FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO


FEUP

# Métodos Estatísticos Relacionais para Previsão de Resultados Médicos

**Tiago Amorim Ferreira Couteiro**

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Vitor Manuel de Morais Santos Costa (Professor Associado)

22 de Julho de 2010

# Métodos Estatísticos Relacionais para Previsão de Resultados Médicos

**Tiago Amorim Ferreira Couteiro**

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: Maria Eduarda Silva Mendes Rodrigues (Professor Auxiliar Convidado)
Vogal Externo: José Nuno Panelas Nunes Lau (Professor Auxiliar)
Orientador: Vitor Manuel de Morais Santos Costa (Professor Associado)

22 de Julho de 2010

# Resumo

A Programação Lógica Indutiva (ILP) é um método de aprendizagem de regras frequentemente usadas para tarefas de classificação. A utilização conjunta de Redes Bayesianas e ILP na aprendizagem aumenta a precisão em comparação com a abordagem padrão de listas de decisão.

Tradicionalmente a combinação das regras individuais para obtenção de um classificador útil resulta de um processo em duas etapas. As regras são geradas na primeira fase e o classificador é aprendido na segunda fase.

O algoritmo SAYU intercala as duas etapas, de forma incremental, construindo uma rede de Bayes durante a aprendizagem das regras. Cada regra candidata só é introduzida no classificador se aumentar o seu desempenho.

SAYU baseia-se em Métodos Estatísticos Relacionais e apresenta excelentes resultados em algumas aplicações médicas. No entanto é limitado por dois problemas: aceitar muitas regras pode levar a erros de classificação e aceitar poucas regras pode levar a um classificador limitado.

Neste trabalho considerou-se um algoritmo que integra um classificador usado em SAYU com outro classificador. Uma dada regra só será integrada se melhorar o desempenho dos dois classificadores.

Foram utilizados três tipos de dados e constatada melhoria em alguns *folds* testados. Este algoritmo necessita, no entanto, de ser testado noutro tipo de dados e com outro tipo de classificadores.

# Abstract

Inductive Logic Programming (ILP) is an approach for learning rules for classification tasks. The use of Bayes Net learner with ILP improves the accuracy compared to the standard decision list approach.

Usually combining the single rules to obtain a useful classifier result a two-step process, where rules are generated in the first phase, and the classifier is learned in the second phase.

The algorithm SAYU interleaves the two steps, by incrementally building a Bayes net during rule learning. Each candidate rule is introduced into the network if only it improves the performance of the classifier.

SAYU, a Statistic Relacional Method, presents excellent results in some medical applications, but it is limited by two problems: to accept many rules can lead to mistakes of classification and accept few rules can lead to a limited classifier.

In this project was considered a algorithm that integrate a classifier used in SAYU with another classifier. Each candidate rule is introduced into the classifiers if only it improves the performance of the both classifiers.

Three different datasets were used and seen improvement in some of the folds. Furthermore, this algorithm needs future studies in another datasets and with nother classifiers.

# Agradecimentos

Ao Prof. Doutor Vitor Manuel de Morais Santos Costa, orientador/coordenador da dissertação, um sincero agradecimento pelo apoio constante que proporcionou ao longo da elaboração de toda a dissertação, orientando o estudo dos conceitos e técnicas mais complexas necessários à compreensão dos procedimentos aplicados neste trabalho.

Pela liberdade de pensamento que me proporcionou desenvolvendo, em simultâneo, o sentido da responsabilidade ficarei eternamente grato.

Tiago Couteiro

# Conteúdo

# Lista de Figuras

LISTA DE FIGURAS

# Lista de Tabelas

# Abreviaturas e Símbolos

| | |
|---|---|
| AA | Aprendizagem Automática |
| ALEPH | *A Learning Engine for Proposing Hypotheses* |
| AUC | *Area Under the Curve* |
| ILP | Programação Lógica Indutiva |
| kFOIL | *First-Order Logic* com Kernel |
| MDIE | *Mode Direct Inverse Entailment* |
| NB | *Naive Bayes* |
| nFOIL | *First-Order Logic* com NB |
| OMOP | *Observational Medical Outcomes Partnership* |
| RDM | *Relational Data Mining* |
| SAYU | *Score As You Use* |
| SRL | Aprendizagem Estatística Relacional |
| TAN | *Tree-Augmented Naive Bayes* |
| RB | *Redes Bayesianas* |
| ML | *Maximum Likelihood* |
| EM | *Expectation Maximization* |
| MAP | *Maximum à Posteriori* |
| SP | *Super Parent* |
| SVM | *Support Vector Machines* |
| SV | *Support Vectors* |
| RP | Partição Recursiva |
| ROC | *Receiver Operator Characteristic* |
| PR | *Precision-Recall* |
| FRP | Taxa de Falsos Positivos |
| TPR | Taxa de Verdadeiros Positivos |
| AUC-ROC | *Area Under the Curve-Receiver Operator Characteristic* |
| AUC-PR | *Area Under the Curve-Precision-Recall* |
| WEKA | *Waikato Environment for Knowledge Analysis* |
| YAP | *Yet Another Prolog* |

# Capítulo 1

# Introdução

A investigação em Aprendizagem Automática (AA) tem, segundo Mitchell [Mit97], como objectivo a construção de programas cujo desempenho, em determinada tarefa, melhora com a experiência. Nos últimos anos ocorreram importantes avanços nos algoritmos que se baseiam neste princípio, em particular nos algoritmos para Aprendizagem Estatística Relacional (SRL) que constituem o foro deste trabalho.

## 1.1 Contexto/Enquadramento

A Aprendizagem Estatística Relacional (SRL) estuda modelos de domínios que apresentam incerteza, isto é, que podem ser tratados através de métodos estatísticos e apresentam estruturas relacionais complexas. Este tipo de aprendizagem tem dado bons resultados no estudo de "Casos Médicos" [DBdCD+05b] e por isso foi alvo de estudo neste trabalho.

A representação do conhecimento desenvolvido em SRL recorre muitas vezes a modelos probabilísticos gráficos como Redes Bayesianas (RB) [RN95] . Por isso foram abordados os classificadores Bayesianos.

O *Naive Bayes* (NB) é um exemplo de classificador Bayesiano incremental simples [Alc02] pois assume que dado o valor da classe todos os atributos são independentes entre si. Nos problemas reais, a independência dos atributos raramente se verifica. Uma alternativa é o classificador *Tree-Augmented Naive Bayes* (TAN) [FG96] que relaxa a independência.

A Programação Lógica Indutiva (ILP) aprende regras que podem ser aplicadas directamente a dados multi-relacionais para encontrar padrões que envolvam múltiplas relações. O sistema *A Learning Engine for Proposing Hypotheses* (Aleph) [Sri93] foi desenvolvido com o intuito de ser um protótipo para explorar ideias de ILP.

A questão fundamental da Aprendizagem Estatística Relacional consiste, no entanto, em combinar as regras individuais para obter um classificador útil. Assim surge *Score As You Use* (SAYU) e abordagens alternativas como *First-Order Logic* com NB (nFOIL) e *First-Order Logic* com Kernel (kFOIL).

## 1.2 Motivação e Objectivos

Depois de analisar os resultados obtidos com SAYU [DBdCD+05a], [DBdCD+05b] em alguns casos reais considera-se que a sua eficiência ainda poderá ser melhorada.

O objectivo deste trabalho consistiu em construir um algoritmo integrando o classificador usado em SAYU com outro classificador de AA por forma a encontrar melhores resultados.

A ideia é de que uma dada regra só seja adicionada ao classificador se for aceite pelos dois classificadores, o do SAYU e o que for escolhido entre os existentes em AA.

## 1.3 Estrutura da Dissertação

A tese encontra-se organizada do seguinte modo:

No capítulo 2 efectua-se a Revisão Bibliográfica sobre AA e Aleph, sistema de Programação Lógica Indutiva.

No capítulo 3 são apresentados os conceitos fundamentais associados ao algoritmo SAYU e referidos algoritmos alternativos nFOIL e kFOIL.

No capítulo 4 é feita a descrição do procedimento proposto neste trabalho.

No capítulo 5 apresentam-se os resultados obtidos e a sua interpretação.

No capítulo 6 apresentam-se as conclusões da dissertação e alguns aspectos que poderão ser importantes para futuro melhoramento do classificador.

# Capítulo 2

# Revisão Bibliográfica

Neste capítulo são abordados conceitos de Aprendizagem Automática Supervisionada e não Supervisionada e de Programação Lógica Indutiva, em particular o sistema Aleph, necessários para o entendimento dos procedimentos aplicados neste trabalho.

## 2.1 Aprendizagem Automática (Machine Learning)

A Aprendizagem Automática é uma área muito vasta da Inteligência Artificial. Segundo Mitchell [Mit97], a investigação em AA tem como objectivo a construção de programas que possam "aprender" com a experiência, ou seja, cujo desempenho em determinada tarefa melhora com a experiência. Mitchell afirma mesmo que um entendimento detalhado dos algoritmos de AA pode também levar a um melhor entendimento da capacidade (e incapacidade) do processo humano de aprendizagem. A Aprendizagem Automática é multidisciplinar e trabalha com ideias de diversas áreas, incluindo Inteligência Artificial, Probabilidade e Estatística, Teoria da Informação, Psicologia, Neuro-ciências e Filosofia, entre outras. De acordo com Mitchell, os algoritmos de AA têm grande valor prático em muitos domínios :

- Problemas de *Data Mining*, onde se analisam automaticamente grandes bancos de dados, com o intuito de encontrar regularidades implícitas;
- Em domínios ainda mal estudados, isto é, onde ainda não se tem conhecimento necessário para desenvolver algoritmos efectivos (por exemplo, no reconhecimento facial a partir de imagens);
- Em domínios onde o programa necessita de se adaptar dinamicamente a mudanças (por exemplo, um sistema que se adapta às preferências de leitura de um indivíduo);

- Em domínios em que o custo da aquisição ou codificação manual do conhecimento é muito elevado, como é o caso do Processamento de Linguagem Natural.

### 2.1.1 Tipos de Aprendizagem

Um Sistema de Aprendizagem pode ser formalizado como um método de aprender uma função G onde os parâmetros são inicialmente desconhecidos, como está descrito na Figura 2.1.

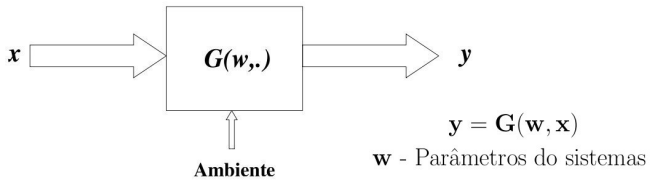



Figura 2.1: Objectivo da Aprendizagem

A Aprendizagem resulta da adaptação do vector de parâmetros *w* de forma a que o Sistema, por interacção com o Ambiente, realize uma transformação apropriada.

Existem várias técnicas de AA originando assim diferentes tipos de aprendizagem. Serão abordados apenas três tipos de aprendizagem:

- Aprendizagem Supervisionada
- Agrupamento
- Aprendizagem por Reforço

#### 2.1.1.1 Aprendizagem Supervisionada

A Aprendizagem Supervisionada é a versão mais tradicional de AA e tem como objectivo deduzir uma função a partir dos dados de treino *d*. Os dados de treino são formados por pares conhecidos de objectos *x* (usualmente vectores) e os respectivos resultados *y*.

Este tipo de aprendizagem está esquematizado na Figura 2.2.

Neste caso o Sistema é adaptado em função de um conjunto de dados de entrada para os quais a saída é conhecida.

A função de saída pode ser um valor contínuo e nesse caso o problema de Aprendizagem Supervisionada é conhecido como *Regressão* ou pode ser prever um rótulo de classe para a saída, e nesse caso o problema é conhecido como *Classificação*.

A tarefa da Aprendizagem Supervisionada consiste em prever o valor da função de saída para qualquer conjunto de dados válido depois de ter sido analisada uma série de

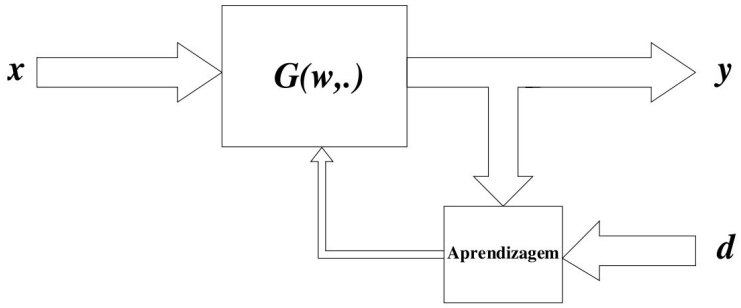



Figura 2.2: Aprendizagem Supervisionada

dados de treino (ou seja, pares de entrada e valores de saída). Para alcançar este objectivo torna-se necessário generalizar, com base nos dados apresentados, para situações desconhecidas.

#### 2.1.1.2 Agrupamento

Neste tipo de Aprendizagem o objectivo é agrupar um conjunto de elementos dado de acordo com características comuns; daí a designação de Agrupamento (*Clustering*).

Dado um conjunto de dados $X = X_1, X_2, ..., X_n$ chama-se agrupamento de X a uma partição de X em $m$ conjuntos (*clusters* ou grupos) $C_1, C_2, ..., C_m$ que verificam as seguintes condições:

1) $C_i \neq \emptyset \; ; i = 1, 2, ..., m$ (nenhum *cluster* pode ser vazio);

2) $\bigcup_{i=1}^{m} C_i = X$ (A união de todos os *clusters* deve ser igual ao conjunto de dados que os gerou);

3) $C_i \bigcap C_j = \emptyset \; ; i \neq j; i, j = 1, 2, ..., m$ (Dois *clusters* não podem conter vectores comuns existindo casos em que esta condição é relaxada).

As características dos elementos contidos num determinado cluster, $C_i$, devem ser semelhantes. O procedimento de aprendizagem que tenta identificar características específicas dos agrupamentos existentes num conjunto de dados constitui o algoritmo de "clustering" [TK08].

Estes algoritmos são classificados como hierárquicos ou de partição. Os algoritmos *hierárquicos aglomerativos* produzem uma sequência de agrupamentos com um número decrescente de *clusters* uma vez que os agrupamentos produzidos em cada passo resultam da fusão de dois *clusters* do passo anterior. Os *algoritmos de partição* têm por objectivo construir uma partição do conjunto dos $n$ dados em $m$ grupos distintos recorrendo a critérios que dividam os grupos.

O que torna o Agrupamento diferente da Aprendizagem Supervisionada é que não são conhecidos, à partida, os grupos a que os elementos devem pertencer. O sistema é que

deve encontrar um agrupamento natural. Do esquema apresentado na Figura 2.3 pode constatar-se que, neste tipo de aprendizagem, não são fornecidos ao agente de Aprendizagem os dados de treino.

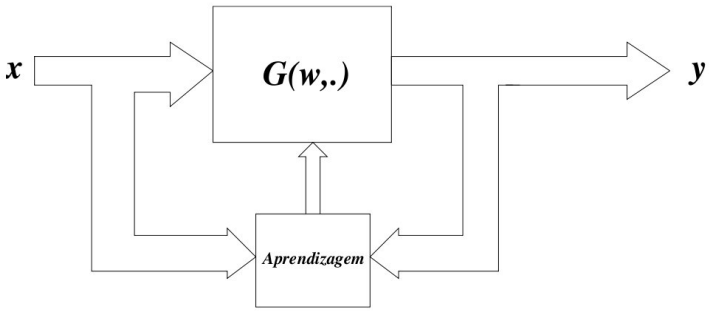



Figura 2.3: Agrupamento

#### 2.1.1.3 Aprendizagem por Reforço

Este tipo de Aprendizagem pode ser descrito como "um agente que interage com o mundo, faz observações, age, e como resultado ou é recompensado ou é punido". O sistema deverá por isso ser capaz de escolher acções de maneira a maximizar o número de recompensas ou a recompensa total.

No modelo da Aprendizagem por Reforço, o agente de Aprendizagem interage com o ambiente [Har96]. Esta interacção consiste na percepção do ambiente e na selecção de uma acção para executar nesse ambiente. A acção altera o ambiente de alguma forma e esta alteração é comunicada ao agente através de um sinal de reforço (ou recompensa). Esta recompensa pode ser positiva ou negativa, e reflecte a qualidade da acção para um estado.

O problema central da Aprendizagem por Reforço reside, portanto, na escolha de acções. Quase todos os algoritmos que implementam este tipo de aprendizagem estimam funções de valor [Lin93], funções que determinam a qualidade do estado, função estado-valor, a qualidade da execução de uma acção num estado - função acção-valor.

A Figura 2.4 esquematiza o processo da Aprendizagem por Reforço.

De notar que este método de Aprendizagem é diferente da Aprendizagem Supervisionada porque não é fornecida explicitamente a acção a aprender. Na Aprendizagem por Reforço o agente aprende, por tentativa e erro, até atingir um objectivo interagindo com o seu ambiente. O objectivo do agente é escolher as acções que maximizam os reforços subsequentes.

Por outro lado, a Aprendizagem por Reforço usa muitas vezes técnicas de Aprendizagem Supervisionada incremental para poder "aprender".

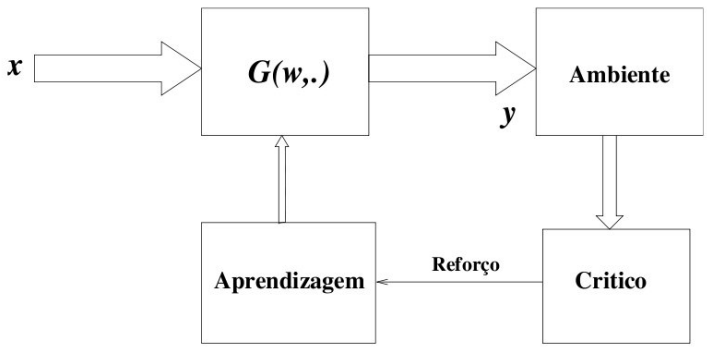



Figura 2.4: Aprendizagem por Reforço

### 2.1.2 Métodos de Aprendizagem Supervisionada

A Aprendizagem Supervisionada é o foro deste trabalho. Assim sendo apresentam-se, de seguida, alguns algoritmos que foram particularmente relevantes para a realização do presente trabalho.

#### 2.1.2.1 Redes Bayesianas

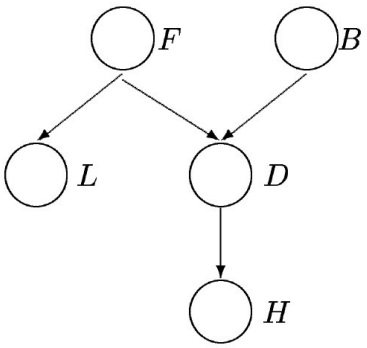
> " Given the number of times in which an unknown event has happened and failed: Required the chance that the probability of it happening in a single trial lies somewhere between any two degrees of probability that can be named." [Bay73]

Uma Rede Bayesiana [RN95] é um grafo acíclico dirigido que representa dependências entre variáveis num modelo probabilístico. Esta abordagem representa uma boa estratégia para lidar com problemas que tratam de incerteza, onde as conclusões não podem ser construídas apenas a partir do conhecimento prévio do problema. Um exemplo simples de RB é apresentado na Figura 2.5.

Cada nó $X_i$ de uma RB é uma variável aleatória e portanto tem associada uma distribuição de probabilidade $P(X_i|Pais(X_i))$, onde $Pais(X_i)$ são todos os nós que possuem arcos que vão em direção ao nó $X_i$. A distribuição conjunta de todas as variáveis $(X_1, X_2, ..., X_n)$ de uma RB é definida por

$$P(X_1, X_2, ..., X_n) = \Pi_{i=1}^{n} P(X_i|X_j;\ X_j \in Pais(X_i)) \tag{2.1}$$

para cada enupla de valores $(x_1, x_2, ..., x_n)$.




Probabilidades: $p(F), p(B), p(L|F), p(D|F,B), p(H|D)$

Figura 2.5: Rede Bayesiana

Normalmente as distribuições $P(X_i|Pais(X_i))$ de cada variável $X_i$ de uma RB devem ser aprendidas a partir dos dados disponíveis. No caso em que se conhece a estrutura da rede e todas as variáveis são observáveis, a aprendizagem é feita pela estimação por maximização da verosimilhança, *Maximum Likelihood (ML) Estimation*. Se a estrutura é conhecida mas existem variáveis não observadas emprega-se o algoritmo de gradiente ascendente [RBKK95] ou o algoritmo *Expectation Maximization* (EM) [MK97] para a aprendizagem (ambos utilizam inferência probabilística).

Utiliza-se inferência probabilística para determinar a distribuição de probabilidade de um dado conjunto de variáveis de consulta Y a partir de alguma evidência observada, E, onde evidência é definida como um conjunto de variáveis para as quais o valor observado é $e$. Seja X o conjunto de todas as variáveis aleatórias de uma RB e Z = (X - E - Y) as variáveis não observadas. A inferência probabilística pode ser feita através do uso da distribuição conjunta de X da seguinte forma:

$$P(Y|e) = \frac{P(Y,e)}{P(e)} = \frac{\sum_z P(Y,e,z)}{\sum_y \sum_z P(y,e,z)} \tag{2.2}$$

onde P(Y,e,z) e P(y,e,z) são provenientes da distribuição conjunta de X = (Y + E + Z), e o somatório é feito sobre todas as possíveis combinações de valores z e y para as variáveis Z e Y, respectivamente.

Na maioria dos casos é necessária a utilização de algoritmos aproximados para a inferência probabilística: *Likelihood Weighting* e *Monte Carlo Markov Chain* usam geração de amostras aleatórias a partir das distribuições de probabilidade $P(X_i|Pais(X_i))$ de cada variável aleatória $X_i$ [RN95].

#### 2.1.2.2 Naive Bayes

O classificador *Naive Bayes* [DP97] é um exemplo simples de uma RB que é aplicado a problemas de classificação.

A Figura 2.6 mostra um exemplo simples de uma RB. A variável *Classe* representa as possíveis classes do problema enquanto que as variáveis (discretas ou contínuas) *Atributo-k* (k=1,...,m) representam os atributos que caracterizam as classes. Devem ser consideradas as probabilidades P(*Classe*) e P($Atributo-k|Classe$) para k = 1, ...,m.

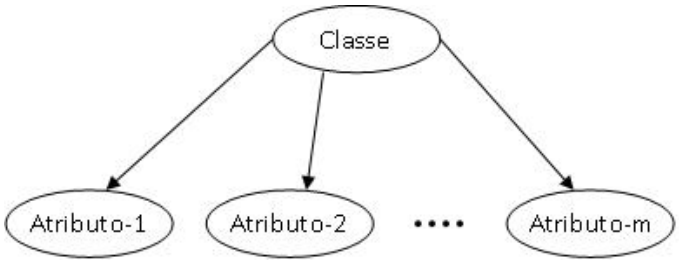



Figura 2.6: Classificador NB

O NB deve escolher uma classe C de um conjunto finito de valores discretos dom(C) dado um conjunto de atributos $X = [A_1, A_2, ..., A_m]$. Considerando apenas o caso mais simples em que todos os atributos são discretos, o classificador Bayesiano descobre a classe "Maximum à Posteriori" (MAP).

$$C_{MAP} = argmax_{c \in dom(C)} P(C|X) \tag{2.3}$$

Pelo teorema de Bayes temos que

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)} \tag{2.4}$$

Assim a fórmula da classe MAP pode ser reescrita como

$$C_{MAP} = argmax_{c \in dom(C)} P(X|C)P(C) \tag{2.5}$$

Como o NB considera que os atributos são condicionalmente independentes dada a classe temos que

$$P(X|C) = \prod_{j-1}^{m} P(X_j|C) \tag{2.6}$$

Substituindo essa expressão na fórmula da classe MAP obtemos a fórmula da hipótese

NB (Naive Bayes)

$$C_{NB} = argmax_{c \in dom(C)} \prod_{j-1}^{m} P(X_j|C) \tag{2.7}$$

Para usar essa expressão o NB deve estimar as probabilidades que aparecem pela contagem dos valores discretos presentes num conjunto de exemplos de treino.
Para cada $c \in dom(C)$ e $a_j \in dom(A_j)(j = 1,2,...,m)$ calcula-se:

$$P(c) = \frac{N(c)}{N} \tag{2.8}$$

$$P(a_j|c) = \frac{P(a_j,c)}{P(c)} = \frac{N(a_j,c)}{N(c)} \tag{2.9}$$

onde dom(C) e dom($A_j$) são conjuntos finitos de valores discretos, N é o número total de exemplos de treino, N(C) é o número de exemplos de treino com a classe "C", e $N(A_j,C)$ é o número de exemplos de treino com o atributo "$A_j$"e a classe "C". Para evitar uma contagem nula podemos adicionar M exemplos virtuais ao conjunto de treino:

$$P(c) = \frac{N(c) + M.Q}{N + M} \tag{2.10}$$

$$P(a_j|c) = \frac{N(a_j,c) + M.Q.Q_j}{N(c) + M.Q)} \tag{2.11}$$

onde $Q = \frac{1}{K}$ se C tem K valores possíveis, e $Q_j = \frac{1}{K_j}$ se $A_j$ tem $K_j$ valores possíveis.

#### 2.1.2.3 Tree-Augmented Naive Bayes

Apesar dos classificadores NB apresentarem normalmente bons resultados em tarefas de classificação e precisarem de um pequeno número de parâmetros estimados, apresentam a desvantagem de se basearem no pressuposto de que os atributos são independentes dada a classe, o que raramente se verifica no mundo real. Desta forma, é usual propor-se a utilização de um outro tipo de classificador, o classificador *Tree-Augmented Naive Bayes*, que possui a vantagem de modelar dependências entre os atributos em forma de uma estrutura em árvore.

A vantagem deste método consiste no facto destas dependências poderem ser aprendidas de forma eficiente a partir de dados de treino e a árvore resultante assegurar que a função de verosimilhança é maximizada.

Na estrutura do classificador TAN, a variável da classe é Pai de todos os atributos e cada atributo possui, no máximo, um outro atributo, de forma que o grafo resultante forma uma árvore.

As estruturas de classificadores NB e TAN podem ser comparadas nas figura 2.7 e figura 2.8.

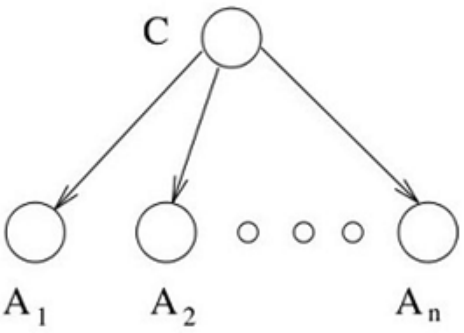



Figura 2.7: Classificador NB

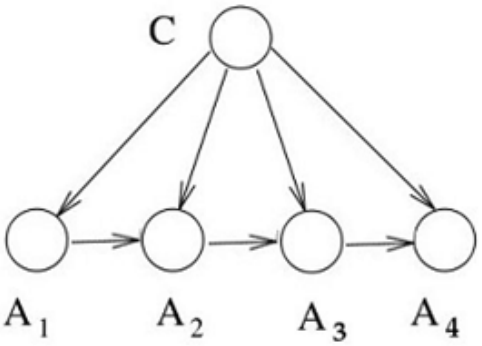



Figura 2.8: Classificador TAN

A construção da árvore de dependências é o que diferencia o TAN do NB. Esta vai determinar o desempenho do TAN. Teoricamente, é devido à dependência entre os atributos que o TAN melhora o desempenho em relação ao NB.

Friedman, ao analisar a construção da árvore de dependências para o algoritmo TAN verificou que, neste caso específico, sabe-se à partida que todos os atributos são dependentes da classe. Assim, torna-se necessário conhecer a quantidade de informação que o atributo X fornece sobre o atributo Y dada a classe, utilizando a seguinte fórmula:

$$I_p(X;Y|C) = \sum_{x,y,c} P(x,y,c)\frac{P(x,y|c)}{P(x|c)P(y|c)} \tag{2.12}$$

onde $I_p$ corresponde à informação mútua entre os atributos X e Y dado C.

Assumindo-se que ambos os atributos são dependentes da classe, calcula-se, deste modo, a informação que X fornece sobre Y dado o valor da classe.

Em seguida apresentam-se as fases da construção da árvore que maximiza a informação mútua:

1. Obter a informação mútua, $I_p(X,Y|C)$ entre cada par de atributos e construir um vector com essa informação mútua, Figura 2.9, passo (a).

2. Desenhar um grafo completo não orientado, tendo como nós os atributos e como custo das ligações a informação mútua entre os atributos, Figura 2.9, passo (b).

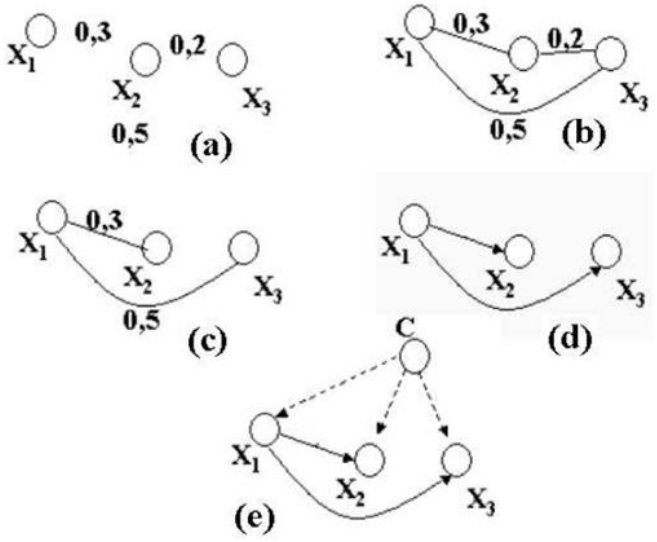



Figura 2.9: Fases da construção da árvore que maximiza a informação mútua.

3. Encontrar uma árvore que maximize a informação mútua, sem criar ciclos, Figura 2.9, passo (c).

4. Transformar a árvore não orientada em orientada, escolhendo como raiz o nó com informação mútua mais elevada, Figura 2.9, passo (d).

5. Adicionar a classe como "Pai" de todos os atributos, Figura 2.9 passo (e).

O algoritmo TAN de Friedman [FG96] foi motivo de estudo por vários autores. Algumas das propostas imediatamente apresentadas foram:

- **Smoothed TAN** apresentado por Friedman no mesmo artigo em que apresentou o TAN e visa combater o problema de existirem poucos exemplos em algumas partições.

  Assim, no smoothed TAN, são estimados em todas as partições

$$\theta^s(x|\prod_x) = \alpha\hat{P}_D(x|\prod_x) + (1-\alpha))\hat{P}_D(x) \tag{2.13}$$

  sendo $\alpha = \frac{N\hat{P}_D(x|\prod_x)}{N\hat{P}_D(x|\prod_x)+s}$ e s o parâmetro de suavização que, segundo Friedman, devia valer 5.

- **Super Parent** (SP), apresentado por Keogh e Pazzani [KP02], que propõem um algoritmo que vai adicionando 'arcos' à estrutura do Naive Bayes. Em cada passo

o algoritmo adiciona mais um 'arco' mantendo a condição de que cada atributo não tenha mais de um pai. O algoritmo designa como órfãos os que não têm outro atributo como pai. No primeiro passo o SP inicializa a rede com o Naive Bayes, em que todos os atributos são órfãos; assim a lista de órfãos é inicializada com todos os atributos.
Em seguida o algoritmo avalia para cada atributo o desempenho como "pai" para cada um da lista de órfãos, entendendo-se por desempenho a taxa de erro nos exemplos de teste. O que obtiver melhor desempenho é escolhido como "super pai". De seguida escolhe-se o "filho favorito", que corresponde ao que apresentou melhor desempenho como "filho do super pai" escolhido. O "filho favorito" é retirado da lista de órfãos e volta-se a escolher o novo "super pai". Este ciclo repete-se até que a lista de órfãos contenha apenas um elemento ou até que se verifique que o desempenho não melhora.
Este algoritmo trouxe como principal contribuição a ideia de não impor ao TAN a criação de uma árvore com $(n-1)$ ligações mas que pare de adicionar ligações quando atinge um critério.

#### 2.1.2.4 Support Vector Machines

*Support Vector Machines* (SVM) realiza a classificação através da construção de um hiperplano N-dimensional que separa os dados em duas categorias de forma optimizada.

Apresenta-se, de seguida, apenas os conceitos básicos a respeito de SVM para problemas de classificação [LC03].

SVM linear com margens rígidas define fronteiras lineares a partir de dados linearmente separáveis. Seja D um conjunto de treino com n dados $x_i \in X$ e $y_i \in Y$ os respectivos rótulos. X constitui o espaço dos dados e Y = [-1, 1]. Diz-se que D é linearmente separável se é possível separar os dados das classes 1 e -1 por um hiperplano [SS02].

Classificadores que separam os dados por meio de um hiperplano são designados lineares. A equação de um hiperplano é apresentada na Equação 2.14, em que $w \cdot x$ é o produto escalar entre os vectores $w$ e $x$, $w \in X$ é o vector normal ao hiperplano descrito e $\frac{b}{||w||}$ representa a distância entre o hiperplano e a origem, com $b \in \Re$.

$$f(x) = w \cdot x + b = 0 \tag{2.14}$$

Essa equação divide o espaço dos dados X em duas regiões: $w \cdot x + b > 0$ e $w \cdot x + b < 0$. A função sinal, Equação 2.15 pode ser utilizada na obtenção da classificação.

$$g(x) = sgn(f(x)) = \begin{cases} 1 & \text{se } w \cdot x + b > 0 \\ -1 & \text{se } w \cdot x + b < 0 \end{cases} \tag{2.15}$$

Chama-se hiperplano canónico do conjunto D ao hiperplano em que w e b são determinados de forma que os exemplos mais próximos do hiperplano $w \cdot x + b = 0$ satisfaçam a Equação 2.16.

$$|w \cdot x_i + b| = 1 \tag{2.16}$$

Essa forma implica a verificação da Inequação 2.17.

$$y_i(w \cdot x_i + b) - 1 \geq 0, \ \ \forall (x_i, y_i) \in D \tag{2.17}$$

Considere-se na Figura 2.10 os hiperplanos $H_1$ e $H_2$.

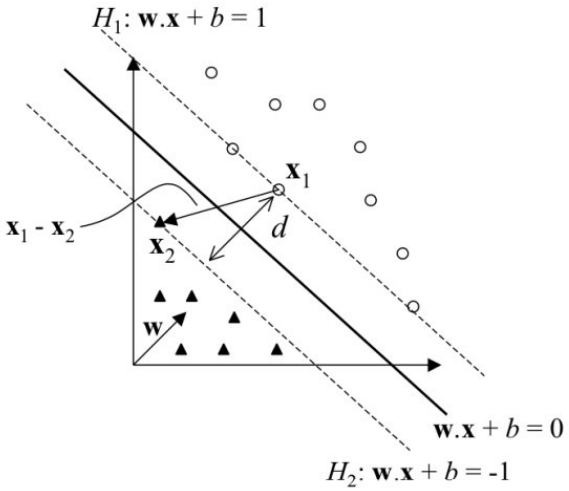



Figura 2.10: Cálculo da distância entre os hiperplanos

Seja $x_1$ um ponto no hiperplano $H_1 : w \cdot x + b = 1$ e $x_2$ um ponto no hiperplano $H_2 : w \cdot x + b = -1$. Projectando $x_1 - x_2$ na direcção de w, perpendicularmente ao hiperplano separador $w \cdot x + b = 0$, é possível obter a distância entre os hiperplanos $H_1$ e $H_2$. Essa projecção é dada pela Equação 2.18.

$$(x_1 - x_2)(\frac{w}{||w||}.\frac{(x_1 - x_2)}{||x_1 - x_2||}) \tag{2.18}$$

Como $w \cdot x_1 + b = 1$ e $w \cdot x_2 + b = -1$ a diferença entre essas equações permite calcular $w \cdot (x_1 - x_2) = 2$. Substituindo este resultado na equação projecção obtém-se a equação distância

$$\frac{2(x_1 - x_2)}{||w||||x_1 - x_2||} \tag{2.19}$$

cuja norma vale $\frac{2}{||w||}$ e representa a distância, d, entre os hiperplanos $H_1$ e $H_2$, paralelos ao hiperplano separador. Como w e b foram determinados de forma a não haver exemplos entre $H_1$ e $H_2$, $\frac{1}{||w||}$ é a distância mínima entre o hiperplano separador e os dados de treino. Essa distância é definida como a margem geométrica do classificador linear. Com base nestes resultados conclui-se que a maximização da margem de separação dos dados em relação a $w \cdot x + b = 0$ pode ser obtida pela minimização de ||w||. Surge assim o seguinte problema de optimização:

$$\min_{(w,b)} \ \frac{1}{2}||w||^2 \tag{2.20}$$

$$\text{Com as restrições:} \quad y_i(w \cdot x_i + b) - 1 \geq 0; \forall i = 1, ..., n \tag{2.21}$$

As restrições são impostas para assegurar que não haja dados de treino entre as margens de separação das classes. Por esse motivo, a SVM obtida é também designada de SVM com margens rígidas.

Se a função objectivo for convexa e os pontos que satisfazem as restrições formarem um conjunto convexo, este problema possui um único mínimo global. Estes requisitos podem ser resolvidos com a introdução de uma função de Lagrange, que tem associados os conhecidos multiplicadores de Lagrange, $\alpha_i$, da Equação 2.22.

$$L(w, b, \alpha) \ = \ \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i (y_i(w \cdot x_i + b) - 1) \tag{2.22}$$

Minimizar a função de Lagrange implica maximizar as variáveis $\alpha_i$ e minimizar $w$ e $b$. Tal ocorre nos chamados pontos de sela, nos quais:

$$\frac{\partial L}{\partial b} = 0 \quad e \quad \frac{\partial L}{\partial w} = 0 \tag{2.23}$$

A resolução destas equações leva aos resultados representados nas expressões 2.24 e 2.25.

$$\sum_{i=1}^{n} \alpha_i y_i \;=\; 0 \tag{2.24}$$

$$w \;=\; \sum_{i=1}^{n} \alpha_i y_i x_i \tag{2.25}$$

Utilizando estes resultados na Equação 2.22, obtém-se o seguinte problema de optimização:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \;\; (x_i \cdot x_y) \tag{2.26}$$

$$\text{Com as restrições:} \quad \begin{cases} \alpha_i \geq 0, \forall_i = 1, ..., n \\ \sum\limits_{i=1}^{n} \alpha_i y_i = 0 \end{cases} \tag{2.27}$$

Esta formulação é denominada forma dual do problema primal. A forma dual apresenta restrições mais simples e permite a representação do problema de optimização em termos de produtos internos entre dados, o que é especialmente útil em SVM não linear.

Obtida a solução $\alpha^*$ do problema dual, o valor de w* pode ser determinado pela Equação 2.25. O parâmetro b* é definido por $\alpha^*$ e por condições de Kühn-Tucker, provenientes da teoria de optimização com restrições, que devem ser satisfeitas no ponto óptimo. Para o problema dual formulado é válida a equação:

$$\alpha_i^* (y_i (w^* \cdot x_i + b^*) - 1 \;=\; 0, \forall_i = 1, ..., n \tag{2.28}$$

Desta equação constata-se que $\alpha_i^*$ somente pode ser diferente de 0 para os dados que se encontram sobre os hiperplanos $H_1$ e $H_2$. Estes são os exemplos que se situam mais próximos do hiperplano separador, exactamente sobre as margens. Para os outros casos $\alpha_i^* = 0$. Esses pontos não participam portanto no cálculo de w* (Equação 2.25). Os

dados que possuem $\alpha_i^* > 0$ são denominados *Support Vectors* (SV) e podem ser considerados os dados mais informativos do conjunto de treino, pois somente eles participam na determinação da equação do hiperplano separador (Equação 2.31). O valor de b* é calculado a partir do número de SV (nSV) através da Equação 2.29.

$$b^* = \frac{1}{n_{SV}} \sum_{x_j \in SV} \frac{1}{y_j} - w^* \cdot x_j \tag{2.29}$$

Substituindo w*, dado pela Equação 2.25, resulta a Equação 2.30

$$b^* = \frac{1}{n_{SV}} \sum_{x_j \in SV} (\frac{1}{y_j} - \sum_{x_j \in SV} \alpha_i^* y_i x_i \cdot x_j) \tag{2.30}$$

A partir de w* e b* obtem-se o classificador g(x) apresentado na Equação 2.31 que representa o hiperplano que separa os dados com maior margem.

$$g(x) = sgn(f(x)) = sgn(\sum_{x_i \in SV} y_i \alpha_i^* x_i \cdot x + b^*) \tag{2.31}$$

Em situações reais é difícil encontrar aplicações cujos dados sejam linearmente separáveis. Este facto deve-se a diversos factores, entre eles a presença de ruídos e *outliers* nos dados ou à própria natureza do problema, que pode ser não linear. SVM linear de margens rígidas deve assim ser adaptada para lidar com conjuntos de treino mais gerais. Para realizar essa tarefa, permite-se que alguns dados possam violar a restrição da Equação 2.21. Isso é feito com a introdução de variáveis de folga $\xi_i$, para todo $i = 1, ..., n$. Essas variáveis transformam as restrições impostas ao problema de optimização primal, que se tornam:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \geq 0, \forall_i = 1, ..., n \tag{2.32}$$

A aplicação desse procedimento suaviza as margens do classificador linear, permitindo que alguns dados permaneçam entre os hiperplanos $H_1$ e $H_2$ e também a ocorrência de alguns erros de classificação. Por esse motivo, as SVM obtidas neste caso dizem-se de margens suaves.

Há ainda muitos casos em que não é possível dividir satisfatoriamente os dados de treino por um hiperplano. Um exemplo é apresentado na Figura 2.11 (a), em que o uso de uma fronteira curva seria mais adequada na separação das classes.

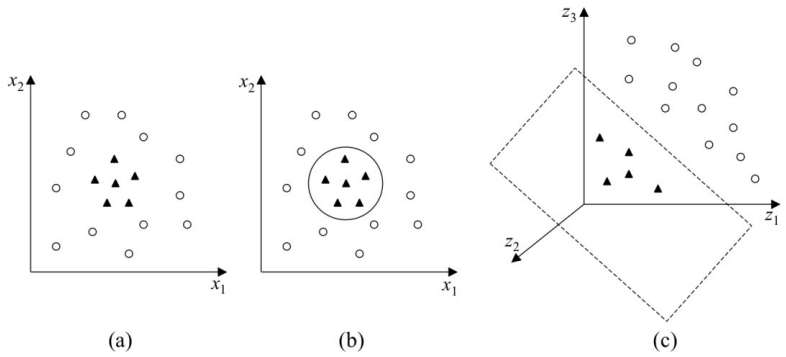



Figura 2.11: (a) Conjunto de dados não linear; (b) Fronteira não linear no espaço de entradas; (c) Fronteira linear no espaço de características

Nestes casos SVM transforma o conjunto de treino do espaço de entradas, num novo espaço de maior dimensão, denominado espaço de características (*featurespace*). Seja $\Phi : X \rightarrow \Im$ um mapeamento, em que X é o espaço de entradas e $\Im$ representa o espaço de características. A escolha apropriada de $\Phi$ faz com que o conjunto de treino mapeado em $\Im$ possa ser separado por uma SVM linear. Para ilustrar esses conceitos, considere o conjunto de dados apresentado na Figura 2.11(a). Transformando os dados de $\Re^2$ para $\Re^3$ com o mapeamento representado na Equação 2.33, o conjunto de dados não linear em $\Re^2$ torna-se linearmente separável em $\Re^3$ como se vê na Figura 2.11(c). É possível então encontrar um hiperplano capaz de separar esses dados, descrito na Equação 2.25. Pode-se verificar que a função apresentada, embora linear em $\Re^3$ (Figura 2.11(c)), corresponde a uma fronteira não linear em $\Re^2$ (Figura 2.11(b)).

$$\Phi(x) = \Phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \tag{2.33}$$

$$f(x) = w \cdot \Phi(x) + b = w_i x_i^2 \sqrt{2}x_1x_2 + w_3 x_2^2 + b = 0 \tag{2.34}$$

O problema de optimização representado na Equação 2.26 passa agora a apresentar a forma apresentada na Equação 2.35

$$Maximizar \sum_{i=1}^{n} \alpha_i + \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j)) \tag{2.35}$$

O classificador extraído passa a ser o indicado na Equação 2.36

$$g(x) = sgn(f(x)) = sgn(\sum_{x_i \in SV} \alpha_i^* y_i \Phi(x_i) \cdot \Phi(x) + b^*) \tag{2.36}$$

em que b* é dado pela Equação 2.37

$$b^* = \frac{1}{n_{SV:\alpha^*<C}} \sum_{x_j \in SV:\alpha_j^*<C} (\frac{1}{y_j} - \sum_{x_i \in SV} \alpha_i^* y_i \Phi(x_i) \cdot \Phi(x_j)) \tag{2.37}$$

onde a constante C é um termo de regularização que impõe um peso à minimização dos erros no conjunto de treino em relação à minimização da complexidade do modelo.

Há problemas em que $\Im$ pode ter dimensão muito alta (até mesmo infinita). A computação de $\Phi$ pode ser então extremamente custosa ou inviável. Das equações 2.35, 2.36 e 2.37 depreende-se que a única informação necessária sobre o mapeamento é sobre o cálculo de produtos escalares entre os dados no espaço de características. Isso consegue-se recorrendo a funções denominadas Kernels.

Um Kernel, K, é uma função que recebe dois pontos $x_i$ e $x_j$ do espaço de entradas e computa o produto escalar desses dados no espaço de características. Tem-se então:

$$K(x_i, x_j) \quad = \quad \Phi(x_i) \cdot \Phi(x_j) \tag{2.38}$$

Para o mapeamento apresentado na Equação 2.33 e dois dados $x_i = (x_{1i}, x_{2i})$ e $x_j = (x_{1j}, x_{2j})$ em $\Re^2$, por exemplo, o Kernel é dado por:

$$K(x_i, x_j) = (x_{1i}^2, \sqrt{2}x_{1i}x_{2i}, x_{2i}^2) \cdot (x_{1j}^2, \sqrt{2}x_{1j}x_{2j}, x_{2j}^2) = (x_i \cdot x_j)^2 \tag{2.39}$$

É comum empregar a função Kernel sem conhecer o mapeamento, que é gerado implicitamente. A utilidade dos Kernels está, portanto, na simplicidade de seu cálculo e na sua capacidade de representar espaços abstractos. Para garantir a convexidade do problema de optimização formulado na Equação 2.35 e também que o Kernel represente mapeamentos nos quais seja possível o cálculo de produtos escalares conforme a Equação 2.38, utilizam-se funções Kernel que seguem as condições estabelecidas pelo Teorema de Mercer. De forma simplificada, um Kernel que satisfaz as condições de Mercer é caracterizado por dar origem a matrizes semi-definidas positivas K, em que cada elemento $K_{ij}$ é definido por $K_{ij} = K(x_i, x_j)$, para todo $i, j = 1, ..., n$.

Alguns dos Kernels mais utilizados na prática são os Polinomiais, os Gaussianos ou *Radial-Basis Function* (RBF) e os Sigmoidais, listados na Tabela 2.1. Todos apresentam

parâmetros que devem ser determinados quando forem utilizados.

Tabela 2.1: Funções Kernel mais comuns

| **Tipo de Kernel** | **Função** $K(x_i, x_j)$ | **Parâmetros** |
|---|---|---|
| Polinomial | $(\delta(x_i \cdot x_j) + k)^2$ | $\delta$, k e d |
| Gaussiano | $exp(-\sigma\|\|x_i - x_j\|\|^2)$ | $\sigma$ |
| Sigmoidal | $tanh(\delta(x_i \cdot x_j) + k)$ | $\delta$ e k |

Convém referir finalmente que em SVM a convexidade do problema de optimização formulado no treino implica a existência de um único mínimo global. Além disso, o uso de funções Kernel na não-linearização de SVM torna o algoritmo eficiente, pois permite a construção de hiperplanos em espaços de grande dimensão o que os torna atractivos do ponto de vista computacional. Entre as principais limitações de SVM encontram-se a sua sensibilidade à escolha de valores de parâmetros e a dificuldade de interpretação do modelo gerado por esta técnica.

#### 2.1.2.5 Partição Recursiva

Partição Recursiva (RP) [DD04] é uma técnica estatística de análise multi-variante que tem por objectivo a construção de árvores de decisão que modelam a influência de uma serie de variáveis sobre a função objectivo. A Figura 2.12 ilustra a metodologia geral envolvida na Partição Recursiva.

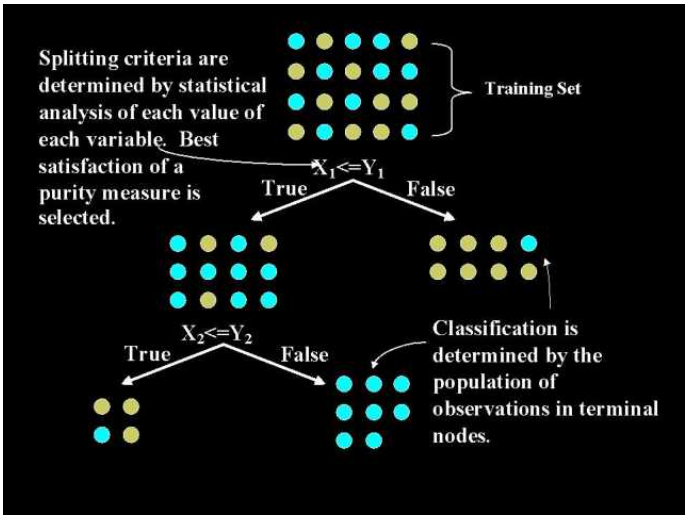



Figura 2.12: Partição Recursiva

Para entender o processo de RP considere-se que cada pergunta pode ter uma resposta verdadeira ou falsa e portanto qualquer nó em particular terá no máximo dois caminhos que levam para o próximo nó. O resultado é a divisão dos dados, em cada nó, em dois grupos independentes - daí a designação de *Partição*. Uma vez que existem dois novos

nós (nós filhos) ligados a um nó anterior (nó pai), o processo repete-se para cada nó filho independente, utilizando apenas as observações presentes no nó filho - daí a designação de *Recursivo*.

As árvores de decisão são modelos estatísticos que utilizam a estratégia de dividir para conquistar: um problema complexo é decomposto em sub-problemas mais simples e recursivamente esta técnica é aplicada a cada sub-problema [Gam04]. A capacidade de discriminação de uma árvore advém da divisão do espaço definido pelos atributos em sub-espaços e a cada sub-espaço ser associada uma classe.

A Figura 2.13 esquematiza uma árvore de decisão.

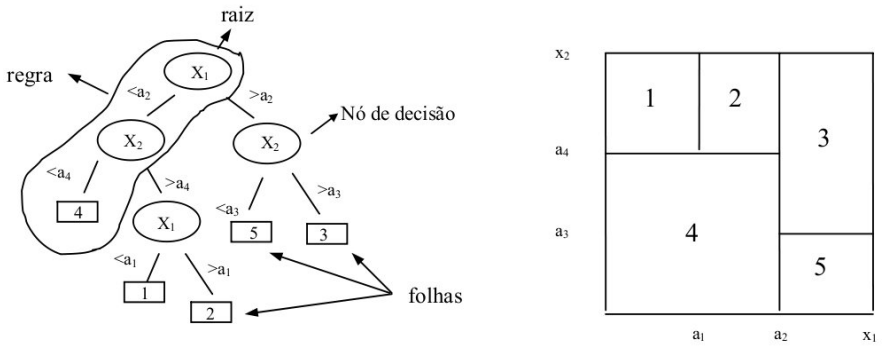



Figura 2.13: Representação de uma árvore de decisão e respectiva representação no espaço

Analisando a figura pode concluir-se que em cada nó de decisão é feito um teste ao atributo, cada ramo corresponde a um possível valor deste atributo e a cada folha está associada a uma classe. A intersecção dessas classes é vazia e a união o espaço completo. Cada percurso na árvore (da raiz à classe) corresponde a uma regra de classificação. No espaço definido pelos atributos cada folha corresponde a um hiper-rectângulo.

O critério utilizado para realizar as partições é o da "utilidade" do atributo para a classificação. O atributo escolhido como atributo teste para um dado nó é aquele que possuir o maior ganho de informação. Nos casos em que a árvore é usada para classificação, os critérios de partição mais conhecidos são baseados na entropia e índice Gini [Ono01].

A entropia pode ser entendida como uma medida da (im)pureza dos dados: num conjunto de dados, é uma medida da falta de homogeneidade dos dados de entrada em relação à sua classificação. Por exemplo, a entropia é máxima (igual a 1) quando o conjunto de dados é heterogéneo [Mit97]. Dado um conjunto de entrada (D) que pode ter c classes distintas, a entropia de D será dada pela equação 2.40.

$$\text{Entropia}(D) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2.40}$$

onde $p_i$ é a proporção de dados em D que pertencem à classe i.

Sendo P(A) o conjunto dos valores que um atributo A de um conjunto de dados D pode assumir, x um elemento deste conjunto, $D_x$ o subconjunto de D formado pelos dados em que A = x.
A entropia que se obtém ao efectuar a partição de D em função do atributo A é dada pela equação 2.41

$$E(A) = \sum_{x \in P(A)} \frac{|D_X|}{|D|} \text{Entropia}(D_X) \tag{2.41}$$

O ganho de informação será dado pela equação 2.42

$$\text{ganho}(D,A) = \text{Entropia}(D) - E(A) \tag{2.42}$$

onde Entropia(D) é uma medida de não-homogeneidade do conjunto D e E(A) uma medida de não-homogeneidade estimada para o conjunto D caso utilize o atributo A para fazer a próxima partição.

O índice Gini mede o grau de heterogeneidade dos dados. Logo, pode ser utilizado para medir a "impureza"de um nó [Ono01]. Este índice num determinado nó é dado pela equação 2.43

$$\text{Indice Gini} = 1 - \sum_{i=1}^{c} p_i^2 \tag{2.43}$$

onde $p_i$ é a frequência relativa de cada classe em cada nó e c é o número de classes. Quando este índice é igual a zero, o nó é puro. Quando se aproxima do valor 1, o nó é impuro (aumenta o número de classes uniformemente distribuídas neste nó). Quando, nas árvores de classificação com partições binárias, se utiliza o critério de Gini tende-se a isolar num ramo os registos que representam a classe mais frequente.

Num algoritmo de *Partição Recursiva*, a árvore estende a sua profundidade até classificar perfeitamente os elementos do conjunto de treino. Quando o conjunto de treino não possui ruído, o número de erros no treino pode ser zero. Quando o conjunto de treino possui ruído ou quando não é representativo, estes algoritmos podem produzir árvores em que há sobre-ajustamento (*overfitting*). Segundo Gama [Gam04] uma árvore de decisão $d$ faz sobre-ajustamento dos dados se existir uma árvore $d'$ tal que $d$ tem menor erro que $d'$ no conjunto de treino mas $d$ tem maior erro na população.

Para saber qual é o número óptimo de nós, representa-se graficamente o percentual de erro no conjunto de treino e teste *versus* o número de nós da árvore. Quando o erro no conjunto de teste começar a crescer, considera-se o número de nós nesse ponto.

Segundo Breiman [Bre01], para prevenir o problema do *overfitting* e melhorar a classificação deve-se realizar a poda da árvore. O processo de poda pode ser feito do seguinte modo: primeiro percorrer toda a árvore, segundo calcular o erro em cada nó de decisão e a soma dos erros nos nós descendentes e finalmente, se o erro do nó é menor ou igual à soma dos erros dos nós descendentes, o nó é transformado em folha.

Na Figura 2.14, tem-se um exemplo de poda [Sil05]. Tomando, por exemplo, o nó B que tem um erro de 2 e como a soma dos erros nos nós descendentes é 2 + 0 = 2, o nó B deve transformar-se em folha.

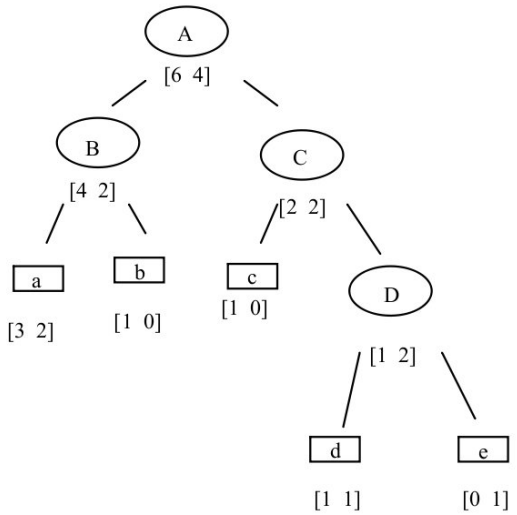



Figura 2.14: Árvore original

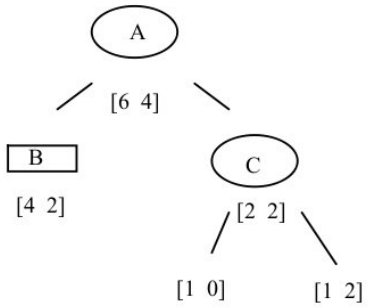



Figura 2.15: Árvore após poda

Concluindo, pode-se referir que os problemas encontrados em algoritmos recursivos são: obtenção de estimativas fiáveis do erro usado para seleccionar as partições e optimização do erro no treino e da dimensão da árvore.

#### 2.1.2.6 Random Forest

*Random Forest* [Bre01] é um classificador constituído por uma colecção de árvores de classificação $\{h(x,\Theta_k), k = 1,...\}$ onde $\{\Theta_k\}$ são vectores aleatórios independentes identicamente distribuídos e cada árvore atribui uma classificação à entrada X.

Neste método, cada árvore é construída da seguinte forma:

- Os dados são retirados de um conjunto de treino, X, por um processo de amostragem aleatória *bootstrap*, isto é, que se obtém por amostragem de X usando o método de amostragem aleatória simples com reposição, sendo formado por apenas 2/3 dos dados usados para construir a árvore.

- Os dados que não são seleccionados nessa amostra são nomeados como dados *out-of-bag*[1].
- Um número aleatório de atributos é seleccionado do conjunto de características e é identificado o atributo com maior informação em cada nó.
- Continua-se o processo de aumento da árvore até que nenhum nó possa ser criado devido ao facto de não aumentar a informação.

Atendendo ao processo de construção das árvores podemos afirmar que *Random Forest* converge e não origina $overfiting$. Para o mostrar consideremos um conjunto de classificadores $h_1(x),\ h_2(x),...,\ h_K(x)$ e um conjunto de treino proveniente aleatoriamente da distribuição dos vectores Y, X. Define-se função marginal através da equação 2.44.

$$mg(X,Y) = av_k I(h_k(X) = Y) - \max_{j \neq Y} av_k I(h_k(X) = j) \tag{2.44}$$

onde $I(\cdot)$ é a função informação.

O erro generalizado é dado pela equação 2.45.

$$PE^* = P_{X,Y}(mg(X,Y) < 0) \tag{2.45}$$

onde o índice $X,Y$ indica que a probabilidade provém do espaço $X,Y$.

Em *Random Forest* $h_k(X) = h(X,\Theta_k)$.
Para um grande número de árvores resulta, da Lei dos Grandes Números e da estrutura das árvores, que à medida que o número de árvores aumenta, $PE^*$ converge para equação 2.46.

$$P_{X,Y}(P_\Theta(h(X,\Theta) = Y) - \max_{j \neq Y} P_\Theta(h(X,\Theta) = j) < 0 \tag{2.46}$$

Este resultado explica o motivo de *Random Forest* não originar $overfiting$, por maior que seja o número de árvores adicionadas, mas sim convergir para um valor limite do erro generalizado.

Algumas das características deste tipo de classificador são:

- Em *Random Forest* não há necessidade de *cross-validation*, ou de um teste separado para obtenção de uma estimativa imparcial do erro de teste. Isto porque cada árvore é construída usando uma amostra *bootstrap*, diferente dos dados originais. Além

[1] out-of-bag: Conjunto de dados que não são utilizados para treino na construção da árvore.

disso a terça parte dos casos omitidos não são usados na construção da árvore e são colocados abaixo dela e faz-se um teste de classificação. No fim da execução tem-se $C_j$, que será a classe que conseguiu a maioria dos votos. Com a proporção de vezes em que $C_j$ não era igual à classe verdadeira $n$ calcula-se a média sobre todos os casos e assim é estimado o erro de *out-of-bag*.

- A taxa de erro em Random Forest depende de dois aspectos:
  - A correlação entre duas árvores quaisquer na floresta. Aumentando-se a correlação, aumenta-se a taxa de erro da floresta.
  - A força de cada árvore na floresta. Uma árvore com uma taxa de erro baixa, é um classificador forte. Aumentando-se a força das árvores individuais, diminui-se a taxa de erro da floresta.
- *Random Forest* tem melhor precisão que a maioria dos algoritmos actuais, mostrando-se eficaz para bancos de dados grandes. Fornece estimativas das variáveis que são importantes na classificação, gerando uma estimativa imparcial interna do erro de generalização.
- *Random Forest* geradas podem ser salvas para uso futuro com outros dados. São construídos protótipos que dão informações sobre a relação entre as variáveis e a classificação, e ainda proximidades entre pares de casos que podem ser usados em "clustering".

## 2.2 Programação Lógica Indutiva (ILP)

A Programação Lógica Indutiva [LD96] desenvolve técnicas e ferramentas para *Relational Data-Mining* (RDM) [LD01]. Num banco de dados relacional típico os dados estão distribuídos por várias tabelas. As ferramentas de ILP podem ser aplicadas directamente a dados multi-relacionais para encontrar padrões que envolvam múltiplas relações; esta é uma característica das abordagens da ILP. Em contraste, a maioria das outras abordagens de *Data Mining*, incluindo as apresentadas até agora, só pode lidar com dados que figuram numa única tabela e necessitam de pré-processamento para integrar dados de várias tabelas (por exemplo, através de associações ou agregação).

A integração de dados de várias tabelas através de associações ou agregação pode, no entanto, causar perda de sentido ou informação.

Supondo que é dada a relação *cliente* (CustoID, Nome, Idade, GastamMuito) e a relação de *compra* (CustoID, PrudutoID, Data, Valor, MetodoPagamento) [LD01], onde cada cliente pode fazer várias compras, e pretende-se caracterizar os clientes que gastam muito. Integrando as duas relações através de uma junção natural dar-se-á origem a uma relação *compra*1 onde cada linha corresponde a uma compra.

Outra possível agregação daria origem à relação *cliente*1 (CustoID, Idade, NCompras, ValorTotal, GastamMuito). Neste caso, no entanto, algumas informações seriam perdidas durante o processo de agregação.

Por outro lado, se a relação *cliente* e relação de *compra* forem consideradas em conjunto, o seguinte padrão pode ser descoberto por um sistema de ILP

$$\begin{aligned} Cliente(CID, Nome, Idade, yes) \leftarrow Idade > 30 \wedge \\ Compra(CID, PID, D, Valor, MP) \wedge \\ MP = cartao\_credito \wedge \\ Valor > 100. \end{aligned} \tag{2.47}$$

Este padrão diz: "Um cliente gasta muito se tiver mais de 30 anos, comprou um produto de valor superior a 100 e pagou com cartão de crédito."

Não seria possível chegar a tal padrão se fossem considerados apenas *compra*1 ou *cliente*1 individualmente.

Além da capacidade de poder usar dados armazenados em várias tabelas, os sistemas de ILP geralmente são capazes de utilizar informação extra sobre o domínio de conhecimento na forma de um programa lógico.

Do esquema apresentado na Figura 2.16 conclui-se que ILP utilizando Programação em Lógica produz regras expressas em lógica de primeira ordem. Usando mecanismos de inversão consegue implementar indução, Aprendizagem Indutiva.


Figura 2.16: Programação Lógica Indutiva

De forma resumida pode afirmar-se que, dado um conjunto de exemplos positivos e negativos, um sistema de ILP encontra uma descrição lógica que os diferencia. Geralmente, esta descrição consiste num conjunto de regras ou cláusulas, formando um programa lógico. Neste caso, os exemplos desconhecidos são aplicados a cada cláusula, sucessivamente, formando uma lista de decisão. Se o exemplo é coberto por uma regra da Teoria, recebe a classificação positiva; se o exemplo não é coberto por nenhuma das regras, recebe a classificação negativa.

Num mundo ideal, onde os exemplos seriam perfeitamente discriminados entre duas classes, a lista de decisão representaria um esquema de combinação ideal. Na prática, é difícil encontrar regras que não cubram exemplos negativos, aumentando assim o número de falsos positivos.

### 2.2.1 Aleph - *A Learning Engine for Proposing Hypotheses*

O sistema Aleph [Sri93] é um sistema de ILP escrito em Prolog, que foi inicialmente conhecido como P-Progol [SC93]. O sistema Aleph foi desenvolvido com o intuito de ser um protótipo para explorar ideias de ILP.

O Aleph possui uma poderosa linguagem de representação que permite representar expressões complexas e incorporar com facilidade novo conhecimento do domínio. Além dessas características comuns aos sistemas de ILP, o Aleph possui outras características muito interessantes, tais como: permitir escolher qual a ordem de geração das regras (geral para o particular ou particular para o geral); mudar a função de avaliação (Entropia, Precisão, Laplace, entre outras) e mudar a ordem de busca.

#### 2.2.1.1 Como funciona o Aleph

Aleph implementa vários algoritmos sendo o mais conhecido o *Mode Direct Inverse Entailment* (MDIE) [LD96] que constrói incrementalmente as cláusulas por meio do seguinte algoritmo:

1. Seleccionar um exemplo positivo a ser generalizado.
2. Construir a cláusula mais específica que envolve o exemplo seleccionado e está dentro das limitações da linguagem fornecida. Isto é normalmente uma cláusula definida com muitos literais, e é chamada de *bottom clause*.
3. Encontrar uma cláusula mais geral do que a *bottom clause*. Isto é feito através da procura de um subconjunto dos literais na *bottom clause* que tenha melhor *score*. Guardar a cláusula se esta for melhor que as que já se tinham encontrado.
4. A cláusula com a melhor pontuação é adicionada à teoria corrente, e todos os exemplos redundantes são removidos.
5. Se exemplos positivos não forem cobertos, reiniciar o processo.

#### 2.2.1.2 Utilização de Aleph

Aleph requer três ficheiros para a construção de teorias. O uso mais simples de Aleph implica:

1. Construir os três ficheiros de dados chamados *filestem.b*, *filestem.f*, *filestem.n*

2. Ler todos os dados usando o comando *read_all (filestem).*

3. Construir uma teoria usando o comando *induce.*

Apresenta-se de seguida um exemplo explicativo do funcionamento do algoritmo: *Michalski's Train Problem*

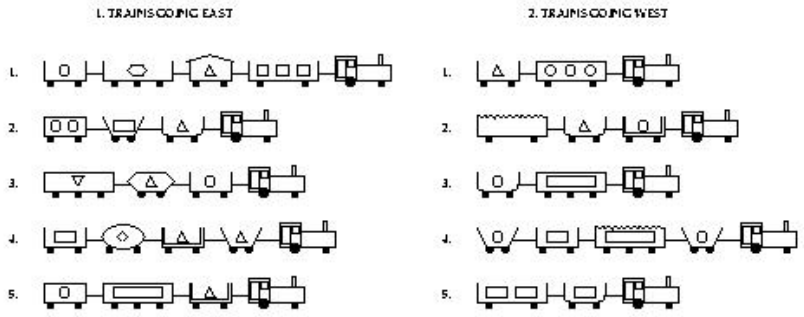



Figura 2.17: Michalski's Train Problem

- *Background knowledge*

  *Background knowledge* está contido num ficheiro com uma extensão *.b. O *Background knowledge* encontra-se na forma de cláusulas Prolog que codificam a informação relevante para o domínio. Pode também conter directivas para o compilador Prolog. Este ficheiro também contém linguagem e as restrições de procura para Aleph. As mais básicas referem-se a *modes*, *types* e *determinations*.

  Alguns exemplos contidos no ficheiro *.b:

```
:- modeh(1,eastbound(+train)).
:- modeb(1,short(+car)).
short(car_12).
closed(car_12).
long(car_11).
:- determination(eastbound/1,short/1).
:- determination(eastbound/1,closed/1).
```

- Exemplos Positivos

  Exemplos positivos de um conceito a ser aprendido são escritos num ficheiro com uma extensão *.f. Os exemplos positivos são simplesmente factos. Alguns exemplos contidos neste ficheiro:

```
eastbound(east1).
```

```
eastbound(east2).
eastbound(east3).
eastbound(east4).
eastbound(east5).
```

- Exemplos Negativos

  Exemplos negativos de um conceito a ser aprendido são escritos num ficheiro com uma extensão *.n. Os exemplos negativos são simplesmente factos. Alguns exemplos contidos neste ficheiro:

```
eastbound(west1).
eastbound(west2).
eastbound(west3).
eastbound(west4).
eastbound(west5).
```

- Ler todos os ficheiros de *input*

  Depois dos ficheiros: *filestem.b filestem.f* e *filestem.n* serem construídos, podem ser lidos pelo Aleph através do comando:

```
read_all(filestem).
```

- Construir a teoria

  O comando básico para seleccionar exemplos e construir teorias é:

```
induce.
```

O código Aleph está contido num único arquivo, normalmente chamado de "aleph.pl". Para carregar o Aleph, é necessário consultar esse arquivo através de um compilador Prolog: o *Yet Another Prolog* (Yap) [CL] realiza essa tarefa.

São listadas as cláusulas que foram procuradas, juntamente com a cobertura de exemplos positivos e negativos:

```
   ?- induce.
...
[5/5]
eastbound(A) :-
   has_car(A,B).
[5/5]
eastbound(A) :-
   has_car(A,B), long(B).
[2/5]
eastbound(A) :-
   has_car(A,B), open_car(B).
...
```

Encontra-se uma cláusula::

```
...
eastbound(A) :-
   has_car(A,B), closed(B), load(B,triangle,1).
[2/0]
[----------------------------------]
[found clause]
eastbound(A) :-
   has_car(A,B), closed(B), load(B,triangle,1).
[pos cover = 2 neg cover = 0] [pos-neg] [2]
[clause label] [[2,0,4,2]]
[clauses constructed] [34]
[----------------------------------]
...
```

Encontra-se a melhor cláusula:

```
...
eastbound(A) :-
   has_car(A,B), short(B), closed(B).
[5/0]
[----------------------------------]
[found clause]
eastbound(A) :-
   has_car(A,B), short(B), closed(B).
[pos cover = 5 neg cover = 0] [pos-neg] [5]
[clause label] [[5,0,4,5]]
[clauses constructed] [70]
[----------------------------------]
[clauses constructed] [70]
[search time] [0.004]
[best clause]
eastbound(A) :-
   has_car(A,B), short(B), closed(B).
[pos cover = 5 neg cover = 0] [pos-neg] [5]
[atoms left] [0]
[positive examples left] [0]
[estimated time to finish (secs)] [0.0]
...
```

O resultado final é apresentado da seguinte forma:

```
...
[theory]

[Rule 1] [Pos cover = 5 Neg cover = 0]
eastbound(A) :-
```

```
   has_car(A,B), short(B), closed(B).
...
```

O comando *induce.* mostra também a performance dos resultados obtidos numa *matriz de confusão*, discutida em mais detalhe na secção 3.2.1.

```
...

[Training set performance]
          Actual
       +        -
     + 5        0        5
Pred
     - 0        5        5

       5        5        10

Accuracy = 1.0
[Training set summary] [[5,0,0,5]]
[time taken] [0.008]
[total clauses constructed] [70]
```

Por análise desta matriz pode concluir-se que a regra encontrada (Rule 1) não classificou nenhum dos comboios incorrectamente (número de falsos positivos e número falsos negativos foram ambos 0).

# Capítulo 3

# SAYU

## 3.1 Aprendizagem Estatística Relacional

A Aprendizagem Estatística Relacional estuda os modelos dos domínios que apresentam tanto incerteza (que podem ser tratados através de métodos estatísticos) como estruturas relacionais complexas. Frequentemente os formalismos de representação do conhecimento desenvolvido em SRL usam lógica de primeira ordem para descrever as propriedades relacionais de um domínio de uma forma geral e recorrem a modelos probabilísticos gráficos (como redes Bayesianas [RN95] ou redes de Markov [RD06]) para modelar a incerteza. Alguns também recorrem aos métodos da Programação Lógica Indutiva para aprender as propriedades de SRL.

Vários estudos em Aprendizagem Relacional mostraram a eficácia de SRL em muitos domínios. Os mais bem sucedidos aceitam as regras relacionais candidatas, durante a aprendizagem relacional, pela sua capacidade de melhorar o classificador estatístico. Nos primeiros desses estudos, os sistemas nFOIL (Landwehr e outros., 2005) e SAYU (Davis e outros., 2005), o classificador é do tipo RB. Um sistema posterior, kFOIL (Landwehr e outros., 2006), usando Kernel permite efectuar simultaneamente classificação e regressão [DCRP].

Neste capítulo são abordados modelos SRL simples que são capazes de melhorar o desempenho de ILP através da combinação de regras.

## 3.2 SAYU - *Score As You Use*

Como referido anteriormente com ILP aprendem-se as regras para uma possível tarefa de classificação. A questão fundamental consiste em determinar como combinar as regras individuais para obter o melhor classificador possível.

Uma abordagem eficaz deste problema consiste em tratar cada regra aprendida como um atributo de aprendizagem e aprender um classificador para determinar a classificação final do exemplo. Esta metodologia define um processo em duas etapas. Na primeira etapa um algoritmo de ILP aprende um conjunto de regras. Na segunda etapa um classificador combina as regras aprendidas.

Neste modelo de aprendizagem relacional podem considerar-se três passos: aprendizagem, selecção e construção do modelo.

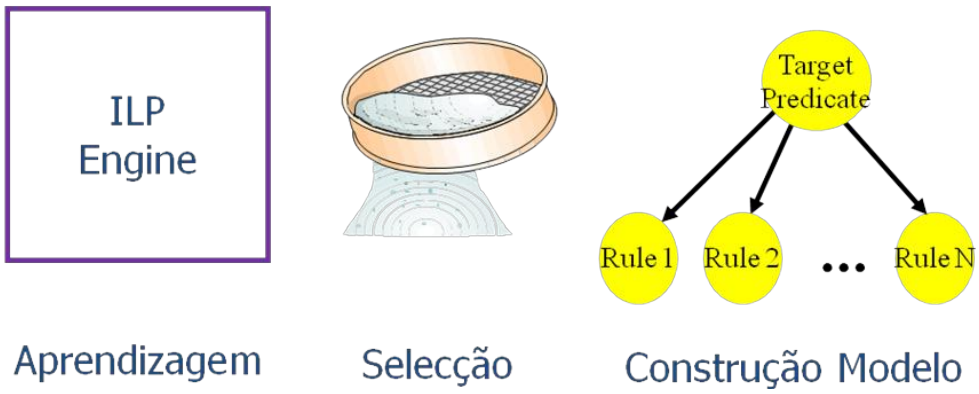



Figura 3.1: Aprendizagem Relacional em Múltiplos Passos

No primeiro passo é feita a aprendizagem usando por exemplo Programação Lógica Indutiva; no segundo passo é feita a escolha dos atributos (regras); no último passo é construído o modelo estabelecendo as relações entre o problema e os atributos como está especificado na Figura 3.1.

Um dos problemas desta abordagem é que as regras aprendidas na primeira etapa são avaliadas por uma métrica diferente da que é utilizada na segunda etapa. A pontuação das cláusulas do ILP tradicional é feita através de uma pontuação de cobertura ou por compressão métrica. Assim, não há nenhuma garantia de que o processo de aprendizagem vai seleccionar as regras que melhor contribuem para a classificação final.

Em alguns casos, convertendo cada regra de aprendizagem num recurso binário para Redes de Bayes melhora-se a precisão em comparação com a abordagem padrão de listas de decisão [DdCDOC04]. Isso origina um processo em duas etapas, onde as regras são geradas na primeira fase e o classificador é aprendido na segunda fase.

Um algoritmo que intercala as duas etapas, de forma incremental, construindo uma rede de Bayes durante a aprendizagem de regras é o algoritmo SAYU [DBdCD+05b].

O esquema de um algoritmo SAYU apresenta-se no Algoritmo 1.

```
Input: Train Set T, Tune Set S, Stop Criteria
Output: A TAN Model
M = BuildTANClassifier(T);
BestScore = AreaUnderPRCurve(M, S);
while Stop creteria not met do
    done = false;
    Choose a positive example as a seed and saturate the example;
    repeat
        NewFeature = Generate new clause according to saturated example;
        M_new = BuilTANClassifier(T ∪ NewFeature);
        NewFeature = AreaUnderPRCurve(M_new, S ∪ NewFeature);
        if NewFeature ≥ BestScore then
            T = T ∪ NewFeature;
            S = S ∪ NewFeature;
            BestScore = NewFeature;
            M = M_new;
            done = true;
        end
    until not(done);
end
return M;
```

**Algorithm 1:** Algoritmo SAYU

Neste algoritmo cada regra candidata é introduzida na rede e avaliada em relação à melhoria do desempenho do classificador utilizando Aleph com *Naive Bayes* e TAN. SAYU modifica a busca padrão de Aleph. Contrastando com Aleph, SAYU permite que todo o exemplo, positivo ou negativo, seja seleccionado como semente. Em vez de usar a cobertura, Aleph passa cada cláusula que constrói ao SAYU, que a converte numa característica binária e a adiciona ao conjunto de treino actual. Em seguida, SAYU aprende um modelo que incorpora a característica nova e avalia-o. Se o modelo não melhorar, a regra não é aceite, e o processo retorna a Aleph para construir a cláusula seguinte. Se uma regra for aceite, ou o espaço da busca estiver esgotado, SAYU selecciona aleatoriamente uma nova semente e reinicia a busca no Aleph. Assim, em cada passo, não encontramos a melhor regra, mas a primeira regra que melhora o modelo. Como SAYU permite que a mesma semente seja seleccionada várias vezes durante a busca e atendendo a que o espaço da busca é extremamente grande, é pouco prático efectuar uma busca exaustiva e pode mesmo conduzir a $overfitting$. Frequentemente recorre-se à designada *cross validation* no conjunto de dados de treino, terminando a busca ao fim do tempo previamente definido [DCRP].

### 3.2.1 Avaliação

Em algoritmos do tipo SAYU é fundamental poder comparar classificadores. Em geral, num problema de decisão binária procura-se aprender um classificador capaz de dar aos exemplos a classificação de positivos ou negativos. A decisão tomada pelo classificador pode ser representada por uma estrutura conhecida como *matriz de confusão*.

A *matriz de confusão* tem quatro categorias. Os verdadeiros positivos (TP) são exemplos correctamente classificados pelo classificador como positivos. Os falsos positivos (FP) referem-se a exemplos negativos incorrectamente classificados como positivos. Os verdadeiros negativos (TN) correspondem aos exemplos negativos correctamente classificados como negativos. Finalmente, os falsos negativos (FN) referem-se a exemplos positivos incorrectamente classificados como negativos.

A *matriz de confusão* é mostrada na Tabela 3.1.

Tabela 3.1: Matriz de Confusão

| | verdadeiros positivo | verdadeiros falso |
|---|---|---|
| previsto positivo | TP | FP |
| previsto falso | FN | TN |

A partir da *matriz de confusão* é possível definir as métricas utilizadas no espaço *Receiver Operator Characteristic* (ROC) ou no espaço *Precision-Recall* (PR), através das Equações 3.1, 3.2, 3.3, 3.4.

A Taxa de Falsos Positivos (FPR) mede a proporção de exemplos negativos que são erradamente classificados como positivos. A Taxa de Verdadeiros Positivos (TPR) mede a proporção de exemplos positivos que estão correctamente classificados. Nas curvas ROC, avaliando a variação dos exemplos positivos correctamente classificados com os exemplos negativos erradamente classificados, representa-se FPR no eixo horizontal e TPR no eixo vertical, Figura 3.2.

*Precision* mede a proporção de exemplos positivos entre os que foram classificados como positivos. *Recall* mede a proporção de exemplos positivos que estão correctamente classificados. Nas curvas PR, medindo-se a variação da precisão com a variação do *Recall* representa-se *Recall* no eixo horizontal e *Precision* no eixo vertical, Figura 3.3.

$$Precision = \frac{TP}{TP+FP} \tag{3.1}$$

$$Reccal = \frac{TP}{TP+FN} \tag{3.2}$$

$$TPR = \frac{TP}{TP+FN} \tag{3.3}$$

$$FPR = \frac{FP}{FP+TN} \tag{3.4}$$

A simples utilização da precisão dos resultados para avaliar um algoritmo pode ser enganadora.

Provost [PFK98] recomendou que, em problemas de decisão binários, se devem usar curvas ROC que avaliam a variação dos exemplos positivos correctamente classificados com os exemplos negativos erradamente classificados. Através destas curvas a performance dos algoritmos pode ser comparada (Figura 3.2).

Manning e Schutze [MSL00] usaram as curvas de PR, que avaliam a diminuição da precisão com o aumento do Recall (Figura 3.3), como alternativa para curvas ROC em tarefas com distribuição de classe muito enviesada.

As curvas PR podem mostrar diferenças entre algoritmos que não são visíveis nas curvas ROC. Na Figura 3.2 o algoritmo 1 domina em relação ao algoritmo 2 mas é significativamente pior no espaço PR (Figura 3.3).

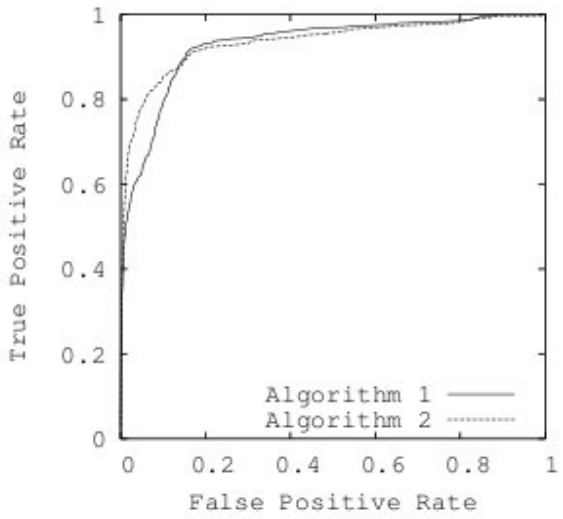



Figura 3.2: Curvas ROC

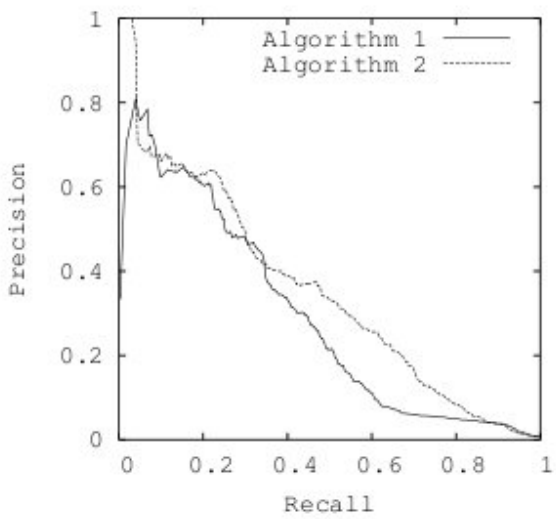



Figura 3.3: Curvas PR

Davis e Goadrich[DG06] mostraram que, em condições especiais, uma curva domina no espaço ROC se também for dominante no espaço PR.

### 3.2.2 Implementação do SAYU

SAYU [DBdCD+05b] exige, em primeiro lugar, um sistema ILP que proponha regras para a aprendizagem proposicional. Em segundo lugar, o algoritmo tem que decidir se mantém ou não uma cláusula. É preciso saber a pontuação de cada cláusula para efectuar a pesquisa de regras, o que torna necessária a existência de uma métrica de pontuação.

No SAYU existem 4 métricas de pontuação: *Accuracy*, *Conditional Log Likelihood*, *Area Under the Curve* ROC (AUC-ROC) e *Area Under the Curve* PR (AUC-PR). A métrica AUC-PR é a mais apropriada para casos de estudo com poucos casos positivos e muitos negativos.

De um modo muito sucinto pode-se afirmar que o sistema de ILP propõe uma regra e a converte num novo atributo. Essa regra é incorporada na aprendizagem; se a regra melhora a pontuação é guardada pelo classificador, caso contrário essa regra é desprezada e volta-se para o classificador inicial.

A implementação depende assim do sistema de ILP e da aprendizagem proposicional. Trabalhos já desenvolvidos utilizam sistemas de aprendizagem ILP, no estilo do algoritmo *Mode Direct Inverse Entailment* [LD96] usado em Progol [Mug95] e Aleph [Sri93].

Em MDIE, a ILP procura aleatoriamente um exemplo não explicado, satura-o e obtém desse modo a regra mais específica, ou cláusula saturada. Em seguida, procura o espaço de cláusulas que generalize a cláusula saturada até encontrar a melhor cláusula. O algoritmo usado pelo SAYU é um pouco diferente pois em vez de procurar a melhor cláusula, procura uma boa cláusula [DBdCD+05a].

A implementação do SAYU encontra-se esquematizada no Algoritmo 2.

```
Input: Stop Criteria, Scoring Function
Output: Propositional Classifier
CurrentScore = Init();
while Stop creteria not met do
    Choose a positive example as a seed and saturate the example;
    repeat
        NewFeature = Generate new clause according to saturated example;
        NewScore = NewAttribute(NewFeature);
        if NewScore exceeds CurrentScore then
            Commit();
        end
    until NewScore exceeds CurrentScore;
end
```

**Algorithm 2:** Implementação SAYU

O algoritmo proposicional de aprendizagem de Bayes tem várias vantagens. Primeiro, permite obter probabilidade para os exemplos. Em segundo lugar, *Naive Bayes* é um método bem conhecido e rápido que muitas vezes funciona bem na aprendizagem incremental. A desvantagem de *Naive Bayes* é que se assume que todas as regras são indepen-

dentes, dado o valor da classe [DP97]. TAN pode também ser utilizado de forma eficiente, e pode representar um conjunto limitado de dependências entre os atributos. Estes dois classificadores são os usados no SAYU [DBdCD+05b].

O objectivo da função de pontuação atrás referida é contribuir não só para a aprendizagem mas também para a avaliação. Técnicas como calcular a área da curva PR permitem lidar com conjuntos de dados que têm uma distribuição de classes altamente distorcidos.

### 3.2.3 Experiência

Trabalhos desenvolvidos com aplicação do sistema SAYU em análises de Mamografias [DBdCD+05a], revelaram que existe uma melhoria significativa dos resultados obtidos com este sistema, quando comparados com o sistema Aleph. Essa melhoria ocorre tanto no caso da aprendizagem com o classificador NB como com o classificador TAN. Tal pode ser constatado na Figura 3.4.

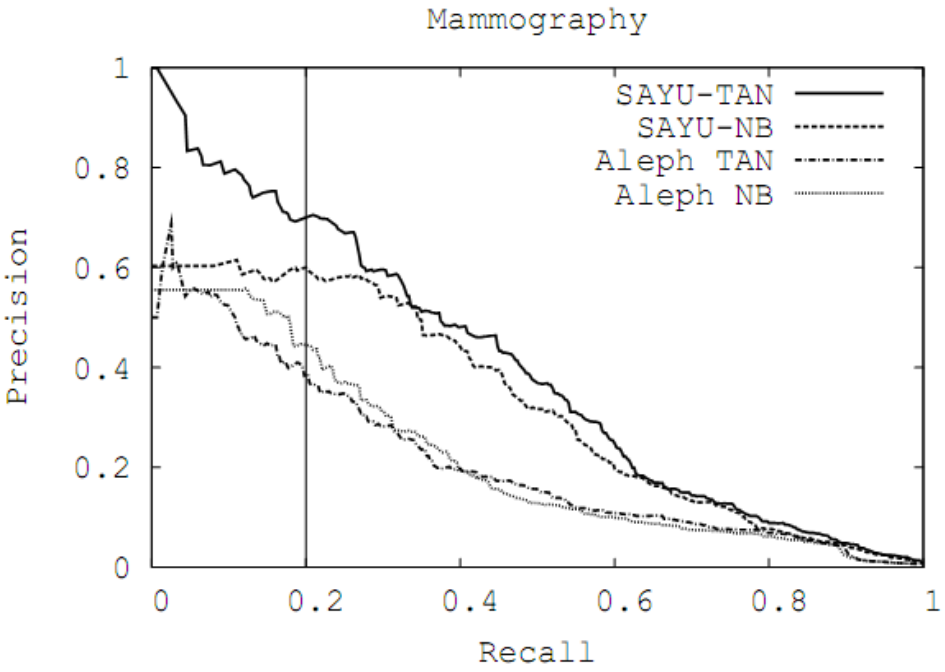



Figura 3.4: PR - SAYU vs Aleph

## 3.3 nFOIL - First-Order Logic com NB

O algoritmo ILP - *First-Order Logic* (FOIL) [Fit04] é um algoritmo alternativo capaz de aprender regras da lógica de primeira ordem. O sistema nFOIL [LKR05] integra o regime de aprendizagem *Naive Bayes* com o sistema de aprendizagem FOIL. Em contraste com outras combinações que usam NB só para o pós-processamento dos conjuntos de

regras o nFOIL, tal como o SAYU, usa o critério de NB directamente para orientar a pesquisa. Dados experimentais demonstram que nFOIL é mais eficiente que o algoritmo FOIL não só na fase de aprendizagem mas também nas abordagens pós-processamento e é, ao mesmo tempo, competitivo com abordagens mais sofisticadas.

A vantagem desta combinação simples de sistemas de aprendizagem é que o modelo probabilístico, lógico ou relacional resultante é fácil de entender e interpretar. Este algoritmo assemelha-se muito ao SAYU com *Naive Bayes* pois tem também no processo de classificação o classificador *Naive Bayes*.

## 3.4 kFOIL - First-Order Logic com Kernel

As abordagens que combinam ILP e Kernels [LPRF06] pretendem resolver problemas como a estabilidade (ou seja, a robustez ao ruído), a uniformidade (ou seja, tratar funções de classificação e funções de regressão de forma uniforme) e expressividade (explorar espaços de hipótese em domínios constituídos por objectos relacionais independentes). A ideia que está por trás do kFOIL é a de aplicar classificadores com Kernels.

No kFOIL existe uma construção dinâmica durante a aprendizagem (usando o algoritmo de cobertura FOIL-like) e pode ser visto efectivamente como uma produção adicional do problema de aprendizagem (para além da função de previsão). Nesse sentido, kFOIL é semelhante ao algoritmo ILP, nFOIL, que combina *Naive Bayes* e FOIL. Enquanto nFOIL assume uma direcção geradora de modelagem, kFOIL baseia-se na regularização empírica da minimização do risco.

kFOIL preserva todas as vantagens dos kernels, em particular, a uniformidade de representação em diferentes tarefas de aprendizagem supervisionadas, expressividade da linguagem de representação e capacidade de reutilização de conhecimento. A força de kFOIL é a sua capacidade para fornecer explicações adicionais sobre o domínio que pode ser lido no conjunto das cláusulas construídas.

Em alguns trabalhos experimentais o kFOIL obteve melhor performance que o nFOIL. Tal facto pode ser comprovado no trabalho de Landwehr, Passerini, Raedt & Frasconi sobre Aprendizagem Relacional com Kernels [LPRF06].

# Capítulo 4

# Algoritmo Proposto

## 4.1 Enquadramento

Como referido na secção 3.2.2, a função de pontuação pode contribuir não só para a avaliação mas também para a aprendizagem. Assim, tal como acontece com qualquer algoritmo que use uma métrica para avaliar se uma dada regra deve ou não ser incorporada, também com o SAYU se coloca uma escolha entre diferentes métricas. Conforme a métrica usada varia o rigor da avaliação e variam o número de regras. Como forma de endereçar este problema propõe-se, neste trabalho, dois tipos de abordagens. Inicialmente pensou-se em melhorar o SAYU através da implementação de um segundo classificador que confirmasse o resultado obtido pelo classificador usado no SAYU, integrando uma dada regra nos classificadores apenas quando passasse em dois testes estatísticos (Algoritmo I). Depois de avaliar os resultados experimentais obtidos verificou-se que não tinha existido diferença significativa em dois casos e noutro tinham sido um pouco menos satisfatórios. Surgiu então a ideia de enfraquecer o SAYU em termos de rigor, integrando-se de novo outro classificador independente, mas retirando o teste estatístico para incorporação da regra nos classificadores (Algoritmo II).

- Algoritmo I

  A proposta é de melhorar a classificação efectuada pelo SAYU através do seguinte procedimento:

  1. Efectuar uma primeira filtragem através do SAYU com um rigor de avaliação não muito exigente.
  2. Efectuar uma nova avaliação através de um classificador independente.
  3. Avaliar a significância da regra através de um teste estatístico.

4. Adicionar a nova regra apenas quando melhora *os dois* classificadores.

- Algoritmo II

  A proposta é de obter melhorias com o uso de dois classificadores 'fracos' através do seguinte procedimento:

  1. Efectuar uma primeira filtragem através do SAYU com um rigor de avaliação não muito exigente.
  2. Efectuar uma nova avaliação através de um classificador independente.
  3. Adicionar a nova regra apenas quando melhora *os dois* classificadores.

## 4.2 Integração do *R* no SAYU

Neste trabalho tornou-se importante o uso de diferentes classificadores. SAYU inclui dois classificadores, mas como são fortemente relacionados, decidiu-se escolher um classificador externo. Existem vários pacotes de software com classificadores para AA. *R* [R] foi o escolhido para fazer a integração com o SAYU pois possui um vasto número de classificadores e especialmente de estimadores, quando comparado com outros sistemas disponíveis como, por exemplo, o *Waikato Environment for Knowledge Analysis* (WEKA) [WEK].

### 4.2.1 Modo de funcionamento - Algoritmo I

Já foi referido que Aleph é carregado através do compilador Prolog. Neste trabalho usou-se o YAP [CL] que é um compilador Prolog de alta performance desenvolvido pelo Centro de Investigação em Sistemas Computacionais Avançados (CRACS) e pelo Laboratório de Inteligência Artificial e de Ciência de Computadores (LIACC/Universidade do Porto) para suporte de Redes Bayesianas e já inclui uma interface Java capaz de suportar o SAYU.

A implementação do *R* no sistema SAYU é realizada da seguinte forma:

- Fase I

  SAYU recebe os *dados de treino* e *dados de ajuste* (tune) de uma regra proveniente do YAP. Esses dados são usados não só para a construção do classificador SAYU mas também para obtenção do *score* através da função de avaliação AUC-PR. Analisa-se o *score* e se este for "significativamente"melhor passa-se para a fase seguinte.

- Fase II

Um outro classificador existente em *R* recebe também os *dados de treino* e *dados de ajuste* para a construção do classificador. Também aqui se recorre à função AUC-PR para analisar o *score* do classificador com a nova regra incorporada.

- Fase III

  É efectuado um teste estatístico para comparar os resultados obtidos pelo classificador com a nova regra e sem ela. Se existir diferença significativa então essa regra é incorporada nos dois classificadores.

A Figura 4.1 esquematiza o funcionamento de todo o processo do Algoritmo I.

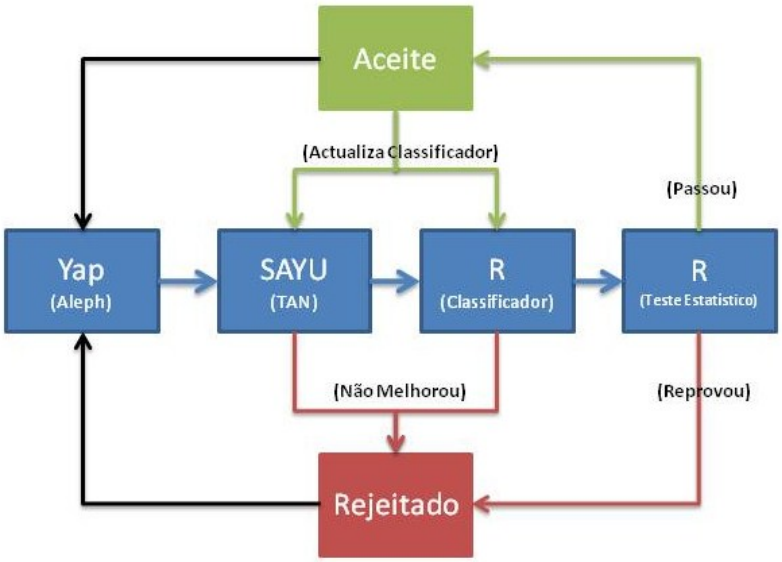



Figura 4.1: Algoritmo I

Analisando a Figura 4.1 constata-se que cada regra gerada por Aleph através de YAP é enviada para o SAYU. É feita uma avaliação pelo classificador do SAYU (Fase I). Se o *score* neste classificador aumentar por introdução dessa regra, os dados de treino e teste são enviados para o classificador do R. É feita uma avaliação pelo classificador do R (Fase II). Se a regra aumentar o *score* em ambos os classificadores é feito um teste estatístico que compara os resultados obtidos pelos classificadores com a nova regra e sem ela (Fase III). Se existir diferença significativa então essa regra é incorporada nos dois classificadores.

Dado que a Fase II implica um elevado tempo de uso do CPU, é feita uma filtragem depois da Fase I em que só as regras que melhoram pelo menos 70% dos *folds* são analisadas pelo segundo classificador. Deste modo, só serão analisadas pelo classificador R as melhores regras.

Com o intuito também de diminuir o tempo de processamento só é efectuado o teste estatístico referido na Fase III se o *score* médio do classificador R tiver aumentado e simultaneamente pelo menos 30% dos *folds* tiverem sido melhorados.

Integrou-se um classificador do R (rpart [TA], randomForest [BACW] ou ksvm [Kar]) para reforçar a análise do *score* proveniente da introdução de uma dada regra, um vez que os classificadores utilizados pelo SAYU, o classificador TAN e o classificador NB, são dois classificadores muito relacionados. A utilização do classificador de Partição Recursiva (rpart) deveu-se à sua simplicidade, ter um baixo custo do CPU e por vezes obter bons resultados. Utilizou-se também um classificador que recorre à construção de árvores de decisão, *Random Forest*, por ser um tipo de classificador muito utilizado e apresentar frequentemente resultados satisfatórios. O último classificador integrado foi o classificador *Support Vector Machines* com kernels, por ser um classificador para o qual se têm obtido bons resultados recentemente.

A integração do R com o SAYU não decorreu da forma esperada, pois não se conseguiu utilizar *software* já existente que liga R com Java (linguagem em que está implementado o SAYU). Por não se conseguir usar o JRI [JRI] para executar o R em Java utilizou-se *pipe* para tal efeito. A ligação entre R e SAYU (Java) é feita através da escrita/leitura de ficheiros com os dados que pretendem enviar um ao outro. Não só os dados das regras (bitemap) como também os resultados obtidos pelo classificador usado no R são enviados através de ficheiros.

Quanto à métrica, usou-se a que é mais utilizada na avaliação para os classificadores do SAYU, isto é, AUC-PR por ser a métrica que melhores resultados dá no uso de dados com muitos falsos positivos e por ter sido a mais utilizada em trabalhos anteriormente realizados nesta área. Para o cálculo de AUC-PR apenas se consideram valores de *recall* iguais ou superiores a 0.2, uma vez que as curvas de *Precision-Recall* apresentam muita instabilidade para valores de *recall* baixos.

Foram utilizados dois testes estatísticos adequados ao estudo de amostras emparelhadas: o teste paramétrico t-Student e o teste não paramétrico Wilcoxon. Para ambos utilizou-se uma significância de 5%.

### 4.2.2 Modo de funcionamento - Algoritmo II

Com este algoritmo pretende-se verificar se, recorrendo a dois classificadores "fracos", se consegue obter melhores resultados. A estrutura do algoritmo é semelhante à do Algoritmo I. No entanto, com o intuito de enfraquecer os classificadores, não se executa nem o teste estatístico t-student nem o wilcoxon para incorporar uma dada regra nos classificadores.

A Figura 4.2 esquematiza o funcionamento de todo o processo do Algoritmo II.

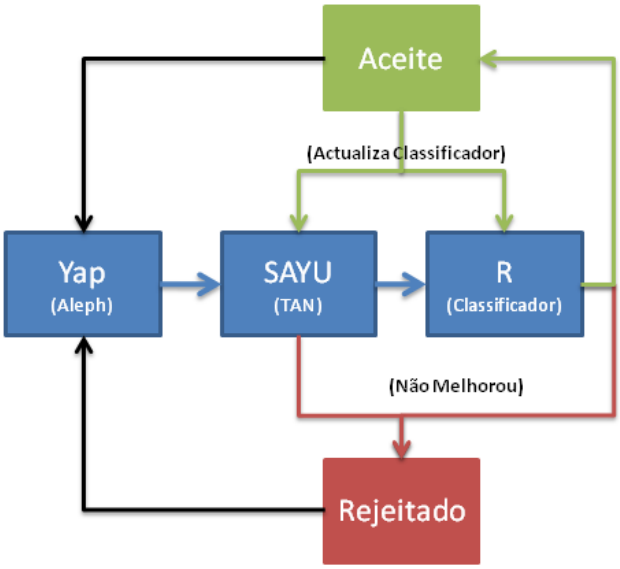



Figura 4.2: Algoritmo II

Analisando a figura podemos ver que cada regra gerada por Aleph através de YAP é enviada para o SAYU. É feita uma avaliação pelo classificador do SAYU. Se o *score* neste classificador melhorar 70% dos *folds* por introdução dessa regra, os dados de treino e teste são enviados para o classificador do R. É feita uma avaliação pelo classificador do R. Se a regra aumentar o *score* em ambos os classificadores essa regra é incorporada nos dois classificadores.

### 4.2.3 Exemplo de utilização de R

Em seguida apresenta-se um exemplo de execução da aplicação implementada, formada por 2300 exemplos e para a qual se utilizou *crossvalidation* de 10 folds. Assim os dados de treino são vectores de tamanho 1840 e os dados de teste vectores de tamanho 230.

No exemplo apenas se descreve a utilização do *R* no SAYU.

Um exemplo dos classificadores usados é o *rpart* (Partição Recursiva) que é executado da seguinte forma:

Por cada regra a ser avaliada, R recebe uma lista de dados de treino (*dados*0) e outra de dados de ajuste (*teste*0); se a regra cobrir o exemplo é dado o valor 1 caso o exemplo não seja coberto pela regra então é dado o valor 0.

R1 é a primeira regra a ser avaliada sendo R0 o valor real do exemplo.

```
> dados0
    R1 R0
1    0  1
```

```
2    1  1
3    1  1
4    0  1
5    1  1
...
1836  0  0
1837  1  0
1838  0  0
1839  0  0
1840  0  0

> teste0
    R1 R0
1    0  1
2    1  1
3    0  1
4    1  1
5    1  1
...
226  0  0
227  0  0
228  1  0
229  0  0
230  0  0
```

Se a regra R1 for incorporada pelos classificadores então são guardados os dados pertencentes à regra R1. Supondo que já se tinham adicionado a regra R1 e uma outra regra R2, considera-se agora o caso onde R3 é a regra a ser avaliada e R0 continua a ser o valor real.

```
> dados0
   R3 R2 R1 R0
1   0  0  1  1
2   1  1  0  1
3   1  1  1  1
4   0  0  1  1
5   1  0  0  1
...
1836  0  0  0  0
1837  1  0  0  0
1838  0  0  0  0
1839  0  0  0  0
1840  0  0  0  0

> teste0
    R3 R2 R1 R0
```

```
1     1  1  0  1
2     0  1  0  1
3     0  0  1  1
4     0  1  1  1
5     1  1  1  1
...
226   0  1  0  0
227   0  0  0  0
228   0  0  0  0
229   0  0  0  0
230   0  1  0  0
```

A função R (*rpart*) constroi a árvore de decisão com os dados *dados*0, em que R0 é a coluna que contém os valores esperados e os dados encontram-se em *dados*0.

```
tree0 <- rpart(R0~., data=dados0)
```

Um exemplo de uma árvore de decisão criada por *rpart* é a que se apresenta de seguida pelo comando *tree*0 onde estão guardados os dados da árvore construída e está representada na Figura 4.3, em que *n* é o número de exemplos pertencente a cada nó (tamanho de cada nó), *split* é o factor de divisão de cada nó, *deviance* corresponde à "deviance" de cada nó e *yval* corresponde ao valor de ajuste representado em cada nó.

```
> tree0
n= 1840

node), split, n, deviance, yval
      * denotes terminal node

1) root 1840 460.00000 0.5000000
  2) R3< 0.5 1491 367.84710 0.4426559
    4) R2< 0.5 1288 313.27640 0.4177019
      8) R1< 0.5 1057 250.73980 0.3869442 *
      9) R1>=0.5 231  56.96104 0.5584416 *
    5) R2>=0.5 203  48.67980 0.6009852 *
  3) R3>=0.5 349  66.30372 0.7449857 *
```

Utilizando esta árvore de decisão as previsões para os dados contidos em *teste*0 são obtidas pela função *predict*:

```
prevs0 <- predict(tree0,teste0)
> prevs0
        1         2         3         4         5         6
0.7485549 0.3893728 0.6184971 0.3893728 0.3893728 0.3893728
....
      225       226       227       228       229       230
0.3893728 0.3893728 0.3893728 0.3893728 0.3893728 0.3893728
```

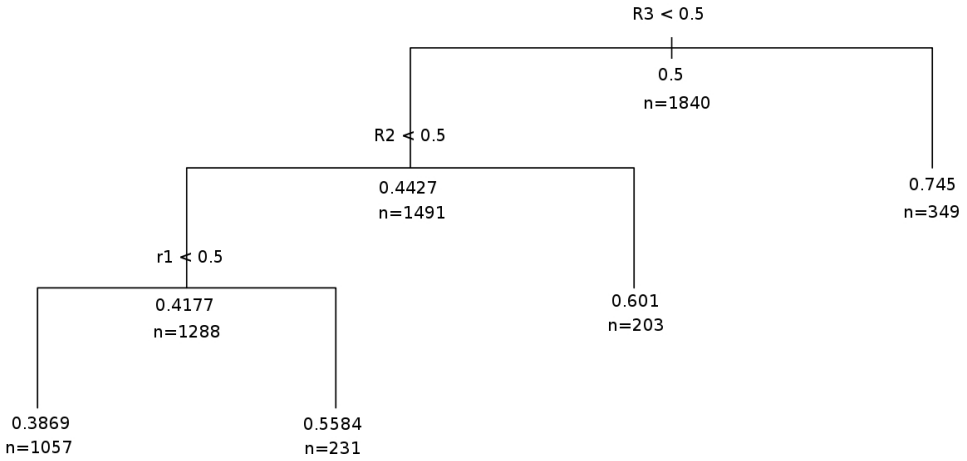



Figura 4.3: Árvore de decisão construída por *rpart*

Os valores obtidos em *prevs*0 são avaliados através da função de avaliação AUC-PR existente em SAYU [DG06]. Esta devolve os valores *new*, que serão comparados com os valores obtidos pelos classificador antes de adicionar a regra, *old*.

```
> new <- c(0.5535184376992154,0.5450405785769136,0.5376308410236963,
0.46730830206671514,0.5494639364135828,0.5294390349089251,
0.5207806726187356,0.49906672158000384,0.5219387427051499)
> old <- c(0.5113580943707518,0.5124974380868861,0.5141178974654727,
0.4585920277086661,0.580732256200783,0.5183701279010726,
0.5143236404272245,0.4630284475193372,0.4510895351179104)
```

**Uso de *R* para efectuar teste estatístico.**

Nesta fase foram considerados dois testes para amostras emparelhadas (Algoritmo I).

Segue-se um exemplo no R da execução do Teste t-Student que é um teste paramétrico apropriado para amostras emparelhadas.

- t-Student

```
> tstat<- t.test(new,old,paired=TRUE)
> tstat

Paired t-test

data:  new and old
t = 2.3437, df = 8, p-value = 0.04714
```

```
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.0003579577 0.0441037762
sample estimates:
mean of the differences
             0.02223087
```

Segue-se um exemplo no R da execução do Teste de Wilcoxon que é um teste não-paramétrico também apropriado para amostras emparelhadas.

- Wilcoxon

```
> tstat<- wilcox.test(new,old,paired=TRUE)
> tstat

Wilcoxon signed rank test

data:  new and old
V = 40, p-value = 0.03906
alternative hypothesis: true location shift is not equal to 0
```

Estes testes devolvem o valor de prova (p-value), que representa a probabilidade de obter um resultado significativamente melhor uma vez que se considerou um teste unilateral. Usualmente uma diferença de médias só é considerada significativamente diferente se o p-value for menor que 5%.

# Capítulo 5

# Validação/Comprovação de Resultados

Avaliaram-se os dois algoritmos propostos em três conjuntos de dados correspondentes a diferentes aplicações de aprendizagem relacional. Uma das aplicações é relativa a *Observational Medical Outcomes Partnership* (OMOP), outra refere-se a um conjunto de citações (Cora) e outra a um conjunto de compostos químicos (Carcinogenesis).

Foram testados diferentes classificadores para os três conjuntos de dados. Neste trabalho comparou-se o SAYU-TAN com a integração de um outro classificador no SAYU. Os classificadores incorporados foram a Partição Recursiva (rpart), o Random Forest (randomForest) e o Support Vector Machine com kernels (ksvm).

Foi usado o seguinte computador na experiência: AMD Athlon(tm) 64 X2 Dual Core Processor 4600+ com 6GB de memória RAM correndo em Linux Ubuntu 9.10 64bits.

Os resultados obtidos foram determinados pré-definindo um limite de tempo de 45 minutos para cada *fold* em todos os classificadores utilizados. Tal limite foi escolhido depois de se verificar que a maioria das regras eram seleccionadas nos primeiros 15 minutos.

## 5.1 OMOP

Este problema baseia-se na previsão de reacções adversas de pacientes a medicamentos [OMO]. Os dados *Observational Medical Outcomes Partnership* contêm pessoas hipotéticas com exposição a drogas e problemas de saúde. Foram utilizados os dados OMOP com um conjunto de 10000 pacientes que incluíam os registos de medicamentos e diagnósticos. O objectivo consistiu em prever o uso de drogas com base nos diagnósticos.

Na implementação dos algoritmos foi utilizado o *cross-validation* dividindo os dados em 10 *folds* [WF99]. Como já foi referido na Secção anterior só foram considerados os valores de *recall* superiores a 0.2 daí só ter sido avaliada 80% da área das curvas *Precision-Recall*.

Tabela 5.1: Resultados obtidos com os dados OMOP - Algoritmo I

| | SAYU-TAN | rpart | randomForest | ksvm |
|---|---|---|---|---|
| Fold 0 | 0,509 | 0,525 | 0,525 | 0,516 |
| Fold 1 | 0,529 | 0,529 | 0,529 | 0,499 |
| Fold 2 | 0,516 | 0,516 | 0,516 | 0,488 |
| Fold 3 | 0,474 | 0,477 | 0,474 | 0,457 |
| Fold 4 | 0,473 | 0,478 | 0,473 | 0,446 |
| Fold 5 | 0,535 | 0,546 | 0,534 | 0,448 |
| Fold 6 | 0,541 | 0,522 | 0,541 | 0,506 |
| Fold 7 | 0,515 | 0,462 | 0,491 | 0,461 |
| Fold 8 | 0,508 | 0,485 | 0,508 | 0,481 |
| Fold 9 | 0,486 | 0,516 | 0,486 | 0,451 |
| **Média** | **0,509** | **0,506** | **0,508** | **0,475** |

A Tabela 5.1 mostra os resultados obtidos para o Algoritmo I com a função de avaliação AUC-PR, *fold* a *fold*.

Como pode ser visto na tabela, há *folds* em que o algoritmo funcionou melhor com a utilização de dois classificadores do que o original SAYU-TAN.

Em seguida, serão discutidos alguns casos em que se obteve melhores resultados. No *fold-0* a melhoria da performance final do classificador ocorre devido a três motivos.

No caso do SAYU-TAN foram aceites 5 regras com um *score* final de 0.509 para AUC-PR.

```
 % clauses in theory:
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,245,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,411,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,140,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,1597,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,2355,_,_,_).
[ Final Test Score 0.509. ]
```

Com o uso do SAYU-TAN com rpart só foram aceites 4 regras.

```
% clauses in theory:
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,245,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,411,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,140,_,_,_).
```

```
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,1597,_,_,_).
[ Final Test Score 0.525. ]
```

A última regra (2355) não passou pelo classificador de Partição Recursiva (rpart) uma vez que o *score* obtido depois de inserida a regra (*MediaR*) ter sido pior que o que era conhecido antes da regra ser analisada (*MediaantR*).

```
MediaR: 0.5001982753074408 MediaantR: 0.5003902432029291
```

No uso do SAYU-TAN com randomForest voltaram a ser aceites apenas 4 regras. Mas desta vez a última regra (2355) foi rejeitada por não passar no teste estatístico (maior que 5%). Compararam-se os resultados antes e depois de regra ter sido incorporada no classificador randomForest.

```
% clauses in theory:
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,245,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,411,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,140,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,1597,_,_,_).
[ Final Test Score 0.525. ]

Test R : 0.059541353373832
```

Com este exemplo conclui-se que a última regra incorporada pelo SAYU-TAN piorou o *score* final, como é mostrado de seguida através de um extracto do log do SAYU.

```
[ add clause 3 (from 246) to Bayes Network (335/176)
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,1597,_,_,_).
 Test Score (at seed 517, 11.840 sec) 0.525376172819702. ]

[ add clause 4 (from 4778) to Bayes Network (245/140)
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,2355,_,_,_).
 Test Score (at seed 304, 124.590 sec) 0.509427007783675. ]
```

Neste caso comprova-se que se pode melhorar o SAYU-TAN com o uso de outro classificador.

No caso do SAYU-TAN com ksvm o resultado final também é melhor que o simples uso do SAYU-TAN mas desta vez não se deve ao facto deste rejeitar uma regra que piore o resultado final, mas sim por ter rejeitado uma regra intermédia (1597) que fez com que adicionasse outras (3474 e 2655) que no fim melhoraram o resultado para este *fold*.

```
% clauses in theory:
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,245,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,411,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,140,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,3474,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,2655,_,_,_).
[ Final Test Score 0.516. ]
```

A quarta regra foi rejeitada pois não passou no teste estatístico relativo aos resultados obtidos pelo ksvm.

```
T.Test R : 0.190216351235369
```

Este algoritmo acabou por incorporar outras duas regras que originaram um melhor *score* final.

```
[ add clause 3 (from 256) to Bayes Network (95/24)
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,3474,_,_,_).
 Test Score (at seed 517, 1150.310 sec) 0.505432951039345. ]

[ add clause 4 (from 281) to Bayes Network (156/76)
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,2655,_,_,_).
 Test Score (at seed 400, 1781.890 sec) 0.515514818336899. ]
```

Procedeu-se à análise de um caso em que o algoritmo funcionou pior que o SAYU-TAN (*fold-7*)

No caso do SAYU-TAN foram aceites 6 regras com score AUC-PR final de 0.515.

```
% clauses in theory:
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,140,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,411,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,436,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,1032,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,1597,_,_,_).
on_drug(A) :-
```

```
   condition_occurrence(_,_,A,_,_,3474,_,_,_).
[ Final Test Score 0.515. ]
% YAP execution halted
```

Já no SAYU-TAN com rpart a regra 436 não foi introduzida pois só melhorou 1 (MelhorR) dos 9 *folds* (inferior aos 30% predefinidos), acabando por introduzir mais uma regra 3188 & 436 , insuficiente para melhorar o *score* em relação ao SAYU-TAN.

```
% clauses in theory:
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,140,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,411,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,3188,_,_,_),
   condition_occurrence(_,_,A,_,_,436,_,_,_).
[ Final Test Score 0.462. ]
% YAP execution halted

MelhorTAN: 9 MelhorR: 1
```

No caso do SAYU-TAN com randomForest a regra 1597 não foi introduzida pois não passou no teste estatístico. O resultado foi superior a 5% acabando por não introduzir nenhuma regra adicional e piorando o *score* em relação ao SAYU-TAN.

```
% clauses in theory:
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,140,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,411,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,436,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,1032,_,_,_).
[ Final Test Score 0.491. ]
% YAP execution halted

T.Test R : 0.0621426175882781
```

No caso do SAYU-TAN com ksvm a regra 411 não foi introduzida por não passar no teste estatístico, acabando por serem introduzidas outras 2 regras (2655 e 2068) com as quais se obteve pior *score*.

```
% clauses in theory:
on_drug(A) :-
```

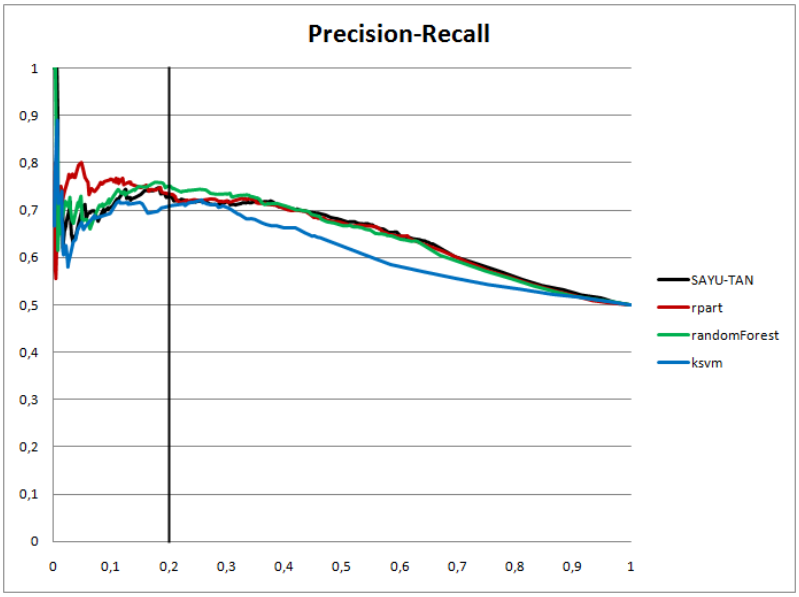



Figura 5.1: Curvas *Precision-Recall* para os dados OMOP - Algoritmo I

```
   condition_occurrence(_,_,A,_,_,140,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,2655,_,_,_).
on_drug(A) :-
   condition_occurrence(_,_,A,_,_,2068,_,_,_).
[ Final Test Score 0.461. ]
% YAP execution halted

T.Test R : 0.159107978389919
```

Na Figura 5.1 mostra-se o gráfico das curvas *Precision-Recall* para os 4 classificadores que foram testados.

Não existem diferenças significativas entre o SAYU-TAN e SAYU-TAN-rpart e também SAYU-TAN-randomForest, como pode ser comprovado na Tabela 5.3. Quanto ao SAYU-TAN-ksvm verificou-se uma diferença significativa, estando os resultados abaixo do esperado. Um possível motivo poderá ter sido o tempo de uso do CPU na aplicação do classificador do R ser muito dispendioso, daí não serem analisadas tantas regras como nos outros casos. Alternativamente pode não ter sido utilizado o classificador *svm* da melhor forma.

Na Tabela 5.2 são mostrados os resultados obtidos pelo Algoritmo II com a função

Tabela 5.2: Resultados obtidos com os dados OMOP - Algoritmo II

| | SAYU-TAN | rpart | randomForest | ksvm |
|---|---|---|---|---|
| Fold 0 | 0,509 | 0,533 | 0,524 | 0,528 |
| Fold 1 | 0,529 | 0,530 | 0,536 | 0,498 |
| Fold 2 | 0,516 | 0,516 | 0,548 | 0,496 |
| Fold 3 | 0,474 | 0,486 | 0,498 | 0,464 |
| Fold 4 | 0,473 | 0,476 | 0,507 | 0,490 |
| Fold 5 | 0,535 | 0,535 | 0,521 | 0,549 |
| Fold 6 | 0,541 | 0,533 | 0,537 | 0,519 |
| Fold 7 | 0,515 | 0,496 | 0,468 | 0,462 |
| Fold 8 | 0,508 | 0,493 | 0,488 | 0,464 |
| Fold 9 | 0,486 | 0,516 | 0,565 | 0,494 |
| **Média** | **0,509** | **0,511** | **0,519** | **0,496** |

de avaliação AUC-PR, *fold* a *fold*. Com base nesta tabela pode verificar-se que o *score* médio dos dez folds aumentou com a integração de *rpart* e *randomForest*. O facto de não ter aumentado com a integração de *ksvm* pode ser explicado pelos motivos já referidos no Algoritmo I.

Na Figura 5.2 apresentam-se as curvas *Precision-Recall* para os 4 classificadores que foram testados.

Através das curvas *Precision-Recall* pode verificar-se que houve uma melhoria com *randomForest*, especialmente para valores elevados de *Recall*.

Com este Algoritmo não existem diferenças significativas entre o SAYU -TAN e qualquer um dos três classificadores que foram integrados no SAYU, como pode ser comprovado na Tabela 5.3.

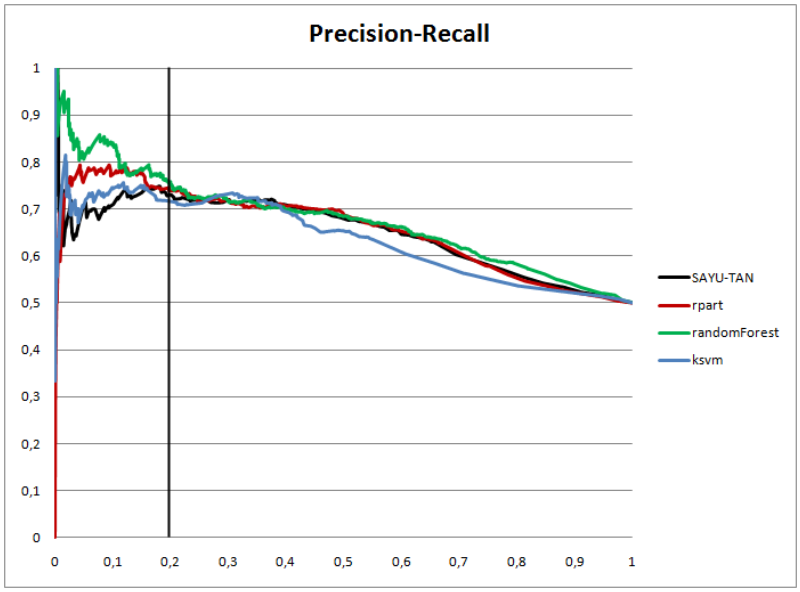



Figura 5.2: Curvas *Precision-Recall* para os dados OMOP - Algoritmo II

Tabela 5.3: Comparação do SAYU-TAN com os outros classificadores para OMOP

| | p-value (*vs* rpart) | p-value (*vs* randomForest) | p-value(*vs* ksvm) |
|---|---|---|---|
| Algoritmo I | 0.694 | 0.773 | 0.0019 |
| Algoritmo II | 0.5851 | 0.3620 | 0.1729 |

## 5.2 Cora

O objectivo deste conjunto de dados é prever se duas citações se referem ao mesmo artigo científico. O conjunto de dados foi inicialmente construído por McCallum et al. [MNRS00]. Neste trabalho utilizou-se a versão dos dados utilizada por Kok e Domingos [KD05]. Core inclui 1295 citações de 112 artigos científicos de Ciências dos Computadores. O *background* inclui data do título, proveniência, autor e ano de cada citação. Artigo científico, título, proveniência, autor e ano forem definidos como sendo as palavras que deveriam encabeçar as cláusulas. Associou-se cada artigo científico ao seu título, proveniência, autor e ano e agregaram-se os dados por artigo científico e autor.

Na implementação dos algoritmos foi utilizado o *cross-validation* dividindo os dados em 5 *folds* já definido por Kok e Domingos [KD05]. De notar que o resultado obtido refere-se a *recall* superior a 0.2 daí só ser avaliada 80% da área das curvas *Precision-Recall*.

Na Tabela 5.4 são apresentados os resultados obtidos pelo Algoritmo I com a função de avaliação AUC-PR, *fold* a *fold*.

Apresenta-se, de seguida, a Figura 5.3 das curvas *Precision-Recall* para os 4 classificadores que foram testados.

Tabela 5.4: Resultados obtidos com os dados Cora - Algoritmo I

| | SAYU-TAN | rpart | randomForest | ksvm |
|---|---|---|---|---|
| Fold 0 | 0,757 | 0,793 | 0,757 | 0,763 |
| Fold 1 | 0,766 | 0,761 | 0,761 | 0,761 |
| Fold 2 | 0,792 | 0,706 | 0,790 | 0,790 |
| Fold 3 | 0,791 | 0,778 | 0,770 | 0,770 |
| Fold 4 | 0,711 | 0,563 | 0,563 | 0,470 |
| **Média** | **0,763** | **0,720** | **0,728** | **0,711** |

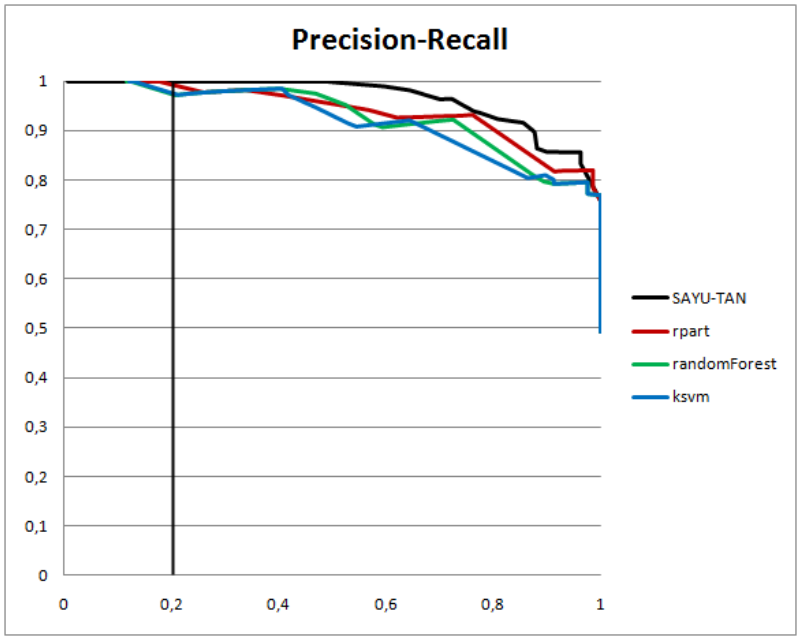



Figura 5.3: Curvas *Precision-Recall* para os dados Cora - Algoritmo I

Apesar de, em média, o SAYU-TAN ter sido superior, em certos *folds* este obteve piores resultados que os outros classificadores. No entanto, não se verificaram diferenças significativas entre SAYU-TAN e qualquer um dos classificadores integrados no SAYU, o que pode ser comprovado na Tabela 5.6.

Na Tabela 5.4 são mostrados os resultados obtidos pelo Algoritmo II com a função de avaliação AUC-PR, *fold* a *fold*.

Na Figura 5.4 representam-se as curvas *Precision-Recall* para os 4 classificadores que foram testados.

Neste algoritmo as médias do *score* dos classificadores testados foram melhores que o SAYU-TAN não sendo, no entanto, significativamente melhores como está apresentado na Tabela 5.6. Analisando *fold* a *fold* a Tabela 5.6 verificou-se que, para estes dados, a

Tabela 5.5: Resultados obtidos com os dados Cora - Algoritmo II

| | SAYU-TAN | rpart | randomForest | ksvm |
|---|---|---|---|---|
| Fold 0 | 0,757 | 0,763 | 0,778 | 0,747 |
| Fold 1 | 0,766 | 0,778 | 0,778 | 0,780 |
| Fold 2 | 0,792 | 0,790 | 0,797 | 0,783 |
| Fold 3 | 0,791 | 0,793 | 0,793 | 0,799 |
| Fold 4 | 0,711 | 0,735 | 0,741 | 0,753 |
| **Média** | **0,763** | **0,772** | **0,777** | **0,772** |

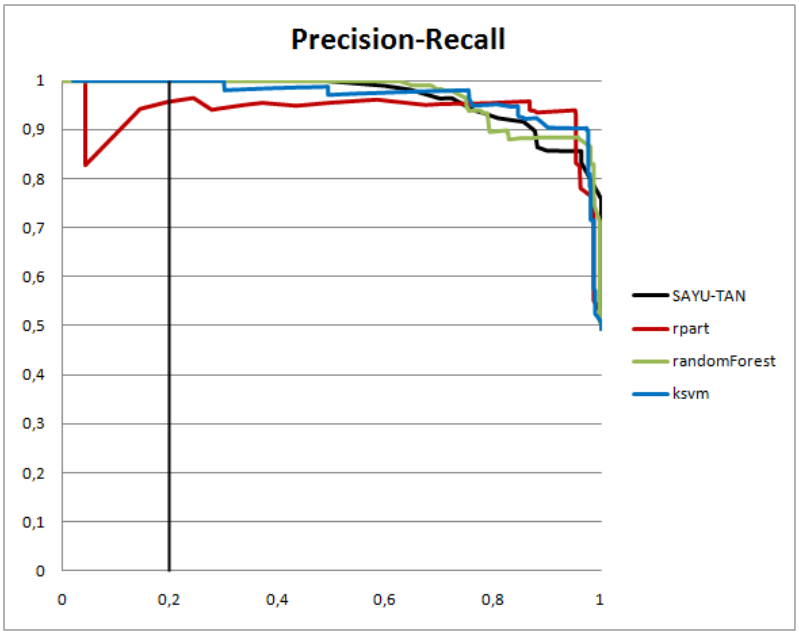



Figura 5.4: Curvas *Precision-Recall* para os dados Cora - Algoritmo II

integração de *ksvm* melhorou muito o *score* tendo até no *fold* 3 atingido um valor muito próximo do máximo (80%).

Tabela 5.6: Comparação do SAYU-TAN com os outros classificadores para Cora

| | p-value (*vs* rpart) | p-value (*vs* randomForest) | p-value(*vs* ksvm) |
|---|---|---|---|
| Algoritmo I | 0.2577 | 0.2835 | 0.3285 |
| Algoritmo II | 0.1376 | 0.05357 | 0.3965 |

Da análise das curvas *Precision-Recall* constata-se que os classificadores integrados no SAYU melhoram o *score*, em especial para valores elevados de Recall.

## 5.3 Carcinogenesis

Este conjunto de dados é bem conhecido pela Comunidade Científica pois foi fornecido por 'National Toxicology Program' para predizer a potencialidade em provocar cancro (carcinogenesis) de 30 produtos químicos previamente seleccionados [JHF].

Os dados que foram utilizados contêm 182 testes positivos e 148 testes negativos [DBdCD+05b]. Estes foram divididos aleatoriamente em 10 *folds* contendo, cada um, aproximadamente o mesmo número de exemplos positivos e negativos.

Na implementação dos Algoritmos foi utilizado o *cross-validation* dividindo os dados em 10 *folds* [WF99].

Tabela 5.7: Resultados obtidos com os dados Carcinogenesis - Algoritmo I

| | SAYU-TAN | rpart | randomForest | ksvm |
|---|---|---|---|---|
| Fold 0 | 0,468 | 0,518 | 0,490 | 0,503 |
| Fold 1 | 0,616 | 0,563 | 0,616 | 0,554 |
| Fold 2 | 0,467 | 0,472 | 0,467 | 0,497 |
| Fold 3 | 0,555 | 0,419 | 0,435 | 0,412 |
| Fold 4 | 0,510 | 0,507 | 0,510 | 0,507 |
| Fold 5 | 0,591 | 0,507 | 0,501 | 0,539 |
| Fold 6 | 0,557 | 0,563 | 0,573 | 0,563 |
| Fold 7 | 0,497 | 0,463 | 0,457 | 0,499 |
| Fold 8 | 0,548 | 0,529 | 0,548 | 0,545 |
| Fold 9 | 0,693 | 0,688 | 0,671 | 0,688 |
| **Média** | **0,550** | **0,523** | **0,527** | **0,531** |

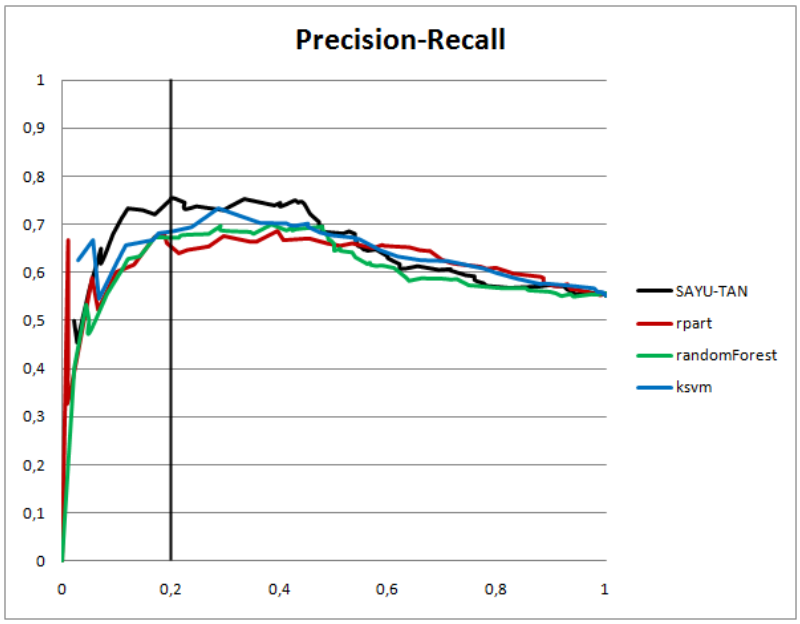



Figura 5.5: Curvas *Precision-Recall* para os dados Carcinogenesis - Algoritmo I

Na Tabela 5.7 são apresentados os resultados obtidos pelo Algoritmo I, com a função de avaliação AUC-PR, *fold* a *fold*.

Analisando a Tabela 5.7 constatou-se que há uma grande variação entre os *scores*, de *fold* para *fold*. Assim, por exemplo, nos *folds* 3 e 5 o *score* baixou em todos os classificadores utilizados quando comparado com o do SAYU-TAN. Já nos *folds* 0 e 6, por exemplo, o *score* aumentou em todos os classificadores.

Na Figura 5.5 representam-se as curvas *Precision-Recall* para os 4 classificadores que foram testados.

O gráfico revela que o algoritmo proposto, quando utilizado com *rpart* ou *ksvm*, é mais eficiente que SAYU-TAN, para valores de *Recall* alto.

Tabela 5.8: Resultados obtidos com os dados Carcinogenesis - Algoritmo II

| | SAYU-TAN | rpart | randomForest | ksvm |
|---|---|---|---|---|
| Fold 0 | 0,468 | 0,629 | 0,633 | 0,581 |
| Fold 1 | 0,616 | 0,527 | 0,673 | 0,543 |
| Fold 2 | 0,467 | 0,482 | 0,513 | 0,497 |
| Fold 3 | 0,555 | 0,400 | 0,533 | 0,424 |
| Fold 4 | 0,510 | 0,511 | 0,553 | 0,465 |
| Fold 5 | 0,591 | 0,523 | 0,626 | 0,534 |
| Fold 6 | 0,557 | 0,572 | 0,544 | 0,531 |
| Fold 7 | 0,497 | 0,498 | 0,538 | 0,475 |
| Fold 8 | 0,548 | 0,519 | 0,519 | 0,494 |
| Fold 9 | 0,693 | 0,686 | 0,652 | 0,628 |
| **Média** | **0,550** | **0,535** | **0,578** | **0,517** |

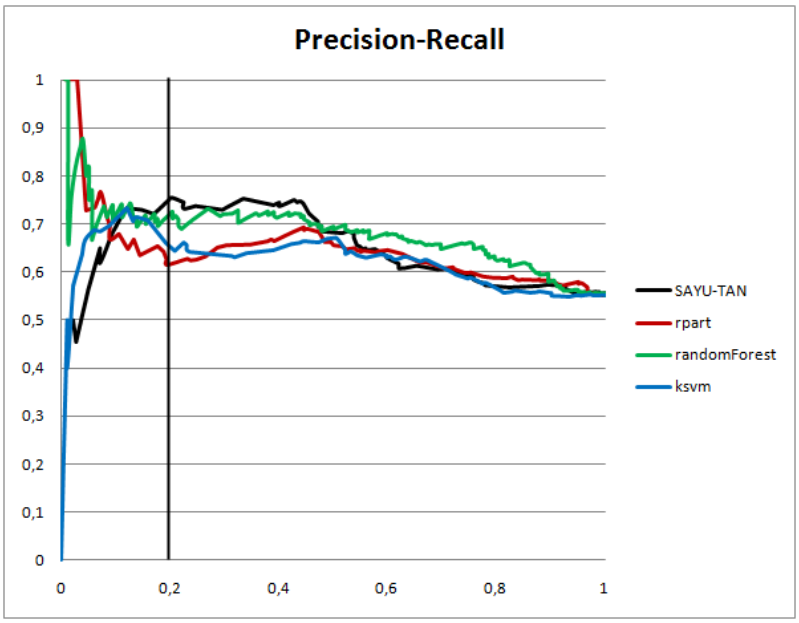



Figura 5.6: Curvas *Precision-Recall* para os dados Carcinogenesis - Algoritmo II

As diferenças obtidas em comparação com SAYU-TAN não são, no entanto, significativas com pode ser constatado pela Tabela 5.9.

Na Tabela 5.8 são mostrados os resultados obtidos pelo Algoritmo II, também com a função de avaliação AUC-PR, *fold* a *fold*.

Na Figura 5.6 representam-se as curvas de *Precision-Recall* para os 4 classificadores que foram testados.

Com este Algoritmo apenas *RandomForest* melhorou muito o *score* para valores altos de *Recall*.

Comparativamente com SAYU-TAN as diferenças continuam a não ser significativas quando considerados todos os $folds$, como prova a Tabela 5.9.

Tabela 5.9: Comparação do SAYU-TAN com os outros classificadores para Carcinogenesis

| | p-value (*vs* rpart) | p-value (*vs* randomForest) | p-value(*vs* ksvm) |
|---|---|---|---|
| Algoritmo I | 0.1360 | 0.1495 | 0.2756 |
| Algoritmo II | 0.5676 | 0.1724 | 0.1466 |

A análise do número de cláusulas consideradas em cada algoritmo, para os três conjuntos de dados, foi também efectuada tendo sido calculada a média de cláusulas por $fold$ que se apresenta na Tabela 5.10.

Tabela 5.10: Resultados Cláusulas

| | TAN | rpart | randomForest | TAN-ksvm |
|---|---|---|---|---|
| Algoritmo I | | | | |
| OMOP | 4,6 | 4 | 4,2 | 3,3 |
| Cora | 4 | 2,6 | 3,4 | 3,2 |
| Carcinogenesis | 3,4 | 2,7 | 3,3 | 2 |
| Algoritmo II | | | | |
| OMOP | 4,6 | 4,8 | 19 | 7 |
| Cora | 4 | 5,8 | 11 | 9,2 |
| Carcinogenesis | 3,4 | 4,2 | 9,6 | 3,5 |

Da análise da Tabela 5.10 conclui-se que ao aumentar a exigência dos classificadores o número de cláusulas diminui (Algoritmo I). Já ao diminuir o rigor dos classificadores (Algoritmo II), o número de cláusulas aumenta significativamente.

# Capítulo 6

# Conclusões e Trabalho Futuro

## 6.1 Satisfação dos Objectivos

Melhorar o SAYU seria uma tarefa difícil uma vez que o nível de eficácia deste classificador é já bastante elevado. No entanto, com a integração de um segundo classificador no SAYU conseguiu-se melhorar a performance do classificador em alguns casos. Para tal, desenvolveram-se dois algoritmos: o Algoritmo I onde o SAYU e o segundo classificador são muito rigorosos e o Algoritmo II onde esses classificadores assumem um menor rigor.

Com base nos resultados obtidos pode concluir-se que nenhum dos classificadores foi significativamente mais eficaz para os três tipos de dados testados.

Com o Algoritmo I os valores do AUC-PR obtidos pelos classificadores foram melhores em alguns folds. Este Algoritmo apresentou resultados bastante satisfatórios com os dados Carcinogenesis, para altos valores de Recall, com *Rpart* e *Ksvm*. Dos três classificadores utilizados só a utilização de *Ksvm* nos dados OMOP apresentou diferença significativa.

Com o Algoritmo II houve classificadores que tanto na média do *score* como nas curvas *Precision-Recall* obtiveram interessantes melhorias. No caso dos dados "Cora"os três classificadores utilizados obtiveram melhor performance que o SAYU-TAN.

Desta forma se conclui que o classificador a integrar no SAYU é determinante para a sua eficiência.

## 6.2 Trabalho Futuro

Os resultados mais interessantes foram obtidos com o Algoritmo II. Assim, este deverá ser o Algoritmo a aperfeiçoar no futuro. Deverá aperfeiçoar-se o classificador *Ksvm* uma

vez que foi com este classificador que os Algoritmos funcionaram menos bem.

Será de considerar também a hipótese de junção de mais classificadores simples pois, como foi constatado, a eficiência do algoritmo depende do classificador que se integra no SAYU.

A análise efectuada *fold* a *fold* sugere que o uso de *bootstrapping* será também uma hipótese a implementar futuramente.

# Referências


[Alc02] Josep Roure Alcobé. An incremental algorithm for tree-shaped bayesian network learning. In *ECAI*, pages 350–354, 2002.

[BACW] Leo Breiman, R port by Andy Liaw Adele Cutler e Matthew Wiener. Random forest in r. http://cran.r-project.org/web/packages/randomForest/.

[Bay73] Thomas Bayes, 1973. *UPhilosophical Transactions of the Royal Society of London*, disponível em http://www.stat.ucla.edu/history/essay.pdf.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[CL] CRACS e LIACC. Yap-prolog.

[DBdCD+05a] Jesse Davis, Elizabeth S. Burnside, Inês de Castro Dutra, David Page e Vítor Santos Costa. An integrated approach to learning bayesian networks of rules. In *ECML*, pages 84–95, 2005.

[DBdCD+05b] Jesse Davis, Elizabeth S. Burnside, Inês de Castro Dutra, David Page, Raghu Ramakrishnan, Vítor Santos Costa e Jude W. Shavlik. View learning for statistical relational learning: With an application to mammography. In *IJCAI*, pages 477–498, 2005.

[DCRP] Jesse Davis, Vitor Santos Costa, Soumya Ray e David Page. Tightly integrating relational learning and multiple-instance regression for real-valued drug activity prediction.

[DD04] Robert Kirk DeLisle e Steven L. Dixon. Induction of decision trees via evolutionary programming. *Journal of Chemical Information and Modeling*, 44(3):862–870, 2004.

[DdCDOC04] Jesse Davis, Inês de Castro Dutra, Irene M. Ong e Vítor Santos Costa. Using bayesian classifiers to combine rules. In *In 3rd Workshop on Multi-Relational Data Mining*, 2004.

[DG06] Jesse Davis e Mark Goadrich. The relationship between precision-recall and roc curves. In *ICML*, pages 233–240, 2006.

[DP97] Pedro Domingos e Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[FG96] Nir Friedman e Moisés Goldszmidt. Building classifiers using bayesian networks. In *AAAI/IAAI, Vol. 2*, pages 1277–1284, 1996.

[Fit04] Melvin Fitting. First-order intensional logic. *Ann. Pure Appl. Logic*, 127(1-3):171–193, 2004.

[Gam04] J Gama. Árvores de decisão, 2004. Machine Learning.

[Har96] M E Harmon. Reinforcement learning: Tutorial, 1996. http://www.anw.cs.umass.edu/~mharmon/rltutorial/frames.html.

[JHF] E Weisburger J Huff e V A Fung. Multicomponent criteria for predicting carcinogenicity: dataset of 30 ntp chemicals. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1469676/.

[JRI] JRI. Jri - java/r interface. http://rosuda.org/JRI/.

[Kar] Alexandros Karatzoglou. ksvm in r. http://rss.acs.unt.edu/Rdoc/library/kernlab/html/ksvm.html.

[KD05] Stanley Kok e Pedro Domingos. Learning the structure of markov logic networks. In *ICML*, pages 441–448, 2005.

[KP02] Eamonn J. Keogh e Michael J. Pazzani. Learning the structure of augmented bayesian classifiers. *International Journal on Artificial Intelligence Tools*, 11(4):587–601, 2002.

[LC03] A C Lorena e A C P L F Carvalho. Introdução às máquinas de vetores suporte (support vector machines), 2003. Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.

[LD96] Nada Lavrac e Saso Dzeroski. A reply to pazzani's book review of "inductive logic programming: Techniques and applications". *Machine Learning*, 23(1):109–111, 1996.

[LD01] N Lavrac e S Dzeroski. Relational data mining. pages 48–73, 2001. Springer, Berlin.

[Lin93] L Lin. Reinforcement learning for robots using neural networks, 1993.

[LKR05] Niels Landwehr, Kristian Kersting e Luc De Raedt. nfoil: Integrating naïve bayes and foil. In *AAAI*, pages 795–800, 2005.

[LPRF06] Niels Landwehr, Andrea Passerini, Luc De Raedt e Paolo Frasconi. kfoil: Learning simple relational kernels. In *AAAI*, 2006.

[Mit97] Tom Mitchell. Machine learning, 1997. McGraw Hill.

[MK97] G J McLachlan e T Krishnan. The em algorithm and extensions, 1997.

[MNRS00] Andrew McCallum, Kamal Nigam, Jason Rennie e Kristie Seymore. Automating the construction of internet portals with machine learning. *Inf. Retr.*, 3(2):127–163, 2000.

[MSL00] Christopher D. Manning, Hinrich Schiitze e Lillian Lee. Review: Foundations of statistical natural language processing, christopher d. manning and hinrich schütze, 2000.

[Mug95] Stephen Muggleton. Inverse entailment and progol. *New Generation Comput.*, 13(3&4):245–286, 1995.

[OMO] OMOP. http://omop.fnih.org/.

[Ono01] M Onoda. Estudo sobre um algoritmo de árvore de decisão acoplado a um sistema de banco de dados relacional, 2001.

[PFK98] Foster J. Provost, Tom Fawcett e Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *ICML*, pages 445–453, 1998.

[R] R. http://www.r-project.org/.

[RBKK95] Stuart J. Russell, John Binder, Daphne Koller e Keiji Kanazawa. Local learning in probabilistic networks with hidden variables. In *IJCAI*, pages 1146–1152, 1995.

[RD06] Matthew Richardson e Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[RN95] Stuart J. Russell e Peter Norvig. A modern, agent-oriented approach to introductory artificial intelligence. *SIGART Bulletin*, 6(2):24–26, 1995.

[SC93] Ashwin Srinivasan e Rui Camacho. P-progol user manual, 1993.

[Sil05] Luiza Maria Oliveira Silva. Uma aplicação de Árvores de decisção, redes neurais e knn para a identificação de modelos arma não-sazonais e sazonais, 2005. Tese de Doutorado.

[Sri93] Ashwin Srinivasan. The aleph manual, 1993.

[SS02] Bernhard Schölkopf e Alex J. Smola. A short introduction to learning with kernels. In *Machine Learning Summer School*, pages 41–64, 2002.

[TA] Terry M Therneau e Beth Atkinson. rpart in r. http://cran.r-project.org/web/packages/rpart/.

[TK08] Sergios Theodoridis e Konstantinos Koutroumbas. Pattern recognition. *IEEE Transactions on Neural Networks*, 19(2):376–376, 2008.

[WEK] WEKA. http://www.cs.waikato.ac.nz/~ml/index.html.

[WF99] Ian H. Witten e Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.