \text{sinal } p(u_1, \dots, u_n)$
then $\text{sinal } p(\text{lggtermos}(t_1, u_1), \dots, \text{lggtermos}(t_n, u_n))$
else indefinido

Figura 5.4: Algoritmo para Calcular lgg de Literais

Exemplos de lgg de alguns literais são mostrados na Tabela 5.2.

L_1	L_2	$\text{lgg}(L_1, L_2)$
$p(r, s)$	$p(v, x)$	$p(A, B)$
$\neg p(r, s)$	$p(r, s)$	indefinido
$\text{append}([1,2], [3,4], [1,2,3,4])$	$\text{append}([r], [s], [r,s])$	$\text{append}([A B], [C D], [A, E F])$

Tabela 5.2: Exemplos de lgg de Literais

O algoritmo para encontrar o lgg de duas cláusulas é um pouco diferente dos demais algoritmos, pode-se entender o algoritmo da seguinte forma

- Dadas duas cláusulas C_1 e C_2 entendidas na forma de conjuntos de literais, nos quais os literais se relacionam através de disjunções;
- $\text{lgg}(C_1, C_2)$ é representado pelo conjunto de literais, U , formado da seguinte forma
 1. É calculado o lgg de cada literal L_i do conjunto C_1 com cada literal K_j do conjunto C_2 ;
 2. Se $\text{lgg}(L_i, K_j)$ não é indefinido então o literal resultante desta operação é inserido no conjunto de resposta U .

Por fim, a Figura 5.5 apresenta a listagem do algoritmo responsável por calcular lgg de cláusulas.

Para ilustrar o funcionamento desses algoritmos, considere as seguintes cláusulas

C_1 : $\text{pai}(\text{jose}, \text{maria}) \leftarrow \text{progenitor}(\text{jose}, \text{maria}), \text{homem}(\text{jose}), \text{mulher}(\text{maria}).$
 C_2 : $\text{pai}(\text{joao}, \text{carlos}) \leftarrow \text{progenitor}(\text{joao}, \text{carlos}), \text{homem}(\text{joao}), \text{homem}(\text{carlos}).$

```

Procedure lggclausulas( $C_1$ : Clausula;  $C_2$ : Clausula)
  if  $C_1 = \{L_1, \dots, L_n\}$  and  $C_2 = \{K_1, \dots, K_m\}$ 
  then  $\{U_i \mid U_i = \text{lggliterais}(L_j, K_k) \text{ para } 1 \leq j \leq n \text{ and } 1 \leq k \leq m \text{ and}$ 
     $\text{lgg}(L_j, K_k) \neq \text{indefinido}\}$ 

```

Figura 5.5: Algoritmo para Calcular lgg de Cláusulas

Estas duas cláusulas podem ser transformadas nos seguintes conjuntos de literais relacionados entre si via disjunção

$C_1: \{\text{pai}(\text{jose}, \text{maria}), \neg \text{progenitor}(\text{jose}, \text{maria}), \neg \text{homem}(\text{jose}), \neg \text{mulher}(\text{maria})\}$
 $C_2: \{\text{pai}(\text{joao}, \text{carlos}), \neg \text{progenitor}(\text{joao}, \text{carlos}), \neg \text{homem}(\text{joao}), \neg \text{homem}(\text{carlos})\}$

Todos os literais presentes na cláusula C_1 serão comparados com todos os literais da cláusula C_2 . Por simplicidade, são mostrados a seguir somente os pares de literais que não resultam em indefinições

$\{\text{pai}(\text{jose}, \text{maria}), \text{pai}(\text{joao}, \text{carlos})$
 $\neg \text{progenitor}(\text{jose}, \text{maria}), \neg \text{progenitor}(\text{joao}, \text{carlos})$
 $\neg \text{homem}(\text{jose}), \neg \text{homem}(\text{joao})$
 $\neg \text{homem}(\text{jose}), \neg \text{homem}(\text{carlos})\}$

O conjunto de literais resultantes das operações de lgg, U , é

$U: \{\text{pai}(A, B), \neg \text{progenitor}(A, B), \neg \text{homem}(A), \neg \text{homem}(C)\}$

Este conjunto pode, finalmente, ser escrito como a seguinte cláusula

$\text{pai}(A,B) \leftarrow \text{progenitor}(A, B), \text{homem}(A), \text{homem}(C)$

Segundo [Sammut 94], a noção básica de lgg permitiu que fosse possível realizar indução em um conjunto de cláusulas providas como dados. Entretanto, lgg não leva em consideração uma teoria de domínio. Plotkin imaginava que um robô podia ver, manipular e tocar coisas. Ele podia explorar seu ambiente e realizar generalizações a respeito dos objetos que ele encontrava. Ele assumiu que o robô podia começar com alguma teoria sobre o mundo que seria estendida através de experiência. Desta forma, qualquer generalização feita pelo robô teria que ser relativa à sua teoria corrente. Em [Plotkin 71] o conceito de lgg foi estendido para a *menor generalização geral relativa* (rlgg)⁴.

⁴Do Inglês Relative Least General Generalization, mais uma vez mantém-se a sigla original por ser universalmente conhecida.

A rlgg de dois exemplos, e_1 e e_2 relativa a uma teoria de domínio K , pode ser definida da seguinte forma

- Calcule o modelo de Herbrand para as cláusulas pertencentes a teoria de domínio K , chamada $M(K)$;
- Construa duas cláusulas $e_1 \leftarrow C(M(K))$ e $e_2 \leftarrow C(M(K))$, onde $C(M(K))$ é a conjunção dos literais de $M(K)$.

$$\text{rlgg}(e_1, e_2) = \text{lgg}(e_1 \leftarrow C(M(K)), e_2 \leftarrow C(M(K)))$$

Tipicamente, os resultados obtidos através da operação de lgg são cláusulas com muitos literais, sendo muitos deles redundantes, entretanto alguns poucos literais são úteis. É importante lembrar que modelos de Herbrand de uma teoria de domínio não são sempre finitos. Se a teoria possui estruturas, por exemplo, o modelo pode ser infinito, e desta forma a rlgg também pode ser infinita.

Para resolver estes problemas, foi proposto por Muggleton e Feng [Muggleton 90] os seguintes passos

- Substituir o modelo de Herbrand da teoria de domínio K por um subconjunto desta teoria chamadado de modelo *h-easy*;
- Remover os literais redundantes da rlgg computada.

Os modelos h-easy podem ser entendidos como subconjuntos do modelo de Herbrand de um conjunto de cláusulas. Dado um conjunto de cláusulas, o modelo 1-easy é o conjunto de todos os fatos que podem ser derivados destas cláusulas com a aplicação de apenas uma resolução. O modelo 2-easy com a aplicação de duas resoluções e assim por diante. Uma discussão mais profunda a respeito do modelo h-easy está fora do escopo deste trabalho, mas pode ser encontrada em [Nicoletti 94].

Para ilustrar a indução por generalização através de rlggs, o próximo exemplo utiliza como teoria de domínio somente fatos. Aliás, esta é uma restrição comum nos sistemas de PLI mais conhecidos, como o FOIL [Quinlan 93] e o GOLEM [Feng 90].

Este exemplo utiliza a seguinte teoria de domínio [Flach 94]

```
append([1,2], [3,4], [1,2,3,4])
append([ ], [ ], [ ])
append([a], [ ], [a])
append([2], [3,4], [2,3,4])
```

Deseja-se encontrar a rlgg a partir dos seguintes exemplos

```

append([1,2], [3,4], [1,2,3,4])
append([a], [ ], [a])

```

Desta forma deve-se encontrar a lgg das seguintes cláusulas

```

append([1,2], [3,4], [1,2,3,4]) ← append([1,2], [3,4], [1,2,3,4]), append([a], [ ], [a]),
    append([ ], [ ], [ ]), append([2], [3,4], [2,3,4])
append([a], [ ], [a]) ← append([1,2], [3,4], [1,2,3,4]), append([a], [ ], [a]),
    append([ ], [ ], [ ]), append([2], [3,4], [2,3,4])

```

E resulta na cláusula

```

append([A|B], C, [A|D]) ←
    append([1,2], [3,4], [1,2,3,4]), append([A|B], C, [A|D]),
    append(W,C,X), append([S|B], [3,4], [S,T,U|V]),
    append([R|G], K, [R|L]), append([a], [ ], [a]),
    append(Q, [ ], Q), append([P], K, [P|K]), append(N,K,O),
    append(M, [ ], M), append([ ], [ ], [ ]), append(G, K, L),
    append([F|G], [3,4], [F,H,I|J]), append([E], C, [E|C]),
    append(B, C, D), append([2], [3,4], [2,3,4])

```

Como dito anteriormente, uma rlgg possui muitos literais redundantes que podem ser removidos. Inicialmente podem ser retirados todos os literais que são fatos, uma vez que estes literais não mudam o significado da cláusula pois eles são conhecidamente verdadeiros. Outra simplificação é retirar todos os literais que introduzem novas variáveis, ou seja, variáveis que não estão presentes na cabeça da cláusula. Cláusulas deste tipo são chamadas de cláusulas *restringidas*. Com estas simplificações, a cláusula fica restrita a

```

append([A|B], C, [A|D]) ← append([A|B], C, [A|D]), append(B, C, D)

```

O primeiro literal faz com que a cláusula se torne uma tautologia, desta forma decidiu-se retirar também este literal. Por fim a cláusula induzida é

```

append([A|B], C, [A|D]) ← append(B, C, D)

```

a qual é uma cláusula interessante pois define que uma lista de cabeça A e cauda B é concatenada com uma lista C resultando em uma lista com cabeça A e cauda D , se a concatenação da cauda B com a lista C resultar em D .

Nesta seção foram discutidos alguns dos conceitos básicos para a indução por generalização. Esta abordagem de indução busca gerar a hipótese partindo da teoria de

domínio e dos exemplos. Uma outra abordagem, chamada de *indução por especialização*, busca induzir a hipótese a partir de uma hipótese geral e da aplicação de restrições a esta hipótese. Esta abordagem de indução é objeto de discussão da próxima seção.

5.7 Indução por Especialização

Indução através de técnicas de especialização busca induzir novos conceitos partindo de uma hipótese o mais geral possível e aplicando restrições a esta hipótese. A princípio, tal hipótese é capaz de cobrir exemplos negativos e positivos, e as sucessivas restrições à hipótese buscam tornar esta hipótese consistente, ou seja, incapaz de cobrir exemplos negativos. Sob a ótica de PLI, a hipótese é composta de cláusulas, e as restrições à hipótese são especializações destas cláusulas. Sob θ – *subsoma* existem duas formas de especializar uma cláusula

- Aplicar uma substituição às variáveis da cláusula.
- Adicionar um literal ao corpo da cláusula.

Escolher qual literal deve ser incluído no corpo da cláusula e/ou qual substituição deve ser aplicada não é uma tarefa trivial, uma vez que muitas escolhas são possíveis. Isto torna o problema de especialização em uma busca na qual a árvore de busca possui um fator de ramificação muito grande. Entre outras soluções pode-se utilizar alguma medida de teoria da informação para guiar este processo de busca, transformando-o em uma busca heurística. Entre os sistemas que utilizam busca heurística para solucionar o problema de especialização está o sistema FOIL [Quinlan 93]. Nesta seção são utilizadas algumas das soluções propostas neste sistema para mostrar o funcionamento de uma indução através de especialização.

Para exemplificar o funcionamento de uma indução por especialização, suponha a árvore genealógica da Figura 5.6 [Cameron-Jones 93].

Como teoria de domínio tem-se a relação progenitor. Através da Figura 5.6 pode-se gerar os seguintes fatos, onde a relação *progenitor*(*A*, *B*) significa “A possui progenitor B”.

progenitor(ann, ian)	progenitor(ann, jane)
progenitor(jane, jan)	progenitor(jane, sean)
progenitor(john, alan)	progenitor(john, joan)
progenitor(sean, alan)	progenitor(sean, joan)

Como principal objetivo, busca-se encontrar uma definição para a relação *antepassado*(*A*, *B*), a qual significa “A possui antepassado B”. Através da Figura 5.6 é possível

Figura 5.6: Árvore Genealógica

gerar os seguintes exemplos positivos

antepassado(ann, alan)	antepassado(ann, ian)
antepassado(ann, jan)	antepassado(ann, jane)
antepassado(ann, joan)	antepassado(ann, sean)
antepassado(jane, alan)	antepassado(jane, jan)
antepassado(jane, joan)	antepassado(jane, sean)
antepassado(john, alan)	antepassado(john, joan)
antepassado(sean, alan)	antepassado(sean, joan)

e os seguintes exemplos negativos

antepassado(alan, alan)	antepassado(alan, ann)
antepassado(alan, ian)	antepassado(alan, jan)
antepassado(alan, jane)	antepassado(alan, joan)
antepassado(alan, john)	antepassado(alan, sean)
antepassado(ann, ann)	antepassado(ann, john)
...	
antepassado(sean, ann)	antepassado(sean, ian)
antepassado(sean, jan)	antepassado(sean, jane)
antepassado(sean, john)	antepassado(sean, sean)

O processo de indução por generalização inicia selecionando uma cláusula mais geral possível. Para a relação antepassado, a cláusula mais geral é

antepassado(A, B)

a qual significa que todo mundo é antepassado de todo mundo. O processo prossegue através de inclusões de novos literais ao corpo da cláusula. Estes literais são construídos

através dos predicados já conhecidos⁵. Consultando a teoria de domínio, verifica-se que existem apenas dois predicados conhecidos: *progenitor* e *antepassado*. Estes predicados são compostos com variáveis para formar os literais que irão compor o corpo da cláusula, novas variáveis podem ser inseridas, com a restrição de que pelo menos uma variável tenha ocorrido na cabeça ou em outros literais da cláusula. Por hora, os literais que possuem o predicado *antepassado* são excluídos, pois a terminação não pode ser garantida.

Além dos predicados conhecidos, podem ser utilizadas algumas relações pré-construídas, tais como igualdade e desigualdade para comparações de cadeias de caracteres e maior, menor, etc. para comparações de valores numéricos. Portanto, os candidatos a corpo da cláusula são

$A = B$	$A \neq B$
$\text{progenitor}(A, A)$	$\neg \text{progenitor}(A, A)$
$\text{progenitor}(A, B)$	$\neg \text{progenitor}(A, B)$
$\text{progenitor}(A, C)$	$\neg \text{progenitor}(A, C)$
$\text{progenitor}(B, A)$	$\neg \text{progenitor}(B, A)$
$\text{progenitor}(B, B)$	$\neg \text{progenitor}(B, B)$
$\text{progenitor}(B, C)$	$\neg \text{progenitor}(B, C)$
$\text{progenitor}(C, A)$	$\neg \text{progenitor}(C, A)$
$\text{progenitor}(C, B)$	$\neg \text{progenitor}(C, B)$

Para escolher qual literal deve compor o corpo da cláusula, é necessário utilizar alguma função heurística. O sistema FOIL utiliza a heurística *greedy* que é explicada logo a diante. Por hora suponha que a função heurística selecionou o literal $\text{progenitor}(A, B)$ como o melhor literal, formando a cláusula

$$\text{antepassado}(A, B) \leftarrow \text{progenitor}(A, B)$$

Esta cláusula não cobre exemplos negativos, desta forma ela não precisa ser especializada. O processo de indução pode adicionar esta cláusula à hipótese. Os exemplos positivos cobertos por esta cláusula são retirados do conjunto de treinamento. Mas como esta cláusula não cobre todos os exemplos positivos, o processo de indução deve continuar a procurar por outras cláusulas que cubram os exemplos positivos restantes. Os exemplos positivos restantes são

$\text{antepassado}(\text{ann}, \text{alan})$	$\text{antepassado}(\text{ann}, \text{ian})$
$\text{antepassado}(\text{ann}, \text{joan})$	$\text{antepassado}(\text{ann}, \text{sean})$
$\text{antepassado}(\text{jane}, \text{alan})$	$\text{antepassado}(\text{jane}, \text{joan})$

⁵Outros predicados não conhecidos poderiam ser considerados se o sistema tivesse a capacidade de inventar novos predicados

O processo de indução novamente inicia com a cláusula mais genérica possível. E a função *greedy* seleciona como primeiro literal da segunda cláusula o literal

$$\text{progenitor}(A, C)$$

A inclusão deste literal faz com que uma nova variável, C , seja incluída na cláusula. Com isto, os exemplos positivos e negativos são rescritos com a inclusão desta nova variável. Segundo o novo literal, $\text{progenitor}(A, C)$, a variável C , é o *progenitor* de A , como nesta aplicação cada exemplo possui dois progenitores (pai e mãe), cada exemplo dá origem a dois outros exemplos. No caso do exemplo

$$\text{antepassado}(\text{ann}, \text{alan})$$

dá origem aos seguintes exemplos, representados por tuplas

$$(\text{ann}, \text{alan}, \text{ian}) \quad (\text{ann}, \text{alan}, \text{jane})$$

onde *ian* e *jane* são os progenitores de *ann*. Os novos exemplos positivos de treinamento, representados por tuplas, são

(ann, alan, ian)	(ann, alan, jane)
(ann, jan, ian)	(ann, jan, jane)
(ann, joan, ian)	(ann, joan, jane)
(ann, sean, ian)	(ann, sean, jane)
(jane, alan, jan)	(jane, alan, sean)
(jane, joan, jane)	(jane, joan, ian)

e os novos exemplos negativos são

(ann, ann, ian)	(ann, ann, jane)
(ann, john, ian)	(ann, john, jane)
...	
(sean, sean, allan)	(sean, sean, joan)

A busca continua por um literal que possa cobrir os exemplos positivos. Como agora está-se procurando por um literal que não é o primeiro literal do corpo da cláusula, pode-se incluir o predicado *antepassado*. Portanto os literais que podem ser incluídos na cláusula são todas as composições possíveis de literais positivos e negativos com os predicados *progenitor* e *antepassado* e as variáveis A , B , C e uma nova variável D .

O segundo literal a ser incluído, segundo a função heurística *greedy*, é $\text{antepassado}(C, B)$, formando a cláusula

$$\text{antepassado}(A, B) \leftarrow \text{progenitor}(A, C), \text{antepassado}(C, B)$$

Esta cláusula não cobre exemplos negativos, desta forma ela é adicionada à hipótese atual. Os exemplos positivos cobertos por esta cláusula são retirados do conjunto de treinamento. Como a última cláusula cobre todos os exemplos positivos, então o conjunto de treinamento não possui mais exemplos positivos e o processo de indução termina como a seguinte hipótese induzida

$$\begin{aligned} \text{antepassado}(A, B) &\leftarrow \text{antepassado}(A, B) \\ \text{antepassado}(A, B) &\leftarrow \text{progenitor}(A, C), \text{antepassado}(C, B) \end{aligned}$$

A função heurística greedy utilizada neste exemplo é definida em termos dos novos exemplos positivos e negativos cobertos antes e depois da inclusão de um literal. Por exemplo, na indução da segunda cláusula, o conjunto de treinamento inicialmente possuía 6 exemplos positivos e 50 exemplos negativos. Depois da adição do primeiro literal, os exemplos positivos passaram a ser 12 e os negativos 36. Definindo p' como o número de exemplos positivos que foram estendidos, p_0 e p_1 como sendo o número de exemplo positivos antes e depois da inclusão do literal (no exemplo: 6 e 12), e n_0 e n_1 como o número de exemplos negativos antes e depois da inclusão do literal (no exemplo: 50 e 36), a função greedy é definida como

$$\text{greedy} = p' \left(\log_2 \left(\frac{p_1 + 1}{n_1 + p_1 + 1} \right) - \log_2 \left(\frac{p_0 + 1}{n_0 + p_0 + 1} \right) \right)$$

a qual dá resultado 6.67 para este literal.

5.8 Considerações Finais

Neste capítulo foram apresentados alguns dos principais conceitos de Programação Lógica Indutiva. A abordagem PLI busca, através do uso de predicados da lógica de primeira ordem, superar algumas limitações apresentadas por métodos de aprendizado simbólico tradicionais que utilizam a linguagem atributo-valor para descrever os exemplos de treinamento.

Através do uso da lógica de primeira ordem como linguagem de descrição de exemplos, hipóteses e conhecimento de fundo, PLI não somente supera os limites impostos por linguagens de descrição mais restritas, como também supera muitas dificuldades do uso de conhecimento de fundo no processo de aprendizado.

Entretando, o aumento da capacidade de representação das linguagens de descrição gera algumas dificuldades. O problema de aprendizado de máquina pode ser visto como um problema de busca, onde o espaço de busca é limitado pela capacidade representacional das linguagens de descrição utilizadas. Quando é utilizada uma linguagem de descrição

mais poderosa, o espaço de busca se torna maior e, em alguns casos, pode superar o poder computacional disponível. Nestes casos é necessário investigar alguns métodos para restringir o espaço de busca.

Capítulo 6

O Ambiente Computacional AMPSAM

6.1 Considerações Iniciais

Nos últimos anos, a área de aprendizado indutivo tem tido uma notável evolução. Pesquisadores desta área têm proposto inúmeros novos sistemas de aprendizado e também melhoramentos aos sistemas já existentes. Este contexto fez com que surgisse uma necessidade de comparar os novos sistemas de aprendizado indutivo e os melhoramentos propostos aos sistemas já existentes. Por outro lado, é sabido que nenhum sistema de aprendizado é ótimo para todos os domínios de aplicação. Desta forma, dada uma aplicação é necessário avaliar os sistemas de aprendizado existentes a fim de determinar qual destes melhor se adequa ao problema que está sendo tratado.

Embora um sistema de aprendizado possa ser avaliado sob diversos critérios (seção 3.7), a taxa de erro verdadeira é a forma mais utilizada pois fornece um índice a respeito da correta classificação de novos casos, o principal objetivo de um sistema de aprendizado indutivo.

Como já foi mencionado, em problemas reais, medir diretamente a taxa de erro verdadeira de um sistema de aprendizado é uma tarefa inviável devido ao tamanho restrito dos conjuntos de exemplos. Desta forma, para avaliar a performance de um sistema de aprendizado de forma não tendenciosa, deve-se utilizar algum método capaz de estimar a taxa de erro verdadeira com precisão, como por exemplo os métodos estatísticos de resampling.

Os métodos de resampling são assim chamados pois consistem de múltiplas seções de particionamento dos dados em amostra de treinamento e teste. Nestes casos, o sistema de aprendizado aprende com a amostra de treinamento e a medição dos erros cometidos

pelo classificador é realizada utilizando a amostra de teste. O número de seções de treinamento e teste varia conforme o método empregado, geralmente este número está entre uma dezena (10-fold cross validation) até algumas centenas no caso do bootstrap (por volta de 200 iterações) ou o leaving-one-out (número de iterações é igual ao número de casos do conjunto de exemplos).

Avaliações experimentais desempenham um papel de extrema importância em Aprendizado de Máquina Indutivo, uma vez que, até hoje, não existe um tratamento formal completo dos métodos utilizados. Isto é, para um certo problema de aprendizado de máquina não existem resultados formais para decidir qual método é ótimo [Kibler 88]. Ainda que existam na literatura modelos teóricos, como aqueles fornecidos pela teoria computacional de aprendizado, que podem ajudar a decidir qual sistema de aprendizado é mais apropriado a uma determinada aplicação, a última palavra é sempre dada por avaliações experimentais [Flexer 96].

Desta forma, devido à importância das avaliações experimentais e por estas avaliações tratarem de uma tarefa lenta e repetitiva, foi proposto criar um ambiente de avaliação de performance de sistemas de aprendizado indutivo, chamado de AMPSAM— Ambiente Computacional para Medir a Performance de Sistemas de Aprendizado de Máquina. Este ambiente, capaz de automatizar grande parte do processo de avaliação é objeto de discussão deste capítulo.

6.2 O Projeto do Ambiente AMPSAM

Como já foi dito anteriormente, o principal objetivo do ambiente AMPSAM é automatizar, em parte, o processo de avaliação de um sistema de aprendizado de máquina indutivo. Este ambiente integra alguns métodos estatísticos de resampling e os sistemas de aprendizado simbólico indutivo mais conhecidos. Para tanto, o ambiente deve apresentar algumas flexibilidades, entre elas

- O ambiente deve ser facilmente extensível, ou seja, deve ser facilmente modificado para que incorpore novos sistemas de aprendizado que venham a ser criados;
- O ambiente deve ser extensível também a novos métodos estatísticos que venham a ser incorporados;
- O ambiente deve interagir com os principais sistemas de aprendizado sem que seja necessário modificar o código fonte destes sistemas, o que tornaria o projeto inviável.

Para atingir esses objetivos, foram tomadas algumas decisões de projeto. Decidiu-se criar o ambiente não como um simples programa, mas como um conjunto de ferramentas. Essas ferramentas interagem entre si através de UNIX C-shell scripts, mas podem

ser utilizadas separadamente para resolver problemas específicos. As ferramentas podem ser divididas em dois grupos

- Ferramentas para particionamento de exemplos segundo diversos métodos de resampling;
- Ferramentas para manipulação sintática de conjuntos de exemplos.

As ferramentas para particionamento de exemplos consistem de programas com funções tais como embaralhar os exemplos, dividi-los em duas ou mais partições mutuamente exclusivas, e unir essas partições em conjuntos de treinamento e teste.

A decisão de projeto de criar algumas ferramentas e uní-las através de shell scripts permite que novos métodos de resampling sejam incorporados ao ambiente mais facilmente. Dependendo das características do novo método a ser incorporado, as ferramentas já existentes, como por exemplo a ferramenta para embaralhamento ou mesmo a ferramenta para particionamento de exemplos, podem ser re-utilizadas. Neste caso bastaria implementar pequenas partes de código, ou até mesmo somente criar um novo shell script.

O segundo grupo de ferramentas é responsável pela manipulação da sintaxe dos arquivos de exemplos. Estas ferramentas são necessárias pois, apesar dos arquivos de exemplos proposicionais compartilharem de um formato semelhante, existem diferenças sintáticas entre arquivos de exemplos armazenados em repositórios, como por exemplo o repositório da University da California at Irvine (UCI) [Merz 96]. Além disso, diferentes sistemas de aprendizado também utilizam diferentes sintaxes para arquivos de exemplos.

Estas ferramentas realizam as tarefas mais comuns na manipulação sintática de arquivos de exemplos. Entre estas tarefas está a mudança de posição da classe entre os demais atributos¹, a alteração do caracter que separa os atributos, a troca do caracter que marca o início de um comentário, etc.

Buscando facilitar o trabalho com as diversas sintaxes existentes para arquivos de exemplos, este trabalho propõe uma sintaxe padrão para o ambiente AMPSAM. Os arquivos de exemplos trazidos dos mais diferentes repositórios podem ser convertidos e armazenados nesta sintaxe padrão. Interfaces que convertem da sintaxe padrão para as sintaxes dos sistemas de aprendizado mais conhecidos podem ser criadas, utilizando as ferramentas sintáticas projetadas.

Na próxima seção será dada uma visão geral do ambiente AMPSAM, mostrando a integração de suas ferramentas.

¹Alguns algoritmos reconhecem a classe dos exemplos pela sua posição entre os demais atributos, usualmente a primeira ou a última posição são as mais freqüentes.

6.3 Uma Visão Geral do Ambiente AMPSAM

O problema geral para medir a performance de algoritmos de Aprendizado de Máquina por exemplos, ou seja, encontrar uma boa estimativa da taxa de erro verdadeira é o seguinte

Dados

- Um algoritmo A_a ;
- Um conjunto de n exemplos na sintaxe S ;
- Um conversor sintático, $CS(S, A_a)$, capaz de converter um conjunto de exemplos na sintaxe S para a sintaxe entendida pelo algoritmo A_a

deve-se utilizar algumas das técnicas estatísticas discutidas na seção 3.5.2 pg. 29 para separar esses n exemplos na sintaxe S em diversos conjuntos de treinamento e teste também na sintaxe S . A Figura 6.1 pg. 92 ilustra este processo no qual o conjunto de n exemplos é separado em k pares

$$(TR_i, TE_i) \quad i = 1 \dots k \quad (6.1)$$

de conjuntos de treinamento e teste respectivamente.

Os conjuntos de treinamento TR_i são convertidos pelo conversor sintático $CS(S, A_a)$ da sintaxe S para a sintaxe entendida pelo algoritmo A_a . Logo em seguida, o algoritmo A_a é treinado com cada um dos conjuntos de treinamento TR_i , produzindo o respectivo classificador $C_i(A_a)$.

Os conjuntos de teste TE_i também são convertidos sintaticamente pelo conversor sintático $CS(S, A_a)$. Logo após, o algoritmo A_a é alimentado com o conjunto de teste TE_i correspondente, medindo assim o erro de classificação E_i do classificador $C_i(A_a)$, $i = 1 \dots k$.

No caso da sintaxe S ser a sintaxe entendida pelo algoritmo A_a , então não há necessidade de realizar conversões sintáticas. Neste caso os conjuntos de treinamento e teste, respectivamente TR_i e TE_i , podem ser fornecidos diretamente ao algoritmo de aprendizado A_a , sem utilizar o conversor sintático $CS(S, A_a)$.

No caso da sintaxe S ser a sintaxe padrão especificada para ambiente AMPSAM, então o usuário tem à sua disposição alguns conversores sintáticos que transformam o arquivo de exemplos da sintaxe padrão para a sintaxe dos algoritmos de aprendizado de máquina mais conhecidos.

A próxima seção discute as ferramentas para particionamento de exemplos, e a seção seguinte as ferramentas para conversão sintática de arquivos de exemplos.

Figura 6.1: Uma Visão Geral do Sistema AMPSAM

6.4 As Ferramentas de Particionamento de Exemplos

A partir desta seção as ferramentas que compõem o ambiente AMPSAM são descritas com maiores detalhes. Logo no início do desenvolvimento deste projeto decidiu-se implementar a interface com o usuário em Inglês. Esta decisão foi tomada pois pretende-se compartilhar este trabalho com outros pesquisadores tornando o ambiente em um sistema de domínio público através da Internet.

Como já foi dito anteriormente, as ferramentas que constituem o ambiente AMPSAM podem ser divididas em dois grupos: ferramentas para particionamento de exemplos segundo os métodos de resampling, e ferramentas para manipulação sintática de conjuntos de exemplos. Uma característica importante das ferramentas para particionamento de exemplos é que estas trabalham independentemente da sintaxe em que se encontra o arquivo de exemplos. Para funcionar corretamente, estas ferramentas necessitam de duas condições

- O arquivo de exemplos deve conter apenas informações a respeito dos exemplos, demais informações como por exemplo o tipo dos atributos devem ser mantidas em um arquivo a parte;
- O arquivo de exemplos deve manter um exemplo por linha.

Para que as ferramentas de particionamento funcionem de forma correta, estas restrições devem ser seguidas. Estretanto, a restrição de manter um exemplo por linha

é bastante comum em arquivos de exemplos proposicionais. Manter uma seção em separado para a declaração de tipo dos atributos também é uma restrição comum dos algoritmos de Aprendizado de Máquina. Além disso, é sempre possível previamente retirar as declarações de atributos de um arquivo de exemplos, particionar os exemplos, e posteriormente voltar a integrar estas declarações com os exemplos já particionados.

Atualmente, foram implementadas as seguintes ferramentas de particionamento de exemplos

- Ferramenta para embaralhar um conjunto de exemplos;
- Ferramenta para particionar um conjunto de exemplos em k partições mutuamente exclusivas;
- Ferramenta para particionar um conjunto de exemplos em duas partições com $x\%$ exemplos e $(100\% - x)\%$ exemplos;
- Ferramenta para particionar um conjunto de exemplos com reposição em duas partições;
- Ferramenta para unir partições de exemplos a fim de criar conjuntos de treinamento e teste.

Estas ferramentas foram implementadas em linguagem C padrão ANSI, uma vez que esta linguagem é portátil a maioria dos sistemas operacionais existentes. Para implementar os métodos de resampling desejados, estas ferramentas foram integradas em C-shell scripts. Foram criados quatro scripts, cada um implementando um método de resampling diferente, são eles

- Holdout e Random Subsampling
- N-Fold Cross Validation
- Leaving-One-Out
- Bootstrapping

Com a implementação desses cinco métodos estatísticos, o ambiente tem à sua disposição uma gama de métodos aplicável a conjuntos de exemplos de tamanhos variados. Outros métodos podem vir a ser integrados ao ambiente, possivelmente utilizando as ferramentas já implementadas. Uma descrição detalhada da implementação e utilização das ferramentas de particionamento de exemplos do ambiente AMPSAM podem ser encontradas em [Batista 97e].

Na seção seguinte são discutidas as ferramentas para manipulação sintática de arquivos de exemplos.

6.5 As Ferramentas de Manipulação Sintática

De uma forma geral, os arquivos de exemplos utilizados pelos algoritmos de aprendizado de máquina seguem um formato geral. Este formato consiste de duas seções distintas

- A primeira seção declara os atributos utilizados no conjunto de exemplos juntamente com seus respectivos tipos de dados;
- A segunda seção consiste dos exemplos propriamente ditos.

A primeira seção consiste na declaração dos atributos que constituem os exemplos, juntamente com especificação do tipo de dado destes atributos. Nos arquivos de exemplos proposicionais, é comum encontrar três tipos de dados: inteiros, reais e enumerações. Na primeira seção também é declarada a classe que, para problemas de classificação, é sempre um atributo do tipo enumerado.

A segunda seção consiste dos exemplos propriamente ditos. Os exemplos são muito freqüentemente organizados na forma de colunas, onde cada coluna está relacionada com um atributo declarado na seção anterior. Normalmente, o primeiro atributo declarado está posicionado na primeira coluna da seção de exemplos, o segundo atributo na segunda coluna e assim por diante. Cada exemplo também está relacionado com uma classe, a classe muitas vezes está posicionada na última coluna, mas também é comum encontrá-la na primeira coluna desta seção.

Apesar deste ser o formato geral em que os conjuntos de exemplos proposicionais se encontram, diferentes sistemas de aprendizado e diferentes repositórios utilizam variações deste formato. Este problema incentivou a criação de uma sintaxe padrão e de conversores sintáticos desta sintaxe padrão para a sintaxe de alguns dos algoritmos de aprendizado de máquina simbólico mais conhecidos. Na próxima seção esta sintaxe padrão é discutida e especificada.

6.5.1 A Sintaxe Padrão para Arquivos de Exemplos

Um dos maiores e mais conhecidos repositórios de conjuntos de exemplos proposicionais é o repositório de Base de Dados para Aprendizado de Máquina da University of Califórnia at Irvine (UCI). É interessante manter a sintaxe padrão semelhante à sintaxe dos arquivos deste repositório, uma vez que muitas das avaliações de algoritmos de aprendizado realizados na área de Aprendizado de Máquina foram feitas baseadas nos conjuntos de exemplos deste repositório. Em um dos seus documentos eletrônicos, o repositório da UCI especifica da seguinte forma a sintaxe para arquivos de exemplos que venham a ser doados

“... Nós também preferimos que os dados possuam um formato padrão. Para base de dados de exemplos: um exemplo por linha, sem espaços, vírgulas entre valores de atributos, e valores desconhecidos denotados por “?” é o formato padrão.”

Apesar de não especificar rigidamente um formato padrão, o repositório da UCI especifica algumas diretrizes básicas, entretanto estas diretrizes não são suficientes para definir uma sintaxe para arquivos de exemplos. Por exemplo, tais diretrizes não especificam a posição da classe em relação aos demais atributos, é comum encontrar a classe como o primeiro ou último atributo nos arquivos de exemplos provenientes do repositório da UCI. Além disso, as diretrizes fornecidas não especificam como deve ser o formato do arquivo de declaração dos atributos, mesmo porque informações como os tipos de dados dos atributos costumam ser inseridas nos arquivos de documentação que acompanham os conjuntos de exemplos.

Desta forma, decidiu-se adotar como sintaxe padrão uma variação da sintaxe do sistema de aprendizado C4.5, uma vez que a sintaxe deste sistema é compatível com as diretrizes do repositório da UCI, e também por que o sistema C4.5, e por consequência a sua sintaxe, serem amplamente conhecidos.

A Figura 6.2 mostra um exemplo de um arquivo de declaração de atributos segundo a sintaxe padrão definida neste trabalho. Neste exemplo, deseja-se descobrir o país de fabricação de um carro a partir de suas características físicas.

O arquivo inicia especificando as classes, isto é os possíveis países de origem dos carros. São especificadas três classes: *USA*, *Europe* e *Japan*. O caracter “[” indica o início de um comentário, desta forma o texto presente a partir deste caracter até o final da linha não será interpretado. Logo a seguir são especificados os atributos, neste exemplo, cada atributo está associado a uma característica física dos carros analisados. Cada atributo possui um tipo de dado associado a ele. Atualmente, a sintaxe padrão suporta três tipos de dados para atributos: *integer*, *continuous* e o tipo enumerado. Para especificar um atributo do tipo inteiro ou contínuo (real), deve-se utilizar as palavras reservadas *integer* e *continuous*, respectivamente. Para especificar um atributo do tipo enumerado basta listar os possíveis valores que este atributo pode assumir. Os atributos e seus respectivos tipos para este exemplo são

1. *fueltype* especifica o tipo de combustível utilizado pelo automóvel. É um atributo do tipo enumerado que pode assumir dois valores *diesel* e *gas*;
2. *nrdoors* especifica o número de portas do automóvel. É um atributo do tipo inteiro;
3. *length* especifica o comprimento do carro. É um atributo contínuo, ou seja, pode assumir valores numéricos reais.

USA, Europe, Japan.	Classes
fueltype:	diesel, gas.
nrdoors:	integer.
length:	continuous.

Figura 6.2: Um exemplo de Arquivo de Declaração de Atributos na Sintaxe Padrão

Segundo a notação BNF, a sintaxe para arquivos de declaração de atributos pode ser definida da seguinte forma

```

<arquivo-atributos> ::=
    <declaracao-classe> .
    <declaracoes-atributos>

<declaracao-classe> ::=
    <string>
    | <string> , <declaracao-classe>

<declaracoes-atributo> ::=
    <declaracao-atributo> .
    | <declaracao-atributo> <declaracoes-atributos>

<declaracao-atributo> ::=
    <string> : <valores-atributos>
    | <string> : <tipo>

<valores-atributos>
    <string>
    | <string> , <valores-atributos>

<tipo> ::=
    continuous
    | integer

```

Um *string* é uma sequência de um ou mais caracteres composto de letras maiúsculas e minúsculas, dígitos e do símbolo “_”, com a restrição que o primeiro caracter deve ser uma letra.

A Figura 6.3 mostra um exemplo de arquivo de exemplos. Este arquivo apresenta apenas três casos ilustrativos, seus atributos e classes seguem as declarações de atributos especificados na Figura 6.2. A primeira coluna refere-se ao tipo de combustível do veículo (atributo fueltype), a segunda coluna refere-se ao número de portas do veículo (atributo

nrdoors), a terceira coluna refere-se ao comprimento do carro (atributo length), por fim a quarta coluna refere-se a classe do problema (país de fabricação).

diesel, 2, 177.8, Japan.
gas, 4, 158.8, USA.
gas, 2, 176.8, Europe.

Figura 6.3: Um Exemplo de Arquivo de Exemplos

Na sintaxe BNF, pode-se definir a sintaxe do arquivo de exemplos da seguinte forma

```
<exemplos> ::=
    <valores-atributos> .
    | <valores-atributos> . <exemplos>

<valores-atributos> ::=
    <valor>
    | <valor> , <valores-atributos>

<valor> ::=
    <string>
    | <numero>
    | ?

<numero> ::=
    <inteiro>
    | <real>
```

Nesta declaração, *inteiro* equivale a qualquer valor numérico inteiro e *real* a qualquer valor numérico real. O símbolo ? é utilizado sempre que não se sabe o valor que um atributo assume em um certo exemplo.

Como foi dito anteriormente, a sintaxe padrão é uma variação da sintaxe criada para o algoritmo C4.5. Atualmente, a única diferença entre a sintaxe padrão e a sintaxe do algoritmo C4.5 é a adição de um tipo de dados para atributos: o tipo integer. É sabido que nenhum algoritmo de aprendizado é capaz de entender um conjunto de exemplos nesta sintaxe (com ressalva ao próprio C4.5, caso o conjunto de exemplos não tenha atributos especificados com o tipo integer). A próxima seção discute em maiores detalhes as ferramentas de conversão sintática. Estas ferramentas são capazes de converter arquivos de uma sintaxe para outra, como por exemplo converter arquivos da sintaxe padrão para a sintaxe do algoritmo que se deseja avaliar.

6.5.2 As Ferramentas para Conversão Sintática

As ferramentas para conversão sintática de arquivos de exemplos são pequenos utilitários capazes de realizar as manipulações sintáticas mais comuns sobre arquivos de exemplos proposicionais. Estas ferramentas foram desenvolvidas através do utilitário Awk, o qual faz parte do conjunto de utilitários padrão do sistema operacional UNIX. O utilitário Awk [Dougherty 90] é capaz de procurar por padrões em um texto e executar algum código (cuja sintaxe é muito próxima da sintaxe da linguagem C padrão) relacionado com este padrão. A possibilidade de criar tais padrões com expressões regulares faz com que o projeto deste tipo de ferramentas seja em muito facilitado.

Atualmente, as ferramentas desenvolvidas são capazes de realizar as seguintes tarefas

- Alterar o nome dos atributos;
- Alterar os caracteres que separam os atributos e exemplos;
- Alterar símbolos, como por exemplo o símbolo que marca o início dos comentários ou o símbolo de atributo desconhecido;
- Apagar alguns exemplos do arquivo de exemplo;
- Apagar as linhas e espaços em branco desnecessários;
- Apagar cabeçalhos;
- Separar os exemplos conforme a classe a que pertencem².

Uma das tarefas mais comuns na manipulação sintática de arquivos de exemplos proposicionais é alterar o caracter ou caracteres que separam os atributos em um arquivo de exemplos. Na sintaxe padrão o caracter “,” realiza esta função. Para arquivos de exemplos, além do(s) caracter(es) utilizado(s) para separar atributos, existe ainda um ou mais caracteres utilizados para separar os exemplos (caracter “.” na sintaxe padrão). Em arquivos de declaração de atributos, também existem caracteres especiais. Um ou mais caracteres separam o nome do atributo do seu tipo (“:” para a sintaxe padrão), um ou mais caracteres separam cada declaração de atributo (“.” para a sintaxe padrão) e por fim um ou mais caracteres separam os valores que um atributo do tipo enumerado pode assumir (“,” para a sintaxe padrão). Em Awk estes caracteres especiais são chamados de *separadores*, e as ferramentas associam um nome a cada um destes caracteres.

Para ilustrar o funcionamento das ferramentas implementadas, será considerado um exemplo que consiste de trocar os caracteres separadores do arquivo de declaração de

²Esta facilidade é particularmente útil no caso da avaliação de alguns algoritmos de aprendizado de máquina relacionais.

atributos e do arquivo de exemplos da seção 6.5.1. A Figura 6.4 mostra quais são os caracteres separadores para o arquivo de declaração de atributos, juntamente com os nomes associados a cada um desses separadores, são eles

- *Attribute Enumeration Separator* representa o caracter ou caracteres que separam os valores que um atributo do tipo enumerado ou a classe podem assumir (uma vez que a classe é um atributo do tipo enumerado). Para as ferramentas, existe uma variável chamada *AES* que é utilizada sempre que se deseja especificar este(s) caracter(es);
- *Attribute Name Separator* representa o caracter ou caracteres que separam o nome do atributo do tipo a ele especificado. Nas ferramentas estes caracteres podem ser especificados através da variável *ANS*;
- *Attribute Record Separator* representa o caracter ou caracteres que separam cada uma das declarações de atributos. Nas ferramentas estes caracteres estão associados à variável *ARS*.

Figura 6.4: Caracteres Separadores para Arquivos de Declaração de Atributos

Como dito anteriormente, o exemplo consiste em alterar os caracteres separadores. Inicialmente é alterado o arquivo de declaração de atributos, sendo que neste exemplo específico o novo arquivo deve possuir as seguintes características

- O caracter “ ” será o novo Attribute Enumeration Separator (AES), ou seja, o símbolo utilizado para separar os valores de uma enumeração;
- O caracter “=” será o novo Attribute Name Separator (ANS), ou seja, o símbolo utilizado para separar o nome do atributo do seu tipo;
- O caracter “;” será o novo Attribute Record Separator (ARS), ou seja, o símbolo utilizado para marcar o fim de uma declaração de atributo.

O script Awk criado para substituir os caracteres separadores é chamado de *chatsep.awk* (*Change Attribute Separators*). Para executá-lo corretamente, basta especificar em seus parâmetros as variáveis que serão substituídas, como mostrado a seguir

```
chatsep.awk AES=',' OAES=' ' ANS=':' OANS='=' ARS='.' OARS=';'
```

As variáveis *OAES*, *OANS* e *OARS* significam respectivamente *Output Attribute Field Separator*, *Output Attribute Name Separator* e *Output Attribute Record Separator*. Estas variáveis indicam quais são os novos caracteres que substituirão os caracteres separadores.

Como o arquivo original de declaração de atributos estava na sintaxe padrão, então não é necessário especificar os valores das variáveis *AES*, *ANS* e *ARS*, pois estas variáveis assumem, por *default*, os valores ditados pela sintaxe padrão. Desta forma, bastaria executar o script Awk com os seguintes parâmetros

```
chatsep.awk OAES=' ' OANS='=' OARS=';'
```

A Figura 6.5 mostra a saída do script *chatsep.awk* com os novos valores para os caracteres modificados.

USA	Europe	Japan;	Classes
fueltype=		diesel	gas;
nrdoors=		integer;	
length=		continuous;	

Figura 6.5: Arquivo de Declaração de Atributos em Nova Sintaxe

Para arquivos de exemplos, também existe um script Awk que altera os caracteres separadores. Para este tipo de arquivo existem dois tipos de separadores, como mostra a Figura 6.6, são eles

- *Example Field Separator (EFS)*, é o caracter ou caracteres que separam os valores dos atributos (“,” para a sintaxe padrão);
- *Example Record Separator (ERS)*, é o caracter ou caracteres que separam os exemplos (“.” para a sintaxe padrão).

O script Awk *chexsep.awk* (*Change Example Separators*) foi criado para substituir estas variáveis, semelhante ao script do exemplo anterior (*chatsep.awk*), este script pode ser executado da seguinte forma

Figura 6.6: Caracteres Separadores para Arquivos de Exemplos

```
chexsep.awk EFS=', ' OEFS=' ' ERS='.' OERS=';'
```

Mais uma vez, as variáveis EFS e ERS não necessariamente precisam ser especificadas, uma vez que estas foram especificadas com os valores da sintaxe padrão.

Este arquivo de exemplos será modificado para que possua como separador de valores de atributos o caracter (espaço) e como separador de exemplos o caracter ”;”. A saída do script chexsep.awk é mostrada na Figura 6.7

```
diesel 2 177.8 Japan;
gas 4 158.8 USA;
gas 2 176.8 Europe;
```

Figura 6.7: Arquivo de Exemplos em Nova Sintaxe

A seção anterior mostrou como funcionam duas das ferramentas implementadas para realizar uma tarefa muito comum em arquivos de exemplo proposicionais: alterar os caracteres de separação. Descrições mais detalhadas juntamente com o código fonte de todas as ferramentas implementadas podem ser encontradas em [Batista 97d]. Na próxima seção é mostrado um exemplo mais complexo, utilizando diversas ferramentas do ambiente AMPSAM.

6.6 Um Exemplo de Conversão Sintática

Esta seção apresenta um problema mais complexo: converter um arquivo de exemplos na sintaxe padrão para a sintaxe entendida pelo algoritmo de aprendizado CN2. Para tanto, é necessário utilizar mais de uma ferramenta para realizar esta conversão.

A Figura 6.8 ilustra os passos necessários para a conversão do arquivo de declaração de atributos, sendo a Figura 6.8 (a) o arquivo na sintaxe padrão e a Figura 6.8 (f) o arquivo na sintaxe entendida pelo sistema CN2.

Figura 6.8: Conversão do Arquivo de Declaração de Atributos entre a Sintaxe Padrão (a) e a Sintaxe CN2 (f)

A primeira conversão a ser realizada é dar um nome à classe. Na sintaxe do algoritmo CN2, a classe é declarada como se fosse mais um atributo. O algoritmo CN2 diferencia a classe dos demais atributos uma vez que a classe é sempre o último atributo. A ferramenta *insatcn.awk* (*Insert Attribute Class Name*) insere o nome da classe na primeira linha do arquivo de declaração de atributos. Seu único parâmetro é o nome que será dado à classe, através da variável *CN* (*Class Name*). O resultado da aplicação deste script é mostrado a seguir.

`insatcn.awk CN='Class'`

Resta então posicionar a classe na última posição do arquivo de declaração de exemplos. Esta troca é feita através de uma ferramenta chamada *chatpos.awk* (*Change Attribute*

Position). Duas variáveis indicam as posições a serem trocadas, são elas *OP* (*Old Position*) e *NP* (*New Position*). Com estas variáveis é possível especificar uma ou mais posições a serem trocadas. Dois valores especiais *F* (*First*) e *L* (*Last*) permitem posicionar um determinado atributo ou classe na primeira ou última posição. Desta forma, é possível alterar a posição da classe com a seguinte execução do script

```
chatpos.awk OP='F' NP='L'
```

A próxima conversão consiste em trocar os separadores. A ferramenta responsável por trocar estes caracteres é *chatsep* (*Change Attributes Separators*), descrita no exemplo da seção 6.5.2. Assim,

```
chatsep.awk OAES=' ' ONAS=':' OARS=';'
```

A conversão seguinte consiste em trocar o tipo de dados dos atributos. Além do tipo enumerado, a sintaxe do algoritmo CN2 possui outros dois tipos de dados: o tipo (*FLOAT*) semelhante ao tipo continuous especificado na sintaxe padrão, e o tipo (*INT*) semelhante ao tipo integer da sintaxe padrão. A ferramenta capaz de trocar os tipos de dados de atributos é chamada de *chatdt.awk* (*Change Attribute Data Type*). Duas variáveis *ODT* (*Old Data Types*) e *NDT* (*New Data Types*) especificam uma lista de nomes de tipos de dados de atributos que serão trocados. Os elementos (nomes de tipos de dados) dessa lista estão separados por vírgulas. Além destas variáveis é também necessário especificar as variáveis separadoras, pois o arquivo não está mais com os separadores da sintaxe padrão.

```
chatdt.awk AES=', ' NAS=':' ARS=';' ODT='continuous, integer'
          NDT='(FLOAT), (INT)'
```

Por fim é necessário modificar o caracter de comentário, que na sintaxe do algoritmo CN2 é definida pelo caracter “%”. Esta tarefa pode ser realizada por uma ferramenta genérica de procura e substituição. Esta ferramenta é chamada de *chpat.awk* (*Change Pattern*) e possui duas variáveis: *OP* (*Old Patterns*) que identifica os padrões a serem pesquisados e *NP* (*New Patterns*) que identifica os textos que substituirão os padrões encontrados.

`chpat.awk OP='|' NP='%'`

Todas estas ferramentas podem ser integradas em um único shell script, mostrado a seguir, responsável por converter arquivos de declaração de atributos da sintaxe padrão para a sintaxe CN2. Desta forma, não é mais necessário converter manualmente os arquivos de exemplos quando se está trabalhando com a sintaxe padrão e o algoritmo CN2.

```

insatcn.awk CN='Class'
chatpos.awk OP='F' NP='L'
chatsep.awk OAES=' ' ONAS=':' OARS=';'
chatdt.awk AES=', ' NAS=':' ARS=';' ODT='continuous, integer'
          NDT='(FLOAT), (INT)'
chpat.awk OP='|' NP='%'

```

Da mesma forma pode-se criar outros shell scripts que convertam arquivos em diferentes sintaxes. Um dos objetivos do ambiente AMPSAM é prover um conjunto de conversores sintáticos entre a sintaxe padrão e a sintaxe de vários dos sistemas de aprendizado simbólico existentes, simplificando ainda mais o processo de avaliação de alguns dos mais conhecidos sistemas de aprendizado simbólico.

6.7 Comentários a Respeito do Utilitário Awk

Awk [Dougherty 90] faz parte dos utilitários do sistema operacional Unix, e possui este nome em referência a seus criadores Aho, Weinberger e Kernighan. Além da versão Awk original foram criadas outras duas versões às quais foram adicionadas algumas funcionalidades, estas versões são chamadas de *nawk* (*New Awk*) e *gawk* (*GNU project's version of Awk*).

Awk é muito útil para trabalhos que envolvem arquivos texto. Este utilitário permite que padrões de caracteres sejam descritos através de expressões regulares. Cada padrão possui um trecho de código associado (escrito em uma linguagem semelhante à linguagem C), e este código é executado sempre que o padrão é encontrado no texto.

A utilização do utilitário Awk simplificou muito a implementação das ferramentas do ambiente AMPSAM. Entretanto algumas características, limitações e até mesmo erros de implementação podem dificultar a sua utilização.

Como características indesejáveis do utilitário Awk pode-se citar a forma como este trata algumas situações incomuns tais, como erros de exceções. Por exemplo, Dougherty cita em [Dougherty 90] que o utilitário Awk possui uma limitação quanto ao tamanho máximo de um registro, o qual não deve ultrapassar 3000 caracteres. Quando tal situação ocorre o utilitário é simplesmente abortado, gerando um arquivo *core* e nenhuma mensagem de erro é dada ao usuário.

Como erros de implementação o mesmo autor cita que alguns scripts Awk grandes fazem com que o utilitário reporte erros de sintaxe que não fazem sentido. Ele diz ainda que removendo todos os comentários o script passou a funcionar sem problemas. Por fim é comentado que o programa com todos os comentários funciona sem problemas em outros dois sistemas operacionais diferentes.

Como limitações sérias impostas pelo Awk, pode-se citar a variável *record separator*. Esta é uma variável do sistema e através dela pode-se definir quais caracteres definem o final de um registro. Awk especifica que esta variável deve conter apenas um caracter. Desta forma, Awk não entende registros separados por dois caracteres como por exemplo “.\n” (ponto e nova linha). Para o ambiente AMPSAM esta restrição teve que ser contornada com implementações adicionais, uma vez que é comum encontrar arquivos nos quais os exemplos estão separados por mais de um caracter. De uma forma geral, scripts Awk poderiam ser mais poderosos se a variável *record separator* pudesse ser especificada como uma expressão regular.

Uma alternativa ao utilitário Awk, a qual predende-se explorar no futuro, é a utilização da linguagem de programação Perl [Wall 96]. A princípio Perl não possui muitos dos problemas encontrados em Awk, e ainda permite a utilização de expressões regulares para a especificação de padrões. Infelizmente, Perl também não permite que o separador de registros seja uma expressão regular, mas permite que esta variável receba mais de

um caracter.

6.8 Trabalhos Relacionados

Alguns trabalhos da comunidade de aprendizado de máquina possuem características e/ou objetivos comuns a partes deste trabalho. Entre eles encontra-se o projeto Statlog o qual faz parte do projeto ESPRIT. Neste projeto foi analisado o comportamento de mais de vinte algoritmos sobre mais de vinte conjuntos de exemplos diferentes. Uma descrição deste projeto assim como os resultados das análises dos algoritmos podem ser encontrados em [Michie 94]. Para facilitar a análise dos diversos algoritmos foi criado um ambiente conhecido como Evaluation Assistant.

Um outro projeto que está sendo desenvolvido por Ron Kohavi é conhecido como *MCC++* [Kohavi 96]. Este projeto foi iniciado na Universidade de Stanford e atualmente conta com a parceria da Silicon Graphics. O principal objetivo é desenvolver uma biblioteca em C++ que possa ser utilizada por outros programas. Atualmente, esta biblioteca possui classes que implementam

- Conversores sintáticos entre conjuntos de exemplos e os algoritmos de Aprendizado de Máquina mais conhecidos;
- Métodos estatísticos para estimar a taxa de erro verdadeira;
- Alguns sistemas de aprendizado como o nearest-neighbor e o ID3;
- Ferramentas para criar conjuntos de exemplos artificiais;
- Interface gráfica para apresentar os resultados dos algoritmos.

O sistema *MCC++* utiliza como sintaxe de entrada padrão para seus algoritmos a sintaxe do algoritmo C4.5, este também foi um dos fatores que contribuíram para que uma variação da sintaxe do C4.5 fosse adotada como sintaxe padrão no ambiente AMPSAM.

O projeto WEKA está sendo desenvolvido na Universidade de Waikato na Nova Zelândia [Garner 95]. Seu objetivo é criar um sistema que auxilie na avaliação de sistemas de aprendizado, principalmente para conjuntos de exemplos do setor de agricultura da Nova Zelândia. Este sistema possui algumas características comuns ao sistema AMP-SAM, entre elas

- O sistema WEKA é composto de ferramentas, entre elas conversores sintáticos e ferramentas para dividir os arquivos de exemplos segundo métodos estatísticos;

- Sintaxe padrão para descrever exemplos proposicionais chamada de *Attribute-Relation File Format (ARFF)*;
- Sintaxe padrão para regras de decisão e, caso a saída de um algoritmo simbólico seja convertida para esta sintaxe, é possível utilizar um avaliador de regras para realizar algumas análises sobre esta saída do algoritmo.

6.9 Considerações Finais

Neste capítulo foram discutidos alguns dos requisitos necessários para a implementação de um ambiente para a avaliação de performance de sistemas de Aprendizado de Máquina. Como Aprendizado de Máquina é uma área de pesquisa muito dinâmica, novos sistemas e melhoramentos a sistemas de aprendizado já existentes são propostos com frequência. Desta forma, se o ambiente fosse implementado como um único sistema seriam necessárias atualizações freqüentes. Portanto, foi decidido criar o ambiente como sendo um conjunto de ferramentas genéricas. Estas ferramentas, quando compostas em C shell scripts, resolvem problemas específicos tais como a conversão de sintaxe de um conjunto de exemplos entre duas sintaxes conhecidas.

A sintaxe padrão proposta busca simplificar o processo de avaliação de algoritmos de Aprendizado de Máquina, uma vez que o ambiente fornece alguns conversores sintáticos capazes de converter arquivos nesta sintaxe para a sintaxe de alguns dos algoritmos de aprendizado de máquina mais conhecidos.

Ainda que a utilização do programa Awk permitiu uma simplificação considerável na implementação do ambiente, ele possui alguns defeitos que o tornam pouco confiável. Como trabalho futuro pretende-se realizar uma re-implementação das ferramentas em Awk para Perl, cuja manutenção é freqüente encontrando-se atualmente na versão 5.

Por fim, pode-se dizer que o ambiente AMPSAM simplifica e torna o processo de estimar a taxa de erro verdadeira para algoritmos de aprendizado de máquina simbólico mais ágil. Para analisar o comportamento do ambiente AMPSAM foram realizados experimentos utilizando conjuntos de exemplos provenientes do repositório de aprendizado de máquina da UCI [Merz 96] e diversos sistemas de aprendizado de máquina simbólico. Uma discussão a respeito dos resultados obtidos em alguns desses experimentos são apresentados no próximo capítulo.

Capítulo 7

Avaliação Experimental de alguns Algoritmos Simbólicos

7.1 Considerações Iniciais

Neste capítulo são realizadas algumas avaliações experimentais de algoritmos de aprendizado simbólico. Estas avaliações têm como principal objetivo demonstrar o funcionamento do ambiente AMPSAM aplicado a conjuntos de exemplos reais e a sistemas de aprendizado simbólico conhecidos. Não é objetivo deste trabalho concluir qual é o melhor sistema de aprendizado para cada conjunto de exemplo selecionado. Métodos estatísticos confiáveis para comparar as performances de sistemas de aprendizado são um assunto em discussão na área de Aprendizado de Máquina e são discutidos brevemente na seção 7.9.

Para estes experimentos, foram selecionados três dos mais conhecidos algoritmos de aprendizado simbólico indutivo: NewId [Boswell 90], CN2 [Boswell 90a] e FOIL [Cameron-Jones 93]. Estes sistemas foram escolhidos pelos seguintes motivos

- Eles possuem sintaxes dos conjuntos de exemplos e atributos diferentes da sintaxe padrão do ambiente AMPSAM, permitindo assim que os conversores sintáticos sejam verificados;
- Estes sistemas pertencem a diferentes classes de sistemas simbólico indutivo. O NewId é um sistema proposicional que representa suas hipóteses através de árvores de decisão, o CN2, também proposicional, utiliza regras de decisão para representar as hipóteses, e o FOIL é um sistema relacional que utiliza cláusulas de Horn para representar suas hipóteses.

Esses algoritmos de aprendizado de máquina simbólico tiveram as suas performances

medidas sobre 6 conjuntos de exemplos de domínios reais. O método utilizado para estimar a taxa de erro verdadeira foi o 10-fold cross validation. Este trabalho apresenta também as taxas de erros aparentes para estes conjuntos de exemplos, a fim de mostrar quanto a taxa de erro aparente é um estimador tendenciosamente otimista da taxa de erro verdadeira.

7.2 Os Conjuntos de Exemplos Utilizados

Para a análise experimental deste trabalho foram escolhidos 6 conjuntos de exemplos reais, retirados do repositório de aprendizado de máquina da UCI. Não foram escolhidos conjuntos de exemplos artificiais uma vez que muitos desses conjuntos são puramente sintéticos e possuem uma estrutura extremamente regular, permitindo que exista uma solução 100% correta, o que é atípico para problemas reais [Prechelt 94]. Um fato a favor dos conjuntos de exemplos artificiais é que muitos desses conjuntos de exemplos possuem características internas bem conhecidas e com isso é possível analisar se um algoritmo é capaz ou não de lidar com tais características. Entretanto, é difícil de medir quanto os resultados obtidos em conjuntos de exemplos artificiais podem dizer a respeito do comportamentos desses sistemas de aprendizado em tarefas de aprendizado para o mundo real.

Os conjuntos de exemplos selecionados para a análise experimental deste trabalho são originários dos seguintes domínios de aplicação

1. *Hepatitis Domain*, chamado neste trabalho de *Hepatitis*. Utilizado para predizer a expectativa de vida de pacientes com hepatite;
2. *Breast Cancer Data*, chamado neste trabalho de *Breast Cancer*. Utilizado para predizer a recorrência de câncer de mama;
3. *Pima Indians Diabetes Database*, chamado neste trabalho de *Pima*. Utilizado para o diagnóstico de diabetes;
4. *Bupa Liver Disorders*, chamado neste trabalho de *Bupa*. Utilizado para diagnosticar doenças do fígado;
5. *Heart Disease Database (Hungarian Institute of Cardiology)*, chamado neste trabalho de *Hungarian*. Utilizado para o diagnóstico de doenças do coração;
6. *Credit Approval*, chamado neste trabalho de *Credit*. Utilizado para aprovação de crédito a clientes.

A Tabela 7.1 apresenta algumas características desses conjuntos de exemplos, tais como o número total de exemplos ($\#E$) presentes em cada conjunto de exemplos, número

de atributos discretos (*#Discreto*), número de atributos contínuos (*#Contínuo*) e se existem ou não atributos desconhecidos (*Desconhecido?*). A tabela mostra também os nomes das duas classes¹ e o número de exemplos pertencentes a cada uma delas.

Nome	#E	#Discreto	#Contínuo	Desconhecido?	Classe			
					X		Y	
1-Hepatitis	155	13	6	y	die	32	live	123
2-Breast Cancer	285	5	4	y	rcr	84	no_rcr	201
3-Pima	768	0	8	n	0	500	1	268
4-Bupa	345	0	6	n	1	145	2	200
5-Hungarian	294	0	13	y	pres	106	abs	188
6-Credit	490	10	5	y	is_+	217	is_-	273

Tabela 7.1: Características dos Conjuntos de Exemplos

Pode ser observado que a proporção de exemplos da classe *X* e *Y* varia bastante entre os conjuntos de dados escolhidos. A Figura 7.1 mostra graficamente a proporção de exemplos entre a classe *X* e *Y* para cada conjunto de exemplos.

Figura 7.1: Proporção entre as Classes *X* e *Y* para os Conjuntos de Exemplos

No caso que o conjunto de exemplos possua equilíbrio entre as classes, e que possua o mesmo custo de classificação incorreta, o requisito mínimo para o classificador é uma taxa de erro menor que 50%. Quando não existir um equilíbrio entre as classes, o

¹Foram selecionados conjuntos de exemplos com duas classes, pois o sistema FOIL aprende um conceito positivo e utiliza a negação por falha para classificar os exemplos negativos.

erro mínimo requerido deve ser menor que aquele obtido pelo classificador que classifica qualquer exemplo como pertencente à classe de maior número de exemplos. Por exemplo, o conjunto de dados Hepatitis é o que apresenta a maior desproporção, aproximadamente 20% dos exemplos pertencem à classe X e 80% à classe Y . Para este conjunto de exemplos, um classificador que decide sempre pela classe majoritária — classe Y — terá uma precisão mínima de aproximadamente 80%, isto é, uma taxa de erro próxima de 20%. Este fato só é verdadeiro se a amostra for representativa da população.

Um desbalanço muito grande entre as classes pode dificultar o aprendizado do conceito referente à classe representada pela minoria dos exemplos. Segundo [Kubat 97], neste contexto, a taxa de erro média não é um bom índice de performance para um sistema de aprendizado, e um melhor índice seria a média geométrica das classificações incorretas para as classes positiva e negativa. Nesse trabalho, é discutido também um critério para diminuir o número de exemplos da classe majoritária a fim de balancear o número total de exemplos em cada classe.

7.3 Complexidade do Modelo e Parâmetros do Modelo

Muitos dos sistemas de aprendizado permitem que sejam feitos ajustes em seus modelos para melhorar o aprendizado. Por exemplo, em árvores de decisão deve-se avaliar quais nós devem ser podados da árvore. Uma árvore com muitos nós pode levar a um overfitting dos exemplos, o que causaria baixas taxas de erro aparente, mas altas taxas de erro verdadeiras. Um outro exemplo ocorre com redes neurais, nestes sistemas é possível determinar o número de camadas e o número de nós (neurônios) participantes em cada camada. Uma rede neural com muitas camadas ou muitos nós por camada pode levar a um overfitting dos exemplos. Outro problema com redes neurais é o excesso de treinamento que também pode levar ao overfitting. Desta forma, para melhorar o desempenho destes sistemas deve-se encontrar o melhor ajuste da *complexidade do modelo*.

Existem na literatura muitos exemplos que mostram a relação entre o grau de complexidade do modelo e as taxas de erro aparente e verdadeira. Por exemplo, a Figura 7.2, criada a partir de informações de [Weiss 96], mostra esta relação para árvores de decisão de diversos tamanhos. É possível verificar que a taxa de erro verdadeira diminui com o aumento do número de nós, ou seja, da complexidade do modelo, entretanto um aumento exagerado dessa complexidade pode levar ao overfitting dos exemplos. Esse overfitting dos exemplos faz com que a taxa de erro verdadeira volte a aumentar, enquanto que a taxa de erro aparente continua a decrescer. A Figura 7.2 mostra também que existe o risco de um *underfitting* dos exemplos, causado pela seleção de um modelo com complexidade muito simples incapaz de aprender o conceito contido nos exemplos. Para alguns conjuntos de exemplos o underfitting pode se tornar um problema mais sério que o overfitting, levando a taxas de erros muito mais altas.

Figura 7.2: Taxas de Erro Aparente (—) e do Conjunto de Teste (- - -) para Árvore de Decisão de Diversos Tamanhos (Criado a Partir de Informações em [Weiss 96])

Além da complexidade do modelo, existem outros fatores dependentes de cada modelo que são diretamente relevantes a performance de um sistema de aprendizado, a estes fatores dá-se o nome de *parâmetros do modelo*. Como exemplo destes fatores pode-se citar o *termo momentum* e a *taxa de aprendizado* das redes neurais, ou a frequência mínima de casos pertencentes a uma determinada classe para que um nó possa se tornar nó terminal em uma árvore de decisão.

Os métodos de resampling podem ajudar a encontrar a melhor complexidade do modelo e os melhores parâmetros do modelo para uma determinada aplicação. Para isso deve-se estimar a taxa de erro verdadeira para o sistema de aprendizado segundo diversos parâmetros diferentes e escolher a combinação de parâmetros que fornece a menor taxa de erro. Por exemplo, pode-se treinar diversas árvores de decisão com número de nós diferentes, e selecionar a árvore que apresenta menor taxa de erro estimada.

Como um dos objetivo deste trabalho é demonstrar o funcionamento do ambiente AMP-SAM, as performances dos sistemas de aprendizado foram obtidas utilizando, no treinamento, os parâmetros padrões de cada sistema. No caso das árvores de decisão, foi realizada uma poda com os mesmos exemplos utilizados no treinamento, na realidade, um método que poderia obter melhores resultados seria realizar a poda utilizando um conjunto de exemplos especialmente separado, conhecido como *conjunto de validação*. É importante ressaltar que, em trabalhos que buscam comparar as performances de diferentes sistemas de aprendizado, é essencial encontrar uma boa combinação de parâmetros e da complexidade do modelo a fim de obter uma performance próxima da melhor possível de cada sistema.

7.4 Resultados da Taxa de Erro Aparente

A fim de verificar se a taxa de erro aparente é realmente um estimador tendenciosamente otimista da taxa de erro verdadeira, foram medidas as taxas de erro sobre os conjuntos de treinamento. Foram utilizadas algumas medidas previamente conhecidas, extraídas de [Monard 95], no qual foram medidas as taxas de erros aparentes para os algoritmos de aprendizado CN2 e FOIL. Essas medidas foram complementadas com as taxas de erro aparentes para o sistema NewId, medidas com o auxílio do ambiente AMPSAM, resultando na Tabela 7.2.

Conjunto de Exemplos	Sistema de Aprendizado	Taxa de Erro Aparente		
		Classe X	Classe Y	Total
1-Hepatitis	CN2	5.6	0.0	3.2
	NewId	51.0	2.0	12.0
	FOIL	8.7	1.6	5.2
2-Breast Cancer	CN2	20.0	0.0	5.9
	NewId	22.6	0.0	6.7
	FOIL	20.2	4.0	8.8
3-Pima	CN2	19.8	0.0	8.7
	NewId	13.0	3.0	7.0
	FOIL	3.7	3.6	3.6
4-Bupa	CN2	0.0	38.0	15.7
	NewId	1.0	3.0	2.0
	FOIL	3.5	1.5	2.3
5-Hungarian	CN2	14.2	0.0	5.1
	NewId	34.0	10.0	19.0
	FOIL	10.4	1.1	4.4
6-Credit	CN2	14.3	0.0	6.3
	NewId	7.0	1.0	4.0
	FOIL	1.9	4.7	3.3

Tabela 7.2: Taxas de Erro Aparente para os Algoritmos CN2 e FOIL ([Monard 95]) e NewId (AMPSAM).

Para medir a taxa de erro aparente, os sistemas de aprendizado foram treinados com todos os exemplos disponíveis. Posteriormente, a taxa de erro foi medida utilizando esses mesmos exemplos. Como dito anteriormente, os sistemas de aprendizado foram treinados com os parâmetros padrões.

O sistema FOIL aprende um conceito que diferencia entre duas classes, aprendendo como classificar os exemplos de uma das classes — classe positiva — e qualquer exemplo que não puder ser classificado com o conceito aprendido é classificado como pertencente à outra classe — classe negativa — em um processo conhecido como negação por falha (seção 4.4.1 pg. 51). Para que o sistema FOIL aprenda tanto o conceito referente à classe X quanto o conceito referente à classe Y é necessário executá-lo duas vezes. Na primeira, os exemplos da classe X são fornecidos como exemplos positivos

e, os exemplos da classe Y são fornecidos como exemplos negativos deste conceito. Na segunda execução é feito o processo contrário, fornecendo os exemplos da classe Y como exemplos positivos e os exemplos da classe X como exemplos negativos. Para o sistema FOIL, a Tabela 7.2 mostra a média ponderada das taxas de erro para a classe X e Y .

7.5 Como Foram Realizadas as Medidas

Para estimar a taxa de erro verdadeira, foi escolhido o método de resampling 10-fold cross validation. Este método é aconselhável para conjuntos de exemplos de tamanho médio, pois é capaz de combinar eficiência computacional e boa precisão para estimar a taxa de erro verdadeira. Todos os conjuntos de exemplos selecionados podem ser considerados de tamanho médio, com exceção do conjunto Hepatitis que possui poucos casos para a classe X . Possivelmente, para este conjunto de exemplos, um método mais preciso e mais caro computacionalmente, como o leaving-one-out ou o bootstrapping, poderia fornecer uma taxa de erro mais precisa.

Os conjuntos de exemplos foram embaralhados e particionados em 10 partições mutuamente exclusivas, utilizando as ferramentas para particionamento de exemplos do ambiente AMPSAM. Posteriormente, foram construídos os conjuntos de treinamento e teste segundo o método 10-fold cross validation. Após a construção dos conjuntos de treinamento e teste, foram aplicados os conversores sintáticos para cada algoritmo. Com isso, para cada conjunto de exemplos, os algoritmos foram medidos utilizando os mesmos 10 conjuntos de treinamento e teste. Este procedimento pode ajudar em uma análise mais minuciosa para entender as possíveis diferenças de performances entre os sistemas de aprendizado. Estatisticamente, as taxas de erro calculadas a partir de conjuntos de treinamento e teste construídos a partir do mesmo ou de diferentes particionamentos dos dados possuem a mesma validade.

O sistema FOIL requereu uma atenção especial pois, por ser um sistema relacional, o formato dos seus arquivos de exemplos é muito diferente do formato dos arquivos proposicionais. Foi necessário criar uma ferramenta para conversão sintática, em AWK, especialmente para este sistema. Por exemplo, o sistema FOIL requer que exemplos positivos e negativos fiquem separados em seções diferentes do arquivo, neste caso uma outra ferramenta que separa exemplos pertencentes a classes diferentes foi muito útil, uma vez que manualmente esta tarefa poderia levar horas.

Os algoritmos CN2 e NewId fornecem seus resultados através de matrizes de confusão (discutidas na seção 3.3 pg. 14), as quais possuem o seguinte formato para problemas que envolvem duas classes

	<i>Classes Predizidas</i>	
<i>Classes Verdadeiras</i>	Classe <i>X</i>	Classe <i>Y</i>
Classe <i>X</i>	a	b
Classe <i>Y</i>	c	d

onde,

- *a* representa o número de casos pertencentes à classe *X*, corretamente classificados;
- *b* o número de casos pertencentes à classe *X* incorretamente classificados como pertencentes à classe *Y*;
- *c* o número de casos da classe *Y* incorretamente classificados como pertencentes à classe *X* e;
- *d* o número de casos da classe *Y* corretamente classificados.

No decorrer deste capítulo os símbolos *a*, *b*, *c*, e *d* são utilizados para descrever tais valores.

O sistema FOIL não fornece como resultado uma matriz de confusão, mas uma lista de todos os exemplos incorretamente classificados. A partir dessa informação foram montadas as matrizes de confusão para esse sistema.

Por fim, foram colhidas as saídas dos algoritmos e esses resultados foram fornecidos ao sistema SMEC— System for Measuring the Error of Classifiers — descrito a seguir, para realizar os cálculos de performance e a construção de alguns dos gráficos mostrados neste capítulo.

7.6 O Sistema SMEC

A fim de facilitar a análise de medidas realizadas utilizando os diversos métodos de resampling implementados no ambiente AMPSAM, encontra-se em fase final de implementação um sistema computacional em Maple V² descrito em detalhes em [Monard 97b].

Maple possui um grande número de comandos para gerar gráficos em uma e duas dimensões, bem como uma vasta biblioteca de rotinas que incluem rotinas da área de estatística e gráficos. Neste trabalho, gráficos de duas e três dimensões são bastante

²Maple V é um poderoso *Sistema de Computação Simbólica* disponível no mercado desde 1995. Maple V é preciso, expansível e interativo, além de estar disponível na maioria dos Sistema Operacionais mais utilizados.

explorados para auxiliar ao usuário a realizar interativamente, utilizando as facilidades de regiões para executar unidades lógicas de código fornecida no ambiente *worksheet* do Maple, diversas análises estatísticas das medidas realizadas.

7.6.1 Descrição do Sistema SMEC

Os dados de um mesmo experimento devem ser fornecidos ao sistema como uma estrutura de *tabela*. Maple possui estruturas de dados tais como seqüência de expressões, listas, conjuntos, arrays e tabelas. A tabela é uma extensão do conceito da estrutura *array*. A diferença entre array e tabela é que uma tabela pode ter qualquer coisa como índice, não somente inteiros.

As matrizes de confusão provenientes de um experimento são inseridas em uma estrutura de dados do tipo tabela, juntamente como o nome do algoritmo e do conjunto de exemplos utilizados no experimento. Diversas tabelas contendo resultados de experimentos diferentes podem ser armazenadas no mesmo arquivo de entrada. Por exemplo, a tabela a seguir nomeada **CN2_hepatitis** mostra a visualização de Maple da representação das matrizes de confusão resultantes do *n-fold cross validation*, para $n = 10$, do algoritmo **CN2** utilizando o conjunto de testes **hepatitis**.

```
CN2_hepatitis:= table([Algorithm = CN2,
                      Examples= hepatitis,
                      Results=[ array([[1,4],[0,1]]),
                                array([[2,3],[0,1]]),
                                array([[1,4],[0,1]]),
                                array([[0,1],[0,15]]),
                                array([[0,2],[2,12]]),
                                array([[0,1],[0,14]]),
                                array([[1,1],[0,13]]),
                                array([[2,2],[0,11]]),
                                array([[1,4],[1,9]]),
                                array([[0,2],[0,13]])]
                      ]):
```

Após inicializar o sistema com o nome da tabela a ser analisada — **CN2_hepatitis** neste exemplo — o usuário pode executar interativamente o sistema que fornece tanto resultados numéricos quanto gráficos.

O sistema SMEC calcula medidas descritivas de centralidade, tanto do erro no conjunto quanto do erro em cada uma das classes X e Y , foram implementadas a média aritmética, a mediana e uma variante da média geométrica — por nós denominada *Média G* — bem como os desvios padrões dessas médias. O mínimo e máximo valor do

erro cometido em todas as execuções no conjunto em cada uma das classes e, a média G também é fornecida pelo sistema.

Alguns dos gráficos³ que o usuário pode requerer são os valores de $\frac{b+c}{a+b+c+d}$, erro no conjunto, de cada execução junto com o valor da média aritmética e geométrica, ou média aritmética e mediana.

7.6.2 As Medidas Calculadas pelo Sistema SMEC

Os índices de performance dos sistemas de aprendizado podem ser relatados na forma de taxa de erro ou na forma de precisão de classificação⁴. Os pesquisadores da área de Estatística costumam divulgar seus resultados na forma de taxa de erro enquanto que os pesquisadores da área de Aprendizado de Máquina e Data Mining — DM — costumam utilizar a precisão de classificação. Um bom motivo para utilizar a taxa de erro ao invés da precisão de classificação é que aparentemente não existe muita diferença entre dois classificadores que são capazes de acertar 96% e 98% dos casos, respectivamente. Entretanto quanto analisados sob a perspectiva da taxa de erro esses mesmos classificadores possuem performance de 4% e 2%, respectivamente, uma diferença de 100%. Por esse motivo, os resultados deste trabalho são relatados na forma de taxa de erro.

Como já foi dito anteriormente, o método escolhido para estimar a taxa de erro verdadeira foi o 10-fold cross validation. Desta forma, cada algoritmo foi executado 10 vezes e foram anotadas as matrizes de confusão. A partir dessas matrizes é possível determinar a taxa de erro para cada classe e a taxa de erro total no conjunto de exemplos. A Tabela 7.3 mostra como esses valores são calculados.

Classes Verdadeiras	Classes Preditas		Taxa de Erro na Classe	Taxa de Erro no Conjunto
	X	Y		
X	a	b	$\frac{b}{a+b}$	$\frac{b+c}{a+b+c+d}$
Y	c	d	$\frac{c}{c+d}$	

Tabela 7.3: Matriz de Confusão e Taxas de Erro de Classificação

Deve-se observar que

- $\frac{b}{a+b}$ é a taxa de erro para a classe X , uma vez que b é o número de exemplos da classe X incorretamente classificados, e $a + b$ é o número de casos pertencentes à classe X ;

³O Apêndice A mostra com maiores detalhes alguns dos gráficos fornecidos pelo sistema SMEC.

⁴*Classification accuracy*.

- $\frac{c}{c+d}$ é a taxa de erro para a classe Y , uma vez que c é o número de exemplos da classe Y incorretamente classificados, e $c + d$ é o número de casos pertencentes à classe Y ;
- $\frac{b+c}{a+b+c+d}$ é a taxa de erro para todo o conjunto de exemplos, uma vez que $a+b+c+d$ é o número total de casos do conjunto de exemplos.

Para cada iteração do método cross validation é obtida uma matriz de confusão, e são calculadas as taxas de erro descritas anteriormente. Devido à quantidade de iterações do método cross validation, de conjuntos de exemplos e de sistemas de aprendizado que estão sendo analisados, são relatadas neste trabalho somente as medidas das taxas de erro calculadas. Como dito anteriormente, para cada taxa de erro são calculadas a média aritmética, a mediana e a média G. A média aritmética pode ser calculada através da equação

$$M_a = \frac{1}{n} \sum_{i=1}^n E_i$$

onde n é o número de iterações (10 para o 10-fold cross validation), e E_i é o valor da taxa de erro para a i -ésima iteração.

A mediana é obtida ordenando-se os valores de E_i e tomando o valor central se n é ímpar, ou se n é par a mediana é igual a média dos dois valores centrais.

O sistema também calcula o que denominamos média G, uma variação da média geométrica apropriada para situações nas quais o número de exemplos em cada uma das classes X e Y está muito desbalanceado.

O valor de E_i para a i -ésima iteração do método de resampling é calculado por

$$E_i = \sqrt{\frac{b}{a+b} \frac{c}{c+d}}$$

se $\frac{b}{a+b} \frac{c}{c+d} > 0$, caso contrário

$$E_i = \sqrt{\max(\frac{b}{a+b}, \frac{c}{c+d})}$$

Finalmente, a média G é calculada como a média aritmética desses valores de E_i

$$M_G = \frac{1}{n} \sum_{i=1}^n E_i$$

O sistema SMEC calcula também os desvios padrões para cada uma dessas médias. Os desvios padrões fornecem um índice do espalhamento ou dispersão das médias, calculados pela equação

$$s = \sqrt{\frac{\sum_{i=1}^n (E_i - M)^2}{n}}$$

onde M é o valor médio para os E_i , isto é M_a ou M_G .

7.7 Resultados Experimentais

Nesta seção são apresentados os resultados das medidas realizadas através do ambiente AMPSAM. Os cálculos estatísticos, como média e variância, foram feitos utilizando o sistema SMEC, assim como parte dos gráficos ilustrativos.

A tabela 7.4 mostra os resultados obtidos com o conjunto de exemplos Hepatitis. Essa tabela apresenta os seguintes resultados

- 1º **Coluna** Nome do algoritmo;
- 2º **Coluna** A matriz de confusão média para as 10 iterações. Esta matriz é encontrada calculando-se o número médio de exemplos para cada um dos elementos da matriz de confusão;
- 3º **Coluna** A taxa de erro para a classe X . São mostradas a média aritmética (coluna *Média*) e a mediana (coluna *Mediana*) e, para cada média é mostrado o desvio padrão relacionado. São apresentadas também a menor e a maior taxas de erro entre as 10 iterações (*[Min, Max]*);
- 4º **Coluna** A taxa de erro para a classe Y . Idem à 3º coluna;
- 5º **Coluna** A taxa de erro para todo o conjunto de exemplos. Idem à 3º coluna;
- 6º **Coluna** A taxa de erro para todo o conjunto de exemplos calculada através da média G adaptada. São apresentados também os desvios padrões e os valores máximos e mínimos da média G nas 10 iterações.

Algoritmo	Matriz Confusão	$\frac{b}{a+b}$		$\frac{c}{c+d}$		$\frac{b+c}{a+b+c+d}$		$\sqrt{\frac{b}{a+b} \frac{c}{c+d}}$
		Média	Mediana	Média	Mediana	Média	Mediana	Média G
CN2	0.8 2.4 0.3 12.0	80.0 ± 19.5	80.0 ± 19.5	2.4 ± 5.0	0.0 ± 5.5	17.3 ± 9.0	16.0 ± 9.1	13.6 ± 10.1
	<i>[Min, Max]</i>	[50.0, 100.0]		[0.0, 14.3]		[6.2, 33.3]		[7.1, 37.8]
NewId	0.9 2.3 0.8 11.5	71.6 ± 21.8	69.4 ± 21.9	6.6 ± 6.4	3.7 ± 7.0	20.0 ± 7.8	20.4 ± 7.8	18.9 ± 9.1
	<i>[Min, Max]</i>	[38.0, 100.0]		[0.0, 16.2]		[8.1, 33.3]		[8.9, 34.5]
FOIL(c1)	1.1 2.1 1.2 11.1	65.5 ± 30.5	67.5 ± 30.7	9.3 ± 7.7	9.1 ± 7.7	21.4 ± 7.1	22.5 ± 7.2	20.1 ± 12.2
	<i>[Min, Max]</i>	[0.0, 100.0]		[0.0, 21.4]		[6.7, 31.2]		[4.6, 46.3]
FOIL(c2)	2.0 1.2 1.7 10.6	38.5 ± 28.3	40.0 ± 28.3	12.5 ± 11.0	15.4 ± 11.4	18.6 ± 9.1	16.7 ± 9.2	17.6 ± 16.3
	<i>[Min, Max]</i>	[0.0, 100.0]		[0.0, 28.6]		[6.2, 31.2]		[3.9, 51.7]

Tabela 7.4: Taxas de Erro Médias para o Conjunto de Exemplos Hepatitis

Pela Tabela 7.4 pode-se verificar que os sistemas de aprendizado são demasiadamente imprecisos na classificação dos exemplos pertencentes à classe X . Um dos motivos que podem ter contribuído para este problema é o desbalanço entre as classes X (20,65%) e a classe Y (79,35%). Provavelmente, técnicas como as propostas em [Kubat 97] podem ajudar a melhorar o desempenho desses algoritmos neste conjunto de exemplos. O algoritmo FOIL que, a princípio, não foi desenvolvido para o aprendizado a partir de exemplos proposicionais obteve um bom desempenho, entretanto a variância da taxa de erro para a classe X é maior que a dos algoritmos proposicionais. Esta variância pode ser verificada através das taxas de erro mínimas e máximas ($[0, 100]$) obtidas pelo algoritmo FOIL — 3º coluna da Tabela 7.4.

O Figura 7.3 mostra graficamente as taxas de erro médias para os algoritmos analisados, assim como a taxa de erro padrão, a qual é a taxa de erro que seria obtida pelo classificador caso este respondesse sempre com a classe de maior frequência. Apesar do algoritmo CN2 ter tido um desempenho muito ruim na classificação de exemplos pertencentes à classe X , seu desempenho médio geral teve uma grande melhora graças ao seu excelente desempenho na classificação da classe Y , classe majoritária.

Figura 7.3: Taxas de Erro Médias para o Conjunto de Exemplos Hepatitis

De forma análoga, a Tabela 7.5 mostra os resultados obtidos para o conjunto de exemplos Breast Cancer, e a Figura 7.4 ilustra graficamente as taxas de erro médias para os algoritmos de aprendizado analisados.

A Figura 7.4 mostra que o desempenho dos algoritmos de aprendizado para este conjunto de exemplos foi muito ruim. De uma forma geral, os sistemas obtiveram um desempenho inferior ao classificador que sempre responde com a classe majoritária, ou seja, suas taxas de erro foram superiores à taxa de erro padrão.

Uma possível explicação para o mau desempenho dos sistemas de aprendizado nos

Algoritmo	Matriz Confusão	$\frac{b}{a+b}$		$\frac{c}{c+d}$		$\frac{b+c}{a+b+c+d}$		$\sqrt{\frac{b}{a+b} \frac{c}{c+d}}$
		Média	Mediana	Média	Mediana	Média	Mediana	Média G
CN2	2.2 6.2 2.7 17.4	74.4 ± 7.9	73.2 ± 8.0	13.4 ± 6.7	10.8 ± 7.2	31.4 ± 4.6	30.4 ± 4.7	30.5 ± 7.9
[Min, Max]		[60.0, 87.5]		[4.8, 23.8]		[24.1, 39.3]		[18.3, 41.2]
NewId	3.8 4.6 3.8 16.0	54.8 ± 11.2	56.4 ± 11.3	19.0 ± 7.0	18.3 ± 7.0	29.9 ± 5.1	28.6 ± 5.3	31.2 ± 7.4
[Min, Max]		[40.0, 75.0]		[5.3, 28.6]		[24.1, 37.9]		[17.8, 42.3]
FOIL(c1)	3.8 4.6 3.9 16.2	54.9 ± 12.2	56.4 ± 12.3	19.5 ± 10.3	20.1 ± 10.3	30.0 ± 10.1	31.9 ± 11.7	31.6 ± 10.2
[Min, Max]		[37.5, 71.4]		[4.8, 38.9]		[13.2, 50.0]		[13.4, 52.2]
FOIL(c2)	4.4 4.0 5.8 14.3	48.0 ± 9.6	52.8 ± 10.8	28.5 ± 8.3	27.5 ± 8.4	34.4 ± 5.9	35.7 ± 6.0	36.3 ± 6.2
[Min, Max]		[30.0, 57.1]		[15.8, 45.5]		[25.0, 44.8]		[25.8, 44.2]

Tabela 7.5: Taxas de Erro Médias para o Conjunto de Exemplos Breast Cancer

Figura 7.4: Taxas de Erro Médias para o Conjunto de Exemplos Breast Cancer

conjunto de exemplos Hepatitis e Breast Cancer pode ser encontrada na literatura. Segundo [Holte 93] muito poucos sistemas de aprendizado obtiveram sucesso em encontrar regras de qualquer tipo que tivessem uma precisão superior a dois pontos percentuais acima da *precisão padrão*⁵. Holte procurou na literatura por artigos que relatassem desempenhos de sistemas de aprendizado para alguns conjuntos de exemplos (incluindo Hepatitis e Breast Cancer). Apesar que muitos dos artigos encontrados não relatarem precisamente como tais índices de performance foram encontrados, esses valores podem dar uma idéia geral da grande variação das medidas de desempenho obtidas sobre esses conjuntos de exemplos. No caso do conjunto de exemplos Hepatitis e Breast Cancer foram encontrados, respectivamente, 33 e 75 índices de performance, cujos intervalos e

⁵Baseline accuracy, é o equivalente em precisão à taxa de erro padrão, ou seja, precisão padrão = 100% - erro padrão.

freqüências são mostrados na Tabela 7.6.

Hepatitis		Breast Cancer	
Taxa de Erro (%)	Freqüência	Taxa de Erro (%)	Freqüência
62..62.9	1	38..38.9	2
29..29.9	1	35..35.9	5
24..24.9	1	34..34.9	5
23..23.9	2	33..33.9	2
22..22.9	1	32..32.9	8
21..21.9	6	31..31.9	6
20..20.9	3	30..30.9	4
19..19.9	3	29..29.9	12
18..18.9	2	28..28.9	15
17..17.9	6	27..27.9	4
16..16.9	4	26..26.9	1
15..15.9	3	25..25.9	2
		24..24.9	2
		23..23.9	4
		22..22.9	3
Total	33	Total	75

Tabela 7.6: Freqüências de Desempenhos por Intervalo (Criado a Partir de Informações em [Holte 93])

A Figura 7.5 mostra o histograma correspondente às medidas de desempenho do conjunto de dados Hepatitis, cuja taxa de erro padrão é de 20.65%. Pode-se verificar que apenas 15 medidas publicadas — menos de 50% — possuem taxa de erro média 2 pontos percentuais (menos que 18.65%) abaixo do erro padrão.

Analogamente, a Figura 7.6 mostra o histograma correspondente às medidas de desempenho do conjunto de exemplos Breast Cancer, cuja taxa de erro padrão é 29.47%. Este caso é ainda pior que o anterior já que somente 16 das performances publicadas — menos de 22% — possuem taxas de erro médias 2 pontos percentuais (menos que 27.47%) abaixo do erro padrão.

A Tabela 7.7 resume os resultados obtidos pelos sistemas de aprendizado analisados para o conjunto de exemplos Pima, Bupa, Hungarian e Credit. Nessa tabela são mostradas as médias aritméticas das taxas de erro e seus respectivos desvios padrões. As tabela com resultados mais detalhados podem ser encontradas no Apêndice A.

A Figura 7.7 mostra graficamente as taxas de erro médias obtidas para o conjunto de exemplos Pima — Tabela B.1, pg. 143. O conjunto de exemplos Pima foi primeiramente utilizado por sistemas de aprendizado de máquina em [Smith 88], nos quais foi utilizado o método holdout para estimar a taxa de erro verdadeira. Através desse método de treinar e testar foi encontrada uma taxa de erro de aproximadamente 24%, inferior às taxas de erro médias obtidas pelos sistemas de aprendizado analisados pelo ambiente AMPSAM. O conjunto de exemplos Pima apresenta uma grande diferença

Figura 7.5: Histograma de Taxas de Erro para Hepatitis (Criado a Partir de Informações em [Holte 93])

Algoritmo	Pima	Bupa	Hungarian	Credit
CN2	28.2 ± 5.5	33.6 ± 8.3	21.9 ± 6.7	14.2 ± 3.4
NewId	29.7 ± 6.3	35.8 ± 8.5	24.6 ± 9.1	14.7 ± 2.9
FOIL(c1)	29.7 ± 4.5	34.4 ± 7.8	21.2 ± 5.3	17.5 ± 3.5
FOIL(c2)	28.9 ± 5.2	34.3 ± 6.8	22.1 ± 5.2	24.0 ± 5.2

Tabela 7.7: Médias Aritméticas das Taxas de Erro para os Conjuntos de Exemplos Pima, Bupa, Hungarian e Credit

de performance a favor da classe majoritária. Existe aproximadamente 1.9 exemplo da classe X para cada exemplo da classe Y .

A Figura 7.8 mostra graficamente as taxas de erro médias obtidas para o conjunto de exemplos Bupa — Tabela B.2, pg. 144. Se comparado com a taxa de erro padrão, os sistemas analisados obtiveram um bom desempenho para este conjunto de exemplos. Para todos eles a performance na classe Y foi superior à classe X , sendo que nesse conjunto de exemplos existe uma relação de 1.3 exemplo da classe Y para cada exemplo da classe X .

A Figura 7.9 mostra graficamente as taxas de erro médias obtidas para o conjunto de exemplos Hungarian — Tabela B.3, pg. 144. Esse conjunto de exemplos foi primeiramente utilizado em [Detrano 89] no qual foi obtida uma taxa de erro de 23%, ligeiramente superior às taxas de erro encontradas pelos algoritmos de aprendizado analisados com o ambiente AMPSAM. Mais uma vez existe uma grande diferença de performance entre as classes X e Y . A relação entre o número de exemplos da classe X e Y é de 1.7 exemplo da classe Y para cada exemplo da classe X .

Figura 7.6: Histograma de Taxas de Erro para Breast Cancer (Criado a Partir de Informações em [Holte 93])

A Figura 7.10 mostra graficamente as taxas de erro médias obtidas para o conjunto de exemplos Credit — Tabela B.4, pg. 145. Esse conjunto de exemplos é o que apresenta a menor desproporção entre exemplos de cada classe, mas mesmo assim a majoritária apresentou uma performance largamente superior, para uma diferença entre as classes de aproximadamente 1.3 exemplo da classe Y para cada exemplo da classe X .

Aparentemente os sistemas de aprendizado analisados são altamente sensíveis à desproporção existente entre as classes. Para todos os sistemas analisados houve uma diferença de performance visivelmente superior a favor da classe majoritária para todos os conjuntos de exemplos selecionados. Mesmo para o conjunto de exemplos Credit, o qual apresenta a menor desproporção entre as classes (1.25 caso da classe Y para cada caso da classe X), a diferença de performance foi notável.

Entre todos os algoritmos, o CN2 é, aparentemente, o algoritmo mais sensível à desproporção entre as classes, também confirmado pelos valores da Média G . Com exceção do conjunto de exemplos Hungarian, o CN2 apresenta repectivamente as melhores performances médias (aritmética e mediana) para a classe majoritária, e as piores performances médias para a classe minoritária entre todos os sistemas analisados. No caso do conjunto de exemplos Hungarian, o algoritmo CN2 só tem uma taxa de erro menor na classe minoritária que o algoritmo NewId.

Para os conjuntos de exemplos Pima, Bupa, Hungarian e Credit, as taxas de erro para os sistemas de apredizado analisados estão bem abaixo da taxa de erro padrão. Entretanto, essas taxas de erro podem ser consideradas muito altas para muitas aplicações reais. Por exemplo, o algoritmo CN2 apresentou a melhor performance para o conjunto de

Figura 7.7: Taxas de Erro Médias para o Conjunto de Exemplos Pima

exemplos Credit. Entretanto, nesse caso a taxa de erro é de 14.2%, indicando que em quase 15% dos pedidos analisados será fornecido crédito a uma pessoa que não irá cumprir com o contrato, ou será negado crédito a uma pessoa capaz de cumprí-lo. Possivelmente, as performances obtidas neste trabalho não são as performances ótimas dos algoritmos analisados para os conjuntos de exemplos selecionados, uma vez que esses sistemas foram treinados utilizando os parâmetros padrões de seus modelos. Desta forma, análises com resultados mais definitivos podem ser conseguidas se os parâmetros do modelo ótimos forem estimados.

A Figura 7.11 mostra a taxa de erro aparente e a taxa de erro obtida através do método 10-fold cross validation para o algoritmo CN2. Como pode ser verificado, para todos os conjuntos de exemplos, a taxa de erro aparente foi vários pontos percentuais inferior à taxa de erro obtida pelo método 10-fold cross validation, mostrando que a taxa de erro aparente é um estimador muito mais otimista que o método 10-fold cross validation. Apesar de que a taxa de erro obtida através de um método de resampling seja apenas uma estimativa da taxa de erro verdadeira, esta estimativa é muito menos tendenciosa que a obtida através do cálculo da taxa de erro aparente.

7.8 Problemas Comuns na Avaliação de Performance

O processo de avaliação de performance de sistemas de aprendizado pode levar a resultados indesejados. Frequentemente, as causas destes maus resultados tem origem conhecida. Nesta seção são discutidos alguns fatores que podem levar a taxas de erro verdadeiras muito altas ou tendenciosas, e como estes problemas podem ser soluciona-

Figura 7.8: Taxas de Erro Médias para o Conjunto de Exemplos Bupa

dos.

7.8.1 Baixa Precisão Relacionada com Conjuntos de Exemplos Ruins

Os sistemas de aprendizado proposicionais aprendem conceitos utilizando como única fonte de informação os exemplos apresentados para treinamento. Por sua vez, estes exemplos possuem um formato muito simples, são constituídos de atributos e uma classe associada. Desta forma, não se pode esperar que o sistema de aprendizado realize “milagres” se foram cometidos erros durante o processo de coleta dos exemplos.

Um dos problemas relacionados com a qualidade do conjunto de exemplos é a capacidade de predição dos atributos, ou seja, se os atributos podem contribuir efetivamente para a determinação da classe. A fim de tentar solucionar este problema, deve-se reservar a escolha dos atributos que irão compor os exemplos a um especialista no domínio.

Outro problema relacionado aos conjuntos de exemplos é quanto à qualidade das medidas dos atributos e da classe. É possível que os atributos tenham sido mal medidos, ou seja, os valores presentes no conjunto de exemplos não correspondem aos valores reais. Como prováveis fontes deste problema, pode-se citar o desleixo dos responsáveis por registrar estas medidas, ou da falta de precisão do dispositivo responsável pela medida.

Mesmo com um conjunto de exemplos com problemas como estes, é possível criar um classificador que apresente taxa de erro aparente baixa, através de overfitting. Entretanto, sob estas condições o classificador não será capaz de classificar corretamente novos casos, e a taxa de erro verdadeira será alta. Desta forma, os métodos de resampling são úteis por não serem facilmente “enganados” por falhas presentes nos exemplos.

Figura 7.9: Taxas de Erro Médias para o Conjunto de Exemplos Hungarian

7.8.2 Erros Cometidos no Processo de Avaliação

Existem dois problemas que podem ocorrer no processo de avaliação que podem levar a resultados incorretos. Estes problemas são chamados de *testar com os exemplos de treinamento* e *treinar com os exemplos de teste*.

O primeiro, testar com os exemplos de treinamento, já foi discutido e resulta na taxa de erro aparente, a qual é uma aproximação extremamente otimista da taxa de erro verdadeira.

O segundo, treinar com os exemplos de teste, pode ocorrer, por exemplo, quando se deseja avaliar melhor a complexidade do modelo de um sistema de aprendizado para um determinado conjunto de exemplos. Como já foi dito anteriormente, a complexidade do modelo pode ser estimada com o auxílio dos métodos de resampling. Entretanto é um erro estimar a complexidade do modelo utilizando todo o conjunto de exemplos e, uma vez determinada a complexidade do modelo, voltar a utilizar o conjunto de exemplos para estimar a taxa de erro verdadeira. Claramente, a estimativa da taxa de erro vai se tornar tendenciosamente otimista, uma vez que os exemplos de teste foram utilizados para determinar a complexidade do modelo.

Um exemplo no qual este problema pode ocorrer é durante a poda de uma árvore de decisão. A poda é geralmente realizada após o treinamento, já com a árvore criada. Um conjunto de casos diferentes do treinamento são utilizados para verificar quais ramos podem ser podados. Entretanto é um erro utilizar os mesmos exemplos para podar a árvore e posteriormente medir a taxa de erro. Esta abordagem torna a estimativa da taxa de erro otimista. Uma melhor abordagem é utilizar parte dos

Figura 7.10: Taxas de Erro Médias para o Conjunto de Exemplos Credit

exemplos de treinamento para a poda da árvore de decisão. Desta forma, trabalhos como [Salzberg 97] recomendam que o conjunto de exemplos seja dividido em três sub-conjuntos: treinamento, teste e *conjunto de validação* o qual é utilizado para estimar a melhor complexidade do modelo.

Apesar destas diretrizes parecerem óbvias, muitos artigos não seguem tais metodologias. Por exemplo, o artigo [Flexer 96] apresenta um panorama das avaliações experimentais realizadas em trabalhos publicados por algumas das principais revistas de Redes Neurais. Neste artigo é mostrado que apenas 4,9% (3 trabalhos), dos 61 trabalhos analisados, utilizaram um conjunto de validação para ajustar os parâmetros do modelo e selecionar melhor a complexidade do modelo. Nos demais trabalhos não fica claro como os autores fizeram tais ajustes. Entretanto, quaisquer decisões a respeito da análise experimental, como por exemplo os ajustes de parâmetros e a seleção da complexidade do modelo, devem ser tomadas antes de medir a taxa de erro sobre o conjunto de teste, sob o risco de invalidar os resultados obtidos. A Tabela 7.8.2 resume alguns resultados encontrados em [Flexer 96].

Requisito	Sim	Não	Não Mencionado
Conj. Diferentes para Treinamento e Teste	72,2%	1,6%	26,2%
Múltiplas Execuções	57,3%	36,1%	6,6%
Conjunto de Validação	4,9%	0,0%	95,1%
Média, Variância e Int. de Confiança	27,7%	72,3%	0,0%

Tabela 7.8: Requisitos Presentes em Publicações de Avaliação Experimental em Redes Neurais [Flexer 96]

Por fim, um outro erro que pode ser cometido é quando um conjunto de exemplos pode

Figura 7.11: Médias Aritméticas das Taxas de Erro Aparente e das Taxas de Erro do 10-CV para o CN2

ser considerado de tamanho médio, mas existe um desbalanceamento muito grande entre as classes. Por exemplo para um conjunto de exemplos de 200 casos é recomendável a utilização de um método de cross-validation. Entretanto, se a maioria dos casos pertencem a uma determinada classe e apenas 20 ou 30 casos à outra classe, e se o objetivo do classificador é classificar corretamente os casos da classe minoritária, então é aconselhável utilizar algum método mais robusto a fim de estimar a taxa de erro para este conjunto de exemplos. Neste caso seria aconselhável utilizar o método bootstrap.

7.9 Qual Sistema de Aprendizado é Melhor?

Sob o aspecto de medir com confiabilidade a performance de sistemas de aprendizado, a comunidade de Aprendizado de Máquina evoluiu muito nos últimos anos. Atualmente, é praticamente inadmissível encontrar trabalhos que relatem performances utilizando somente a taxa de erro aparente. Também, muito se tem discutido a respeito de como comparar as performances de dois sistemas de aprendizado para dizer qual é o melhor sistema em uma determinada aplicação, mas aparentemente ainda não existe um consenso a respeito de qual método estatístico deve ser utilizado.

Em muitos trabalhos na área de Aprendizado de Máquina, os resultados dos experiments realizados são mostrados na forma de um “ranking”. Entretanto uma pergunta muito freqüente é

Existe uma diferença estatisticamente significativa entre os resultados apre-

sentados ?

Ou seja,

Dados os resultados das medidas realizadas é possível afirmar, com segurança, que um certo sistema de aprendizado é superior a outro em um determinado conjunto de exemplos?

Para responder esta pergunta deve-se utilizar algum método estatístico que compute a significância da diferença entre médias. Entretanto, segundo [Feelders 95] encontrar um procedimento para comparar dois ou mais sistemas de aprendizado não é uma tarefa simples e pode requerer mais do que um conhecimento básico em estatística.

Atualmente, o teste de significância conhecido como *resampled paired t test* [Hildebrand 86] é um dos mais difundidos para comparar performances de sistemas de aprendizado. Ele consiste de aplicar o método random subsampling por aproximadamente 30 vezes, e utilizar o teste *t* de Student para verificar se existe uma diferença significativa entre as taxas de erro médias. O teste *t* de Student pode ser calculado da seguinte forma: dados dois algoritmos *A* e *B*, onde $p_A^{(i)}$ (respectivamente, $p_B^{(i)}$) é a proporção de exemplos de teste classificados incorretamente pelo classificador gerado por *A* (respectivamente, *B*) durante a *i*-ésima iteração do método random subsampling. Seja $p^{(i)} = p_A^{(i)} - p_B^{(i)}$ e $\bar{p} = \frac{1}{n} \sum_{i=1}^n p^{(i)}$, então o teste *t* de Student pode ser calculado pela equação

$$t = \frac{\bar{p}\sqrt{n}}{\sqrt{\frac{\sum_{i=1}^n (p^{(i)} - \bar{p})^2}{n-1}}}$$

Supondo que sejam realizadas 30 iterações, e se deseja verificar que as duas performances médias sejam diferentes com 95% de confiança, então deve-se encontrar em uma tabela de probabilidades para a distribuição *t* de Student o valor de

$$t_{29,0.975} = 2.04523$$

Desta forma, a diferença entre as médias é significativa, segundo este teste, sempre que

$$|t| > t_{29,0.975}$$

ou seja,

$$|t| > 2.04523$$

Apesar deste método ser altamente difundido na comunidade de Aprendizado de Máquina, muitas críticas estão sendo feitas a respeito de sua confiabilidade quando aplicado a problemas desta área. Segundo [Dietterich 97], este teste possui uma alta probabilidade de apontar uma diferença significativa entre as médias, quando na realidade não existe diferença alguma — erro do tipo I. Dietterich é ainda enfático em dizer que este método nunca deve ser utilizado para problemas de classificação em Aprendizado de Máquina devido à sua baixa confiabilidade.

Outros testes mais robustos utilizam quatro valores diferentes para o cálculo da significância da diferença entre as médias, são eles

Número de exemplos classificados incorretamente por ambos A e B	Número de exemplos classificados incorretamente por A , mas não por B
Número de exemplos classificados incorretamente por B , mas não por A	Número de exemplos classificados corretamente por A e B

Para esses valores é utilizada a notação

n_{00}	n_{01}
n_{10}	n_{11}

Um teste que utiliza essas medidas é conhecido como *Teste de McNemar*. Este teste utiliza a distribuição χ^2 para calcular a significância da diferença entre as médias. Esse teste pode ser calculado da seguinte forma

$$m = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}}$$

Segundo este teste pode-se afirmar com 95% de confiabilidade que as performances dos algoritmos A e B são diferentes quando o valor $m > \chi^2_{1,0.95}$. Segundo uma tabela de distribuição de probabilidades para χ^2 , $\chi^2_{1,0.95} = 3.841459$. Desta forma, existe diferença de performance entre A e B se

$$m > 3.841459$$

Um outro teste de significância é descrito em [Salzberg 97] e é chamado de *teste da binomial*. Para entender melhor este teste suponha que um algoritmo A obteve em uma avaliação experimental uma performance superior a um outro algoritmo B . Sabendo que n_{10} exemplos foram classificados corretamente por A , mas não por B , este teste busca responder à pergunta

Supondo que os algoritmos A e B possuem a mesma performance, qual é a probabilidade de que A “vença” o algoritmo B pelo menos por n_{10} vezes?

Esta probabilidade pode ser calculada utilizando a distribuição binomial. Estatisticamente, deseja-se calcular a probabilidade

$$P(X \geq n_{10} | p = 0.5)$$

onde p é a probabilidade de A ser melhor que B , $p = 0.5$ pois está-se supondo que ambos algoritmos possuem a mesma performance.

Esta probabilidade pode ser calculada da seguinte forma

$$\sum_{s=n_{10}}^{n_{10}+n_{01}} \frac{n!}{s!(n-s)!} (0.5)^n$$

Como dito anteriormente, não existe um consenso a respeito de qual método estatístico deve ser utilizado em experimentos da área de Aprendizado de Máquina. Uma das possíveis abordagens para comparar a performance de sistemas de aprendizado é proposta em [Salzberg 97]. Esta abordagem consiste nos seguintes passos

1. Selecione os algoritmos e os conjuntos de exemplos que farão parte do experimento;
2. Divida o conjunto de exemplos em k partições para utilizar o método cross validation. Tipicamente $k = 10$, mas valores maiores para k podem ser utilizados para conjuntos de exemplos menores;
3. Execute o método cross validation da seguinte forma
 - (a) Divida o conjunto de treinamento T em dois subconjuntos $T1$ e $T2$. $T1$ será utilizado para treinamento e $T2$ será utilizado para ajustar os parâmetros e a complexidade do modelo;
 - (b) Quando os parâmetros e o modelo foram otimizados, treine o sistema de aprendizado no conjunto $T = T1 \cup T2$;
 - (c) Meça a performance no conjunto de teste;
 - (d) A performance final é a média em todas as k partições e, esses k valores fornecem uma estimativa da variância dos algoritmos.
4. Para comparar os algoritmos, utilizar o teste da binomial ou o teste de McNemar.

7.10 Considerações Finais

Nesta seção foi realizado um estudo experimental a respeito do desempenho dos algoritmos CN2, NewId e FOIL sobre diversos conjuntos de exemplos reais. Este experimento

teve como principal objetivo avaliar o projeto do ambiente AMPSAM aplicado a sistemas de aprendizado conhecidos e a conjuntos de exemplos reais. Foi utilizado também o sistema SMEC, o qual utiliza os resultados fornecidos pelo ambiente AMPSAM para calcular e mostrar graficamente algumas medidas descritivas mais importantes relacionadas com tendência central e dispersão dos dados. Ainda que este capítulo tenha brevemente discutido testes estatísticos para comparar performances, não era objetivo deste trabalho realizar tais comparações, uma vez que ainda não existe uma metodologia bem definida a este respeito.

A falta de uma análise matemática que possa determinar qual método de aprendizado irá desempenhar bem sobre um conjunto de exemplos faz com que os estudos experimentais tenham uma importância central na área de Aprendizado de Máquina. Tal importância deve ser refletida na seriedade das execuções desses estudos e forma correta de relatar os resultados. Esses resultados devem ser descritos detalhadamente e fim de que seja possível que outros pesquisadores possam repetir esses estudos.

Com a realização dos experimentos apresentados neste capítulo foi possível verificar que o ambiente AMPSAM é capaz auxiliar no processo de avaliação, automatizando tarefas repetitivas que podem dar margem à introdução de erros. Sob este aspecto, as ferramentas sintáticas foram importantes, pois permitiram que os arquivos fossem automaticamente convertidos para as sintaxes exigidas.

Capítulo 8

Conclusões

Um sistema de aprendizado supervisionado é um programa capaz de realizar decisões baseado na experiência contida em casos resolvidos com sucesso. As regras de classificação induzidas por um sistema de aprendizado podem ser analisadas segundo dois critérios: a complexidade dessas regras e o erro de classificação sobre um conjunto independente de exemplos.

Muitas vezes, complexidade e taxa de erro na classificação de novos casos estão diretamente relacionados. Regras de classificação demasiadamente complexas podem indicar que o conhecimento contido nos exemplos não foi generalizado, em um processo conhecido como overfitting. Sistemas de aprendizado utilizam diferentes métodos para criar regras de classificação simples.

Os sistemas de aprendizado simbólico proposicionais, os quais podem ser caracterizados pelo uso da linguagem atributo-valor para descrever os exemplos, possuem teoria e métodos bem definidos. Muitos desses sistemas utilizam árvores ou regras de decisão para descrever os conceitos induzidos. Nesses sistemas é comum a aplicação de técnicas que buscam evitar que os conceitos induzidos sejam demasiadamente complexos; a estas técnicas dá-se o nome de bias para evitar overfitting. Uma das técnicas mais conhecidas para evitar o overfitting é conhecida como poda. Esta técnica busca eliminar nós — em árvores de decisão — ou condições — em regras de decisão — pouco relevantes ao conceito induzido.

Os sistemas de aprendizado proposicionais, apesar de amplamente difundidos, possuem severas restrições aos conceitos que podem ser induzidos. A expressividade da linguagem atributo-valor permite que somente conceitos que podem ser expressos na linguagem lógica proposicional possam ser aprendidos. Uma tentativa de tentar ampliar alguns desses limites é através de indução construtiva.

A idéia central em indução construtiva é construir novos atributos, em função de atributos primitivos que descrevem os exemplos, tal que esses novos atributos facilitem a

representação do conceito. A seleção desses novos atributos é considerada uma arte guiada pela intuição do analista e pela exploração dos dados.

Entretanto, ainda que o uso de indução construtiva possa ampliar a linguagem de descrição dos exemplos, facilitando eventualmente a representação do conceito, as limitações mais severas dos sistemas proposicionais, como a utilização de conhecimento de fundo e a possibilidade de expressar relações entre objetos, só podem ser superadas quando é utilizada uma linguagem mais expressiva, tal como a lógica de primeira ordem. Os sistemas de aprendizado relacionais, tais como os sistemas de Programação Lógica Indutiva, utilizam a lógica de primeira ordem como linguagem de descrição de exemplos, hipóteses e conhecimento de fundo. Esta é uma área de pesquisa ativa em Aprendizado de Máquina, e grande parte de sua teoria e métodos ainda estão sendo formulados. Um dos principais problemas que envolve o aprendizado relacional reside na alta expressividade da lógica de primeira ordem, o que implica em um espaço de hipóteses muito grande. Assim, muitos dos sistemas de aprendizado relacional buscam restringir tal expressividade a fim de tornar o aprendizado factível.

Além da complexidade dos conceitos induzidos, os sistemas de aprendizado podem ser avaliados pelo erro de classificação em novos casos. Neste caso estudos experimentais são necessários, uma vez que não existe uma análise matemática que possa determinar se um algoritmo de aprendizado irá desempenhar bem em um conjunto de exemplos.

Entre as análises matemáticas que podem prover resultados de aplicação prática, encontra-se o modelo PAC — Probably Approximately Correct — provido pela teoria de aprendizado computacional. PAC é uma análise de pior caso. Independentemente da distribuição de probabilidade do conjunto de exemplos, PAC garante que os resultados de classificação serão corretos com uma pequena margem de erro. Apesar desta análise prover resultados teóricos interessantes, mesmo para classificadores simples os resultados podem indicar que um grande número de casos são necessários para garantir a performance estipulada.

Técnicas experimentais para estimar a taxa de erro verdadeira não somente provêm uma base para comparar objetivamente a performance de diversos algoritmos de aprendizado no mesmo conjunto de exemplos, mas também podem ser uma ferramenta poderosa para projetar um classificador. Métodos estatísticos são essenciais para uma análise experimental não tendenciosa. Entre os métodos estatísticos mais conhecidos estão os métodos de resampling, os quais são capazes de estimar a taxa de erro verdadeira de forma confiável mesmo para pequenas amostras de casos resolvidos. Os métodos de resampling são assim chamados pois consistem de repetidos experimentos de amostragem, treinamento e teste.

Neste trabalho foi projetado e implementado um ambiente chamado AMPSAM, com o objetivo de ajudar na aplicação dos métodos de resampling a conjuntos de exemplos reais. Para permitir que o ambiente seja extensível a novos métodos de resampling e novos sistemas de aprendizado, AMPSAM foi projetado como uma coleção de progra-

mas independentes os quais podem interagir entre si através de scripts pré-definidos ou de novos scripts criados pelo usuário. O ambiente utiliza um formato padrão para arquivo de exemplos, o qual é independente da sintaxe de qualquer algoritmo. AMPSAM também inclui ferramentas para particionar conjuntos de exemplos em conjuntos de treinamento e teste utilizando diferentes métodos. Além de holdout, que é o estimador de taxa de erro mais comum, AMPSAM suporta os métodos *n-fold cross-validation* — incluindo o *leaving-one-out* — e o método *bootstrap*.

Com o auxílio do ambiente AMPSAM foi realizado um estudo experimental, no qual foi estimada a taxa de erro de três algoritmos de aprendizado proposicionais e relacionais bem conhecidos: CN2, NewId e FOIL. Esses sistemas foram analisados utilizando conjuntos de exemplos reais. Através desse estudo foi possível verificar que o ambiente AMPSAM é eficaz na automatização das análises experimentais de sistemas de aprendizado simbólico. Foi possível verificar também que a estrutura modular (na forma de ferramentas) deste ambiente é uma abordagem adequada a este tipo de problema. Entretanto, experimentos futuros poderão dizer se as ferramentas projetadas são suficientes, ou se será necessário adicionar ferramentas com novas funcionalidades ao ambiente AMPSAM.

Como trabalhos futuros pretende-se reescrever as ferramentas sintáticas na linguagem de programação Perl, bem como criar uma interface gráfica para o ambiente, a fim de tornar a sua utilização mais simples. Pretende-se também investigar mais profundamente a teoria estatística com a finalidade de discutir métodos que possam contribuir em análises experimentais de algoritmos de aprendizado.

Apêndice A

Gráficos Gerados pelo Sistema SMEC

Neste apêndice são mostrados alguns dos gráficos criados pelo sistema SMEC ¹. Esses gráficos foram criados a partir dos dados obtidos na análise experimental discutida no Capítulo 7. Através do sistema SMEC o usuário pode requerer uma grande variedade de gráficos para os diferentes algoritmos e conjuntos de exemplos utilizados no experimento. Os gráficos foram gerados utilizando os dados referentes ao algoritmo de aprendizado CN2, utilizando o conjunto de exemplos Hepatitis.

Figura A.1: Média Aritmética (○) e Média Geométrica (+) para cada Iteração e para todo o Conjunto de Exemplos

Entre os gráficos que podem ser requeridos pelo usuário, estão os seguintes: a média

¹Neste sistema foi também adotado o Inglês pelos motivos antes expostos.

aritmética e a média G da taxa de erro em todo o conjunto de exemplos, juntamente com o valor dessas médias para cada iteração do método de resampling, como mostra a Figura A.1.

Este mesmo gráfico pode ser criado utilizando a média aritmética e a mediana, como mostra a Figura A.2.

Figura A.2: Média Aritmética e Mediana para todas as Iterações e Média Aritmética (o) para cada Iteração

O sistema também fornece gráficos dos valores da taxa de erro para as classes X e Y . A Figura A.3 mostra a média aritmética e a mediana da taxa de erro para a classe X em todo o conjunto de exemplos, e para cada iteração do método de resampling.

Figura A.3: Taxa de Erro para a Classe X – Média Aritmética e Mediana para todas as Iterações e Média (o) para cada Iteração

A Figura A.4 mostra o mesmo gráfico da figura anterior, mas para a classe Y .

É possível também visualizar gráficos para a matriz de confusão de cada iteração do método de resampling — Figura A.5 — bem como da média dos elementos da matriz de confusão para todas as iteração — Figura A.6.

Pode ser visualizada também a taxa de erro porcentual para todo o conjunto de exemplos para cada iteração, bem como do erro cometido em cada uma das classes — Figura A.7.

Deve ser observado que os gráficos em Maple são coloridos e permitem ser visualizados de diversas formas pelo usuário, que interativamente pode trocar o estilo, cores, perspectiva, etc. Por exemplo, a Figura A.8 é o mesmo gráfico da Figura A.7 visualizada de uma perspectiva diferente.

Figura A.4: Taxa de Erro para a Classe Y – Média Aritmética e Mediana para todas as Iterações e Média ($\circ$) para cada Iteração

Figura A.5: Matriz de Confusão para uma Iteração

Figura A.6: Matriz de Confusão para todas as Iterações

Figura A.7: Taxa de Erro Porcentual para cada Iteração em todo o Conjunto de Exemplos e para cada Classe

Figura A.8: Outra Perspectiva da Figura A.7

Apêndice B

Resultados Complementares das Análises Experimentais

Este apêndice apresenta em maiores detalhes os resultados obtidos pelo ambiente AMP-SAM para os conjuntos de exemplos Pima, Bupa, Hungarian e Credit. Esses resultados são mostrados na forma de tabelas conforme o formato descrito na seção 7.7 pg. 119.

A Tabela B.1 mostra os resultados obtidos pelos sistemas de aprendizado para o conjunto de exemplos Pima.

Algoritmo	Matriz Confusão	$\frac{b}{a+b}$		$\frac{c}{c+d}$		$\frac{b+c}{a+b+c+d}$		$\sqrt{\frac{b}{a+b} \frac{c}{c+d}}$
		Média	Mediana	Média	Mediana	Média	Mediana	Média G
CN2	45.0 5.0 16.7 10.2	9.9 ± 3.4	9.2 ± 3.5	61.4 ± 12.2	60.1 ± 12.3	28.2 ± 5.5	30.6 ± 5.9	24.1 ± 4.4
[Min, Max]		[4.1, 17.6]		[41.2, 78.6]		[15.6, 35.1]		[17.9, 31.9]
NewId	39.3 10.7 12.1 14.8	21.3 ± 3.4	21.0 ± 3.4	44.0 ± 15.1	43.6 ± 15.1	29.7 ± 6.3	27.5 ± 6.7	30.0 ± 5.8
[Min, Max]		[16.3, 28.3]		[17.6, 69.7]		[22.1, 44.2]		[22.1, 41.7]
FOIL(c1)	38.4 11.6 11.2 15.7	23.1 ± 5.1	23.0 ± 5.1	40.3 ± 9.4	41.2 ± 9.4	29.7 ± 4.5	29.9 ± 4.5	30.2 ± 5.0
[Min, Max]		[16.4, 31.4]		[17.6, 53.3]		[21.1, 37.7]		[21.7, 39.6]
FOIL(c2)	40.4 9.6 12.5 14.4	19.4 ± 6.0	20.5 ± 6.1	46.6 ± 13.1	43.5 ± 13.5	28.9 ± 5.2	28.6 ± 5.2	28.9 ± 5.4
[Min, Max]		[8.3, 27.5]		[19.0, 70.6]		[21.1, 39.0]		[20.3, 40.1]

Tabela B.1: Taxas de Erro para o Conjunto de Exemplos Pima

Os resultados obtidos pelos sistemas de aprendizado analisados para o conjunto de exemplos Bupa são mostrados na Tabela B.2.

Algoritmo	Matriz Confusão	$\frac{b}{a+b}$		$\frac{c}{c+d}$		$\frac{b+c}{a+b+c+d}$		$\sqrt{\frac{b}{a+b} \frac{c}{c+d}}$
		Média	Mediana	Média	Mediana	Média	Mediana	Média G
CN2	4.9 9.6 2.0 18.0	66.4 ± 10.8	67.6 ± 10.9	10.4 ± 7.8	5.3 ± 9.4	33.6 ± 8.3	31.9 ± 8.5	24.9 ± 10.0
[Min, Max]		[42.9, 80.0]		[4.5, 26.7]		[20.6, 50.0]		[14.7, 42.8]
NewId	8.4 6.1 6.3 13.7	42.5 ± 10.2	42.5 ± 10.2	31.3 ± 9.2	32.6 ± 9.3	35.8 ± 8.5	33.9 ± 8.8	36.3 ± 8.8
[Min, Max]		[21.4, 57.1]		[20.0, 52.4]		[20.6, 54.3]		[20.7, 54.7]
FOIL(c1)	8.8 5.8 6.1 13.9	39.9 ± 14.6	35.1 ± 15.3	30.4 ± 8.4	28.7 ± 8.6	34.4 ± 7.8	35.3 ± 7.9	34.1 ± 7.5
[Min, Max]		[21.4, 73.3]		[20.0, 47.6]		[23.5, 48.6]		[23.1, 48.8]
FOIL(c2)	8.7 5.9 6.0 14.0	39.9 ± 8.8	41.5 ± 9.0	29.8 ± 11.4	31.8 ± 11.7	34.3 ± 6.8	33.9 ± 6.8	33.4 ± 7.4
[Min, Max]		[21.4, 56.2]		[11.1, 50.0]		[25.7, 47.1]		[21.4, 45.7]

Tabela B.2: Taxas de Erro para o Conjunto de Exemplos Bupa

A Tabela B.3 mostra os resultados obtidos pelos sistemas de aprendizado para o conjunto de exemplos Hungarian.

Algoritmo	Matriz Confusão	$\frac{b}{a+b}$		$\frac{c}{c+d}$		$\frac{b+c}{a+b+c+d}$		$\sqrt{\frac{b}{a+b} \frac{c}{c+d}}$
		Média	Mediana	Média	Mediana	Média	Mediana	Média G
CN2	6.1 4.5 1.9 16.9	44.3 ± 18.9	40.1 ± 19.3	10.5 ± 8.5	8.4 ± 8.9	21.9 ± 6.7	24.1 ± 7.1	18.1 ± 6.3
[Min, Max]		[7.7, 75.0]		[0.0, 29.4]		[10.0, 31.0]		[7.3, 27.7]
NewId	5.8 4.8 2.4 16.4	46.8 ± 18.7	49.2 ± 18.9	13.5 ± 9.2	12.1 ± 9.3	24.6 ± 9.1	27.6 ± 9.6	23.4 ± 8.6
[Min, Max]		[11.1, 84.0]		[0.0, 35.3]		[3.4, 34.5]		[3.3, 34.8]
FOIL(c1)	7.3 3.3 2.9 15.9	33.5 ± 19.5	29.2 ± 20.0	15.4 ± 8.8	13.6 ± 9.0	21.2 ± 5.3	22.0 ± 5.4	20.4 ± 6.8
[Min, Max]		[7.7, 75.0]		[6.2, 35.3]		[10.7, 27.6]		[11.1, 3.6]
FOIL(c2)	7.0 3.6 2.9 15.9	34.9 ± 12.5	35.9 ± 12.6	15.5 ± 7.5	15.5 ± 7.5	22.1 ± 5.2	21.5 ± 5.2	20.7 ± 5.4
[Min, Max]		[15.4, 53.3]		[5.6, 29.4]		[16.7, 33.3]		[16.0, 33.6]

Tabela B.3: Taxas de Erro para Conjunto de Exemplos Hungarian

Por fim, a Tabela B.4 mostra os resultados obtidos para o conjunto de exemplos Credit.

Algoritmo	Matriz Confusão	$\frac{b}{a+b}$		$\frac{c}{c+d}$		$\frac{b+c}{a+b+c+d}$		$\sqrt{\frac{b}{a+b} \frac{c}{c+d}}$
		Média	Mediana	Média	Mediana	Média	Mediana	Média G
CN2	6.1 4.5 1.9 16.9	44.3 ± 18.9	40.1 ± 19.3	10.5 ± 8.5	8.4 ± 8.9	21.9 ± 6.7	24.1 ± 7.1	18.1 ± 6.3
[Min, Max]		[7.7, 75.0]		[0.0, 29.4]		[10.0, 31.0]		[7.3, 27.7]
NewId	5.8 4.8 2.4 16.4	46.8 ± 18.7	49.2 ± 18.9	13.5 ± 9.2	12.1 ± 9.3	24.6 ± 9.1	27.6 ± 9.6	23.4 ± 8.6
[Min, Max]		[11.1, 84.0]		[0.0, 35.3]		[3.4, 34.5]		[3.3, 34.8]
FOIL(c1)	7.3 3.3 2.9 15.9	33.5 ± 19.5	29.2 ± 20.0	15.4 ± 8.8	13.6 ± 9.0	21.2 ± 5.3	22.0 ± 5.4	20.4 ± 6.8
[Min, Max]		[7.7, 75.0]		[6.2, 35.3]		[10.7, 27.6]		[11.1, 3.6]
FOIL(c2)	7.0 3.6 2.9 15.9	34.9 ± 12.5	35.9 ± 12.6	15.5 ± 7.5	15.5 ± 7.5	22.1 ± 5.2	21.5 ± 5.2	20.7 ± 5.4
[Min, Max]		[15.4, 53.3]		[5.6, 29.4]		[16.7, 33.3]		[16.0, 33.6]

Tabela B.4: Taxas de Erro para Conjunto de Exemplos Credit

Apêndice C

Aprendizado de Máquina na Web

Existem diversos sites na Web relacionados a Aprendizado de Máquina. A seguir são referenciados alguns que, na nossa opinião, destacam-se pela sua abrangência, apresentação e links para vários outros sites com informações relevantes.

Um primeiro site recomendado é o relacionado ao excelente livro sobre Inteligência Artificial de Russell e Norvig[Russell 95]. Neste site encontram-se, muito bem organizadas, informações sobre os temas tratados no livro, entre elas Aprendizado de Máquina.

- Prof. Stuart J. Russell
<http://www.cs.berkeley.edu/~russell/aima.html>

Um outro pesquisador que mantém uma lista muito grande de links para outros sites na Internet, relacionados com Aprendizado de Máquina é o Prof. David W. Aha.

- David W. Aha
<http://www.aic.nrl.navy.mil/~aha/research/machine-learning.html>

Existem também diversas páginas pessoais de pesquisadores conhecidos que mantêm links para artigos on-line. Segue uma listagem das páginas de alguns dos principais pesquisadores da área de Aprendizado de Máquina

- Ad Feelders
<http://diamond.ddi.nl/~ad/>
- Arthur Flexer
<http://www.ai.univie.ac.at/oefai/nn/arthur.html>
- Andy Pryke
<http://www.cs.bham.ac.uk/~anp/>

- Claude Sammut
<http://www.cse.unsw.edu.au/~claudio/>
- Cullen Schaffer
<http://louis.hunter.cuny.edu/faculty/schaffer/papers/list.html>
- David Jensen
<http://eksl-www.cs.umass.edu:80/~jensen/>
- Dennis F. Kibler
<http://www.ics.uci.edu/~kibler/>
- Johannes Fürnkranz
<http://www.ai.univie.ac.at/~juffi/>
- Lutz Prechelt
<http://wwwipd.ira.uka.de/~prechelt/>
- Pat Langley
<http://www.isle.org/~langley/>
- Peter Clark
<http://www.cs.utexas.edu/users/pclark/>
- Ronny Kohavi
<http://robotics.stanford.edu/users/ronnyk/ronnyk-bib.html>
- Ross Quinlan
<http://www.cs.su.oz.au/People/quinlan.html>
- Stephen Muggleton
<http://www.comlab.ox.ac.uk/oucl/groups/machlearn/steve.muggleton.html>
- Steven Salzberg
<http://www.cs.jhu.edu/~salzberg/home.html>
- Thomas G. Dietterich
<http://www.cs.orst.edu/~tgd/>

Existem também diversas páginas de grupos de pesquisa em Aprendizado de Máquina. Algumas delas são listadas a seguir

- University of California at Irvine
<http://www.ics.uci.edu/Al/ML/Machine-Learning.html>
- Oxford University Computing Laboratory
<http://www.comlab.ox.ac.uk/oucl/groups/machlearn/>

- Knowledge Systems Laboratory of the National Research Council of Canada
http://ai.iit.nrc.ca/home_page.html
<http://ai.iit.nrc.ca/bibliographies/>
- Austrian Research Institute for Artificial Intelligence
<http://www.ai.univie.ac.at/oefai/ml-resources.html>
- Carnegie Mellon University
<http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/learning/0.html>
- German National Research Center for Information Technology
<http://www.gmd.de/ml-archive>
- AI, CogSci and Robotics: Artificial Intelligence
<http://www-mice.cs.ucl.ac.uk/misc/ai/ai.html>
- Machine learning at The University of Wisconsin
<http://www.cs.wisc.edu/~shavlik/uwml.html>
- Machine Learning Library in C++
<http://www.sgi.com/Technology/mlc/>
- Machine Learning Project
<http://www.cs.waikato.ac.nz/ml/>
- The KAML Group, University of Ottawa
<http://www.csi.uottawa.ca/dept/kaml/KAML.html>

Por fim, algumas páginas com outros recursos a respeito de Aprendizado de Máquina

- OFAI Library Information System Biblio. Este é um sistema interessante pois permite que sejam feitas buscas de artigos, livros e relatórios técnicos da área de Inteligência Artificial por nome de autor e/ou título. O sistema apresenta links para alguns trabalhos que estão on-line
<http://www.ai.univie.ac.at/biblio.html>
- Journal of Artificial Intelligence Research
<http://www.cs.washington.edu/research/jair/home.html>
- Computer Science Technical Reports Archive Sites. Uma lista bastante completa de endereços eletrônicos de repositórios de relatórios técnicos de inúmeras universidades
<http://www.dai.ed.ac.uk/research/siteslist.html>
- Machine learning course pages around the country
<http://www.cs.iastate.edu/~honavar/Courses/cs673/machine-learning-courses.html>

- Yahoo Machine Learning
http://www.yahoo.com/Science/Computer_Science/Artificial_Intelligence/Machine_Learning/
- Machine Learning Online
http://mlis.www.wkap.nl/ml_links.htm

Entre as listas de Aprendizado de Máquina, destaca-se a lista moderada pelo Prof. M. Pazzani da University of California at Irvine. A frequência da lista é de aproximadamente 2 semanas. Pedido de inscrição deve ser feito para ml-request@ics.uci.edu enquanto que contribuições devem ser enviadas para ml@ics.uci.edu

Referências Bibliográficas

- [Aha 91] Aha, D.W.; Kibler, D.; Albert, M.K. *Instance-based Learning Algorithms*. Machine Learning 6, 1, 1991, pp. 37-66.
- [Batista 97a] Batista, G.E.A.P.A.; Monard, M.C. *Um Ambiente de Aquisição de Conhecimento e Avaliação de Algoritmos de Aprendizado Simbólico Utilizando Exemplos*. Anais 2º Workshop de Dissertações em Andamento, ICMSC, USP, Fevereiro, 1997, pp. 47-48.
- [Batista 97b] Batista, G.E.A.P.A.; Monard, M.C. *Descrição da Implementação Prolog de uma Ferramenta para Extração de Conhecimento de Redes Neurais*. Relatório Técnico ICMSC-SP, Nº 54, ISSN-0103-2569, Março, 1997.
- [Batista 97c] Batista, G.E.A.P.A.; Monard, M.C. *AMPSAM: Um Ambiente Computacional para Medir a Performance de Sistemas de Aprendizado de Máquina*. Anais do I Encontro Nacional de Inteligência Artificial, 1997.
- [Batista 97d] Batista, G.E.A.P.A.; Monard, M.C. *Projeto e Implementação de Conversores Sintáticos para Arquivos de Exemplos Proposicionais*. (Em andamento).
- [Batista 97e] Batista, G.E.A.P.A.; Monard, M.C. *Descrição da Implementação dos Métodos Estatísticos de Resampling do Ambiente AMPSAM*. (Em andamento).
- [Boswell 90] Boswell, T. *Manual for NewId version 4.1*. The Turing Institute, TI/P2154/RAB/4/2.3, January, 1990.
- [Boswell 90a] Boswell, T. *Manual for CN2 version 4.1*. The Turing Institute, TI/P2154/RAB/4/1.3, January, 1990.
- [Bratko 90] Bratko, I. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 1990.
- [Breiman 84] Breiman, L.; Friedman, J.; Olshen, R.; Stone C. *Classification and Regression Trees*. Wadsworth, CA, 1984.

- [Bruner 56] Bruner, J.S.; Goodnow, J.J.; Austin, G.A. *A Study of Thinking*. Wiley, 1956.
- [Cameron-Jones 93] Cameron-Jones, R.M.; Quinlan, J.R. *Efficient Top-Down Induction of Logic Programs*. Technical Report, Basser Department of Computer Science, University of Sydney, NSW 2006, Australia, 1993.
<ftp://ftp.cs.su.oz.au/pun/ml/cj+q.sigart.ps>
- [Casanova 87] Casanova, M.A.; Giorno, F.A.C.; Furtado, A.L. *Programação Lógica e a Linguagem Prolog*. Editora Edgard Blücher, 1987.
- [Castiñeira 91] Castiñeira, M.I.; Monard, M.C.; Nicoletti, M.C. *Aprendizado de Máquina: Descrição e Implementação de um Algoritmo Geral para a Construção de Árvores de Decisão*. Notas do ICMSC-USP, Nº 98, ISSN 0103-2577, 1991.
- [Cendrowska 87] Cendrowska, J. *PRISM: An Algorithm for Inducing Modular Rules*. International Journal of Man-Machine Studies, 27, 1987, pp. 349-370.
- [Clark 89] Clark, P.; Niblett, T. *The CN2 Induction Algorithm*. Machine Learning, 3(4), pp. 261-283, 1989.
<http://www.cs.utexas.edu/users/pclark/papers/cn2.ps>
- [Detrano 89] Detrano, R.; Janosi, A.; Steinbrunn, W.; Pfisterer, M.; Schmid, J.; Sandhu, S.; Guppy, K.; Lee, S.; Froelicher, V; *International Application of a New Probability Algorithm for the Diagnosis of Coronary Artery Disease*. American Journal of Cardiology, 64, 1989, pp. 304-310.
- [De Jong 88] De Jong, K. *Learning with Genetic Algorithms: An Overview*. Machine Learning, Nº 3, 1988, pp. 121-138.
- [De Raedt 92] Raedt L.; Bruynooghe M. *A Theory of Clausal Discovery*. Proceedings of the 13th International Joint Conference on Artificial Intelligence. Morgan Kaufmann, 1992.
- [Dietterich 97] Dietterich, T.G. *Statistical Tests for Comparing Supervised Classification Learning Algorithms*. In Press, 1997.
<ftp://ftp.cs.orst.edu/pub/tgd/papers/stats.ps.gz>
- [Dougherty 90] Dougherty, D. *Sed & Awk*. O'Reilly & Associates, 1990.
- [Feelders 95] Feelders, A.; Verkooijen, W. *Which Method Learns Most from the Data? Methodological Issues on the Analysis of Comparative Studies*. Preliminary Proceedings of the 5th International Workshop on AI and Statistics, 1995, pp. 219-225.
<http://diamond.ddi.nl/~ad/docs.aistats95.ps>

- [Feng 90] Feng, C.; Muggleton, S.H. *GOLEM 1.0 User's Commands*. 1990.
- [Flach 94] Flach, P. *Simply Logical. Intelligent Reasoning by Example*. Wiley, 1994.
- [Flexer 96] Flexer, A. *Statistical Evaluation of Neural Networks Experiments: Minimum Requirements and Current Practice*. Proceedings of the 13th European Meeting on Cybernetics and Systems Research, 2, 1996, pp. 1005-1008.
<ftp://ftp.ai.univie.ac.at/papers/oefai-tr-95-06.ps.Z>
- [Freitas 92] Freitas, A.A.; Kirner, C. *Introdução a Algoritmos Genéticos*. Anais do I Workshop sobre Redes Neurais, UFSCar, São Carlos, pp. 71-88, 1992.
- [Fürnkranz 94] Fürnkranz, J.; Widmer, G. *Incremental Reduced Error Pruning*. Cohen, W. and Hirsh, H. (Eds.), Proceedings of the 11th International Conference on Machine Learning, New Brunswick, NJ, Morgan Kaufmann, 1994, pp. 70-77.
- [Fürnkranz 96] Fürnkranz, J. *Separate-and-Conquer Rule Learning*. Technical Report, Austrian Research Institute for Artificial Intelligence, OEFAI-TR-96-25, 1996.
<ftp://ftp.ai.univie.ac.at/papers/oefai-tr-96-25.ps.Z>
- [Garner 95] Garner, S. *WEKA: The Waikato Environment for Knowledge Analysis*. Proceedings of the New Zealand Computer Science Research Students Conference, University of Waikato, New Zealand, 1995, pp. 57-64.
<http://www.cs.waikato.ac.nz/~ml/publications/1995/Garner-95-WEKA.ps.gz>
- [Haykin 94] Haykin, S. *Neural Networks A Comprehensive Foundation*. Macmillan College Publishing Company, Inc., 1994.
- [Helft 89] Helft, N. *Induction as a Nonmonotonic Inference*. Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, 1989, pp. 149-156.
- [Henery 94] Henery, R.J. *Classification*. Machine Learning, Neural and Statistical Classification, Michie, D; Spiegelhalter D.J.; Taylor C.C. (Eds.), Ellis Horwood, 1994, pp. 6-16.
- [Hildebrand 86] Hildebrand, D. *Statistical Thinking for Behavioral Scientists*. Duxbury Press, Boston, MA, 1986.

- [Hilton 95] Hilton, G.E.; Neal, R.M.; Tibshirani, R.; Revow, M.; Rasmussen, C.E.; Camp D.; Kustra, R.; Ghahramani, Z. *Assessing Learning Procedures Using DELVE*. The DELVE Development Group, (Draft), November, 1995.
- [Holland 86] Holland, J.H. *Escaping Brittleness: The Possibilities of General-purpose Learning Algorithms Applied to Parallel Rule-based Systems*. Michalski, R.S.; Carbonell, J.G.; Mitchell, T.M. (Eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 2, Morgan Kaufmann, 1986.
- [Holte 93] Holte, C.R. *Very Simple Classification Rules Perform Well on Most Commonly Used Datasets*. Machine Learning, 11, 1993, pp. 63-91.
- [Hopfield 82] Hopfield, J.J. *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*. Proceedings of the National Academy of Sciences of the U.S.A., 81, 1982, pp. 3088-3092.
- [Kearns 94] Kearns, M.J.; Vazirani, U.V. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.
- [Kibler 88] Kibler, D. Langley, P. *Machine Learning as a Experimental Science*. Machine Learning, 3(1), 1988, pp.5-8.
- [Kohavi 96] Kohavi, R.; John G.; Long, R.; Manley D.; Pfleger, K. *MLC++: A Machine Learning Library in C++*. Technical Report, Computer Science Department, Stanford University, August, 1994.
- [Kubat 97] Kubat, M.; Matwin, S. *Addressing the Curse of Imbalanced Training Sets: One-Sided Selection*. Draft, Department of Computer Science, University of Ottawa, 1997.
- [Lavrač 94] Lavrač, N.; Džeroski, S. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, London, 1994.
- [Lavrač 95] Lavrač, N.; Raedt, L. *Inductive Logic Programming: A Survey of European Research*. AI Communications 8(1), 1995, pp. 3-19.
- [Martins 94] Martins, C.A.M. *Uso de Árvores de Decisão na Geração de Hipóteses Indutivas em Aprendizado de Máquina*. Dissertação de Mestrado, ICMSC-USP, 1994.
- [McCulloch 43] McCulloch, W.S.; Pitts, W. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics, N^o 5, 1943, pp. 115-133.
- [Merz 96] Merz, C.J.; Murphy, P.M. *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California, Department of Information

and Computer Science.
<http://www.ics.uci.edu/~mlearn/MLRepository.html>

- [Michalski 69] Michalski, R.S. *On the Quasi-Minimal Solution of the Covering Problem*. Proceedings of the 5th International Symposium on Information Processing FCIP-69, Vol. A3, Bled, Yugoslavia, 1969, pp. 125-128.
- [Michalski 86] Michalski, R.S.; Mozetic, I.; Hong, J.; Lavrac, N. *The Multi-purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains*. Proceedings of the Fifth Annual National Conference on Artificial Intelligence, Philadelphia, 1986, pp. 1041-1045.
- [Michalski 87] Michalski R.S. *Concept Learning*. Encyclopedia of Artificial Intelligence, Shapiro, S.C.; Eckroth D.; Vallasi G.A. (Eds.), John Wiley & Sons, 1987, pp. 185-194.
- [Michie 94] Michie, D.; Spiegelhalter, D.J.; Taylor, C.C. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.
- [Milaré 97] Milaré, C.R.; Batista, G.E.A.P.A.; Monard, M.C. *Uma Ferramenta para Extração de Conhecimento de Redes Neurais*. Anais do XXIV Semish, 1997, pp 59-70.
- [Mingers 89] Mingers, J. *An Empirical Comparison of Pruning Methods for Decision Tree Induction*. Machine Learning, 4, 1989, pp. 227-243.
- [Minsky 69] Minsky, M.L.; Papert, S.A. *Perceptrons*. MIT Press, Cambridge, 1969.
- [Mitchell 80] Mitchell, T.M. *The Need for Biases in Learning Generalizations*. Technical Report, Computer Science Department, Rutgers University, 1980.
- [Monard 95] Monard, M.C.; Nicoletti, M.C. *Performance Comparison in Real World Applications of Two Symbolic Learning Systems for Knowledge Acquisition*. Anais II SBAI, Curitiba, PR, Setembro, 1995, pp. 231-236.
- [Monard 97a] Monard, M.C.; Batista, G.E.A.P.A.; Kawamoto, S.; Puglisi, J.B. *Uma Introdução ao Aprendizado Simbólico de Máquina*. Notas Didáticas ICMSC, Nº 29, ISSN 0103-2585, 1997.
- [Monard 97b] Monard, M.C.; Batista, G.E.A.P.A. *An Interactive Computational System for Measuring the Error of Classifiers that Discriminate Between Two Alternatives*. (Em andamento).
- [Morgan 73] Morgan, J.; Messenger, R. *THAID: A Sequential Search Program for the Analysis of Nominal Scale Dependent Variables*. Technical Report, Institute for Social Research, University of Michigan, 1973.

- [Muggleton 90] Muggleton, S.H.; Feng, C. *Efficient Induction of Logic Programs*. TIRM-90-044, Turing Institute Press, Glasgow, October, 1990.
- [Muggleton 94] Muggleton, S.; Raedt L. *Inductive Logic Programming: Theory and Methods*. The Journal of Logic Programming 19, 1994, pp. 629-679.
<ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP/Papers/lpj.ps>
- [Nicoletti 93] Nicoletti, M.C.; Monard, M.C. *Inverting Resolution: Plotkin's Least General Generalization*. Notas do ICMSC-USP, ISSN-0103-2577, Nº 130, Março, 1993.
- [Nicoletti 94] Nicoletti, M.C. *Ampliando os Limites do Aprendizado Indutivo de Máquina através das Abordagens Contrutiva e Relacional*. Dissertação de Doutorado, IFQSC-USP, 1994.
- [Pagallo 90] Pagallo, G.; Haussler, D. *Boolean Feature Discovery in Empirical Learning*. Machine Learning, 9, 1990, pp. 57-94.
- [Plotkin 70] Plotkin, G. D. *A Note on Inductive Generalization*. B. Meltzer and D. Michie (Eds.), Machine Intelligence 5, Edinburgh University Press, 1970, pp. 153-163.
- [Plotkin 71] Plotkin, G. D. *Automatic Methods of Inductive Inference*. Ph.D. Thesis, Edinburgh University, 1971.
- [Prechelt 94] Prechelt, L. *Proben1 — A Set of Neural Network Benchmark Problems and Benchmark Rules*. Technical Report, Fakultät für Informatik, Universität Karlsruhe, 1994.
<ftp://ftp.ira.uka.de/pub/papers/techreports/1994/1994-21.ps.gz>
- [Quinlan 86] Quinlan, J.R. *Induction of Decision Trees*. Machine Learning 1, 1986, pp. 81-106.
- [Quinlan 87a] Quinlan, J.R. *Generating Production Rules from Decision Trees*. Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Italy, 1987, pp. 304-307.
- [Quinlan 87b] Quinlan, J.R. *Simplifying Decision Trees*. International Journal of Man-Machine Studies, Nº 27, 1987, pp. 221-234.
- [Quinlan 88] Quinlan, J.R. *C4.5 Programs for Machine Learning*. Morgan Kaufmann Publishers, CA, 1988.
- [Quinlan 90] Quinlan, J.R. *Learning Logical Definition from Relations*. Machine Learning 5, 1990, pp. 239-266.
- [Quinlan 93] Quinlan, J.R.; Cameron-Jones, R.M. *FOIL: A MidTerm Report*. Brazdil, P.B. (Ed.), Proceedings of the 6th European Conference on Machine Learning ECML-93, Springer Verlag, Viena, Austria, 1993, pp. 3-20.

- [Rivest 87] Rivest, R.L. *Learning Decision Lists*. Machine Learning, 2, 1987, pp. 229-246.
- [Rosenblatt 58] Rosenblatt, F. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*. Psychological Review, № 65, 1958, pp. 386-408.
- [Rumelhart 86] Rumelhart, D.E.; McClelland, J.L. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1, MIT Press, Cambridge, 1986.
- [Russell 95] Russell, S.J.; Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall International, 1995.
- [Salzberg 97] Salzberg, S.L. *On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach*. Data Mining and Knowledge Discovery, 1, 1997, pp.1-11.
<http://www.cs.jhu.edu/~salzberg/critique.ps>
- [Sammut 93] Sammut, C. *The Origins of Inductive Logic Programming. A Prehistoric Tale*. S. Muggleton (Ed.), Third International Workshop on Inductive Logic Programming, Bled, Solvenia, 1993.
http://www.cse.unsw.edu.au/~clauder/research/papers/ilp_history.ps.gz
- [Sammut 94] Sammut, C. *Knowledge Representation*. Machine Learning, Neural and Statistical Classification, Michie, D; Spiegelhalter D.J.; Taylor C.C. (Eds.), Ellis Horwood, 1994, pp. 228-245.
- [Schaffer 93] Schaffer, C. *A Conservation Law for Generalization Performance*. Machine Learning, 10, 1993, pp. 153-178.
<http://louis.hunter.cuny.edu/faculty/schaffer/papers/conservation.ps>
- [Smith 88] Smith, J.W.; Everhart, J.E.; Dickson, W.C.; Knowler, W.C.; Johannes, R.S. *Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus*. Proceedings of the Symposium on Computer Applications and Medical Care, IEEE Computer Society Press, 1988, pp. 261-265.
- [Sprent 89] Sprent, P. *Applied Nonparametric Statistical Methods*. Chapman and Hall, 1989.
- [Stanfill 86] Stanfill, C.; Waltz, D. *Toward Memory-based Reasoning*. Communications of the ACM 29, 12, 1986, pp. 1213-1228.
- [Venturini 93] Venturini, G. *SIA: A Supervised Inductive Algorithm with Genetic Search for Learning Attributes Based Concepts*. Brazdil, P.B. (Ed.), Proceedings of the 6th European Conference on Machine Learning ECML-93, Springer Verlag, Viena, Austria, 1993, pp. 3-20.

- [Wall 96] Wall, L.; Christiansen, T.; Schwartz R. *Programming Perl*. O'Reilly & Associates, 1996.
- [Weiss 91] Weiss, S.M.; Kulikowski, C.A. *Computer Systems That Learn. Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann Publishers, CA, 1991.
- [Weiss 95] Weiss, S.M.; Indurkha, N. *Rule-Based Machine Learning Methods for Functional Prediction*. Journal of Artificial Intelligence Research, 3, 1995, pp. 383-403.
<http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume3/weiss95a.ps.Z>
- [Weiss 96] Weiss, S.M.; Indurkha, N. *Selecting the Right-Size Model for Prediction*. Applied Intelligence, 6, 1996, pp. 261-273.