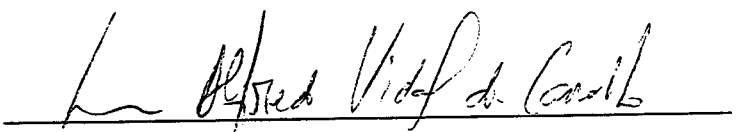
UMA NOVA METODOLOGIA PARA MELHORAMENTO DE CLASSIFICADORES BASEADOS EM ILP

Rogerio Lopes Salvini

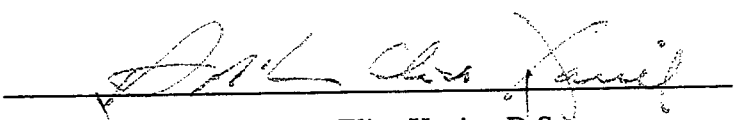
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

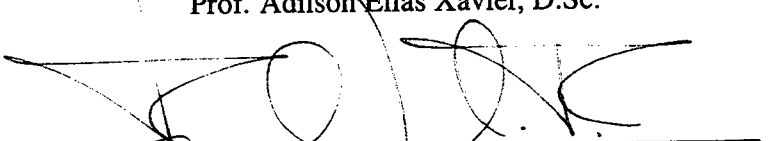


Prof. Luis Alfredo Vidal de Carvalho, D.Sc.

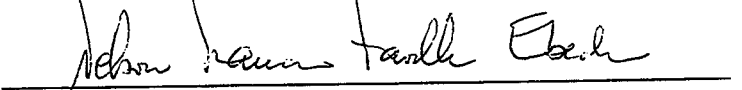
Prof. Valmir Carneiro Barbosa, Ph.D.



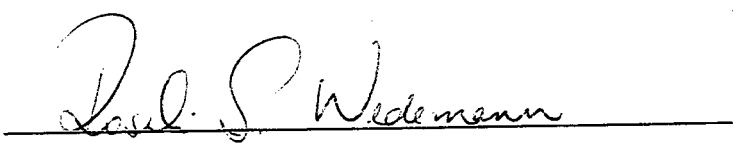
Prof. Adilson Elias Xavier, D.Sc.



Prof. Felipe Maia Galvão França, Ph.D.



Prof. Nelson Francisco Favilla Ebecken, D.Sc.



Prof. Roseli Suzi Wedemann, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2008

SALVINI, ROGERIO LOPES

Uma nova metodologia para melhoramento de classificadores baseados em ILP [Rio de Janeiro] 2008

XI, 74 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2008)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Programação Lógica Indutiva
2. *Ensembles*
3. Máquinas de Vetor Suporte
4. Classificadores baseados em cláusulas

I. COPPE/UFRJ II. Título (série)

Ao meu guru.

Agradecimentos

Primeiramente, agradeço aos meus pais que, com grande esforço, permitiram que eu atingisse o nível mais alto de formação acadêmica.

Várias pessoas estiveram presentes em algum momento desta minha jornada. Gostaria de agradecer principalmente às pessoas que me deram suporte (e me suportaram) até o final.

À minha orientadora Inês de Castro Dutra que me aceitou como seu orientando e que, apesar de algumas dificuldades impostas pela distância física, não deixou de participar ativamente e dar a sua valiosa contribuição no trabalho. Agradeço também a ela pela sua amizade e compreensão.

Aos professores Luis Alfredo Vidal de Carvalho e Adilson Elias Xavier, que desde o mestrado depositaram sua confiança em mim. A amizade e a ajuda destes professores em momentos críticos foi inestimável.

Devo agradecer sobretudo ao meu amigo Eduardo Aguilar que, além de bom amigo foi também orientador deste trabalho. Aprendi (e continuo aprendendo) muito com ele.

E, finalmente, a todos os meus colegas do PESC, que estiveram junto, deram força e incentivo nos momentos difíceis desta caminhada.

Gostaria também de agradecer à CAPES pela bolsa de estudos concedida, o que permitiu que eu me dedicasse aos meus estudos e à realização desta tese.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UMA NOVA METODOLOGIA PARA MELHORAMENTO DE CLASSIFICADORES BASEADOS EM ILP

Rogerio Lopes Salvini

Setembro/2008

Orientador: Inês de Castro Dutra

Programa: Engenharia de Sistemas e Computação

Sistemas em Programação Lógica Indutiva (ILP) têm sido usados com sucesso na extração de modelos relacionais de dados. A maioria dos sistemas ILP usam um algoritmo guloso de cobertura para obter o melhor conjunto de cláusulas que descreve os exemplos. Este conjunto de cláusulas é chamado de teoria. A busca pela melhor teoria, usando o algoritmo guloso, consome muito tempo e frequentemente produz classificadores demasiadamente complexos. Uma abordagem alternativa para obter um classificador ILP é aprender uma cláusula por vez e usar métodos de *ensemble* para combinar estas cláusulas. A vantagem desta abordagem é que é muito mais rápido obter uma cláusula no sistema ILP do que obter uma teoria inteira. Além disso, este método obtém classificadores menos complexos. Neste trabalho, apresentamos três diferentes abordagens para combinar cláusulas. Na primeira abordagem, usamos o método de *bagging* para gerar um *ensemble* de cláusulas e comparamos o classificador obtido com uma teoria e um *ensemble* de teorias. Na segunda abordagem, introduzimos uma nova metodologia, denominada *Nata*, que seleciona as melhores cláusulas baseadas em suas coberturas. Após selecionar as cláusulas, o classificador é apresentado como a disjunção destas cláusulas. Na terceira abordagem, combinamos as classificações dadas pelas cláusulas obtidas pelo sistema ILP usando Máquinas de Vetor Suporte gerando um classificador híbrido. Testamos estas abordagens em problemas clássicos em ILP e em uma base de dados real obtida do *National Mammography Database* (NMD) americano. Nossos resultados mostram que estas abordagens são mais rápidas e produzem classificadores melhores e menos complexos do que os métodos tradicionais.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A NEW METODOLOGY TO IMPROVE CLASSIFIERS BASED ON ILP

Rogerio Lopes Salvini

September/2008

Advisor: Inês de Castro Dutra

Department: Systems Engineering and Computer Science

Inductive Logic Programming systems (ILP) have been successfully used to extract relevant information from relational data. Most ILP systems use a greedy cover algorithm to obtain the best set of clauses that describe the examples. This set of clauses is called a theory. Generally, the search for the best theory, using the greedy algorithm, is very time consuming and can produce overly complex classifiers. An alternative approach to obtain an ILP classifier is to learn one clause at a time and use ensemble methods to combine these clauses. The advantage of this approach is that it is much faster (severals orders of magnitude) to obtain one clause in the ILP system than to obtain a whole theory. And, as we show, this method obtains less complex classifiers. In this work, we present three different approaches to combine the clauses. In the first one, we use *bagging* to generate an ensemble of clauses and compare the obtained classifer with a theory and an ensemble of theories. In the second approach, we introduce a new methodology, called *Nata*, that selects the best clauses based on their coverage. After selecting the clauses, the classifier is presented as the disjunction of these clauses. In the third approach, we combine the clauses obtained by the ILP system using Support Vector Machines generating a hybrid classifier. We tested these approaches on classic ILP problems and on a real dataset obtained from the *National Mammography Database* (NMD). Our results show that these approaches are faster than traditional ones and can produce better and less complex classifiers.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Aprendizado de Máquina (AM) tem sido uma fascinante e próspera área da Inteligência Artificial nos últimos anos. Técnicas de AM estão sendo aplicadas em problemas de diversos domínios como Engenharia, Medicina, Biologia, Lingüística, Segurança etc.

Redes Neuronais Artificiais e Árvores de Decisão são alguns exemplos clássicos e populares de algoritmos de AM. Mais recentemente métodos Bayesianos e Máquinas de Vetor Suporte têm recebido bastante atenção da comunidade científica.

Por outro lado, sistemas em Programação Lógica Indutiva (*Inductive Logic Programming* (ILP)) têm sido usados com sucesso na extração de modelos relacionais de dados. A tarefa de um sistema ILP é produzir um classificador representado em Lógica de Primeira Ordem, dados exemplos positivos, exemplos negativos, a descrição destes exemplos (também chamada de conhecimento prévio ou *background knowledge*) e um conjunto de restrições que define a forma como um classificador deve ser construído.

Ao contrário dos classificadores proposicionais ou numéricos, e estatísticos, sistemas ILP produzem classificadores que possuem um grande poder de expressão, pois são baseados em Lógica de Primeira Ordem. Isto faz com que estes classificadores possam ser facilmente entendidos por especialistas em diversos domínios.

A maioria dos sistemas ILP usa um algoritmo que iterativamente examina cláusulas candidatas (espaço de busca) a fim de encontrar cláusulas "boas", que modelem os dados da melhor forma possível de acordo com critérios pré-determinados (tipo de função de avaliação das cláusulas, tamanho máximo da cláusula, quantidade de memória disponível e tempo de computação, entre outros), e formar uma boa teoria (ou conjunto de cláusulas).

Idealmente, a busca irá parar quando as regras (cláusulas) cobrirem todos os exemplos positivos e nenhum exemplo negativo. Porém, ao lidar com problemas do mundo real onde muitas vezes os dados possuem algum tipo de ruído, informação errada ou faltando, normalmente estes requisitos são relaxados tal que as regras cubram a maior parte dos exemplos positivos e apenas alguns poucos exemplos negativos.

Isto acaba se tornando um problema maior quando o domínio em questão envolver dados altamente desbalanceados, por exemplo, quando possuir muito mais exemplos po-

sitivos do que negativos, o que tende a aumentar o número de exemplos falso-positivos cobertos.

Além disso, o espaço de busca de um sistema ILP cresce conforme o aumento da base de dados, o que pode tornar inviável a busca pela melhor teoria.

Uma alternativa é a utilização de *ensembles*, onde o sistema gera classificadores mais fracos, explorando uma parte menor do espaço de busca. Estes classificadores, quando combinados, podem produzir um classificador mais forte.

Neste contexto, os objetivos do nosso trabalho são:

- reduzir o tempo de obtenção de um classificador baseado em ILP,
- ser capaz de obter bons classificadores ILP para bases de dados que têm um espaço de busca muito grande.

Para atingir estes objetivos, trabalhamos diretamente com a geração de cláusulas. Aprender cláusulas de forma independente é mais simples e mais rápido de se obter do que teorias, onde as cláusulas são aprendidas condicionadas às que foram geradas anteriormente.

Neste trabalho apresentamos três abordagens para gerar, selecionar e combinar cláusulas:

1. Na primeira abordagem usamos o método de *bagging* para gerar *ensembles* de cláusulas o qual comparamos com ensembles de teorias.
2. Na segunda abordagem introduzimos uma nova metodologia para selecionar cláusulas baseada em suas coberturas e construímos um classificador, denominado Nata, fazendo a disjunção destas cláusulas.
3. Na terceira abordagem, combinamos as classificações dadas pelas cláusulas para os exemplos através das Máquinas de Vetor Suporte (*Support Vector Machine* (SVM)), metodologia esta que chamamos de híbrida.

Estas abordagens mostraram ser mais rápidas e produziram classificadores melhores do que as teorias produzidas com os algoritmos de cobertura gulosos.

Assim, nossas contribuições são as seguintes:

1. Provamos que *ensemble* de cláusulas consegue gerar classificadores tão fortes como uma teoria e *ensemble* de teorias em um espaço de tempo mais curto.
2. Com base no *ensemble* de cláusulas temos mais duas contribuições:

    2.1 Fornecer ao especialista um classificador mínimo que descreve todos os dados relevantes do domínio.

2.2 A combinação de *ensemble* de cláusulas com SVM produz resultados ainda melhores dos que já existem na literatura e do que o próprio *ensemble* de cláusulas.

Aplicamos nossa metodologia em cinco bases de dados clássicas em ILP onde obtivemos resultados competitivos com os da literatura.

Além disso, conseguimos melhores resultados numa base de dados real de mamografias, extraída do "*National Mammography Database*" (NMD) americano.

Este texto está organizado da seguinte forma: no capítulo seguinte serão apresentados alguns fundamentos teóricos, sublinhando aqueles que formaram a base deste trabalho. No Capítulo 3 apresentamos a nossa primeira abordagem com um estudo empírico que compara *ensembles* compostos por teorias com *ensembles* compostos por simples cláusulas. Mostramos que ensembles de cláusulas podem ser tão eficientes quanto os de teorias com a vantagem de serem classificadores mais simples e gerados em muito menos tempo. No Capítulo 4 apresentamos a segunda abordagem baseada na seleção do menor conjunto de cláusulas que maximiza a cobertura de exemplos. Finalmente, no Capítulo 5, apresentamos a metodologia para combinar os resultados das classificações dadas pelas cláusulas através de SVM. Por fim, o último capítulo será dedicado às conclusões sobre o que já foi estudado e alternativas de trabalhos futuros.

Notas sobre tradução e terminologia

Alguns termos que aparecem no texto dessa tese, tais como *ensemble*, *bagging*, *kernel*, *fold* etc. Optamos por deixá-los em inglês pois é difícil encontrar uma tradução adequada que expresse o mesmo significado do termo em português.

Outros termos foram traduzidos para o português, mas os acrônimos utilizados, tais como ILP, SVM, BK etc. são os que correspondem aos termos em inglês. Esses acrônimos foram mantidos nessa forma por serem amplamente utilizados pela comunidade.

A utilização desses termos e acrônimos não representa de forma alguma desconsideração à língua portuguesa, e somente foram utilizados pelos motivos acima apresentados.

Capítulo 2

Fundamentos teóricos

2.1 Conceitos de Lógica e Programação Lógica

Lógica de Primeira Ordem (LPO), também conhecida por Cálculo de Predicados de Primeira Ordem ou Lógica de Predicados, é um sistema dedutivo formal usado em matemática, filosofia, linguística e computação.

Diferentemente das linguagens naturais, LPO usa uma linguagem formal, sem ambiguidades, interpretada por estruturas matemáticas. LPO estende a Lógica Proposicional permitindo a quantificação sobre os indivíduos de um dado domínio de discurso. Por exemplo, podemos expressar que "Todo indivíduo tem a propriedade P".

Enquanto a Lógica Proposicional lida com proposições declarativas simples, LPO adicionalmente trata de predicados e quantificação. Tome por exemplo as seguintes sentenças: "Sócrates é um homem" e "Platão é um homem". Na Lógica Proposicional estas sentenças serão duas proposições não relacionadas denotadas por p e q. Na LPO contudo, ambas as sentenças poderiam ser conectadas pela mesma propriedade: $homem(X)$, que significa X é um homem. Quando $X = socrates$ temos a primeira proposição, p, e quando $X = platao$ temos a segunda proposição q. Tal construção permite uma lógica muito mais poderosa quando quantificadores são introduzidos, como por exemplo, "para todo X, se $homem(X)$ então...".

Além disso, a LPO permite a descrição de objetos e relações entre os objetos da maneira mais apropriada para seu domínio, como no exemplo a seguir [61]:

"Rei John, o maldoso, reinou na Inglaterra em 1200."

Objetos: John, Inglaterra, 1200.

Relações: reinou.

Propriedades: maldoso, rei.

Nesse exemplo, "rei" é considerado como uma propriedade de pessoa. Porém, se fosse mais apropriado, "rei" poderia ser uma relação entre pessoas e países ou, ainda, entre países e pessoas em um mundo onde cada país tem seu rei.

Os sistemas de aprendizado que utilizam esse tipo de linguagem de representação, são chamados de *sistemas de aprendizado relacional*. Nesses sistemas, objetos podem ser descritos estruturalmente, isto é, em termos de seus componentes e relações entre esses componentes.

O uso de LPO como linguagem de representação em sistemas de aprendizado, permite que relações ou predicados possam ser induzidos. Isso faz com que o espaço dos conceitos passíveis de serem aprendidos seja aumentado.

Esses sistemas possuem uma alta expressividade para representar conceitos e a habilidade de representar conhecimento do domínio. Além disso, os sistemas de aprendizado relacional têm a vantagem de expressarem seu conhecimento de uma forma diretamente inteligível aos humanos, característica muito importante quando o objetivo é a extração de conhecimento.

O *alfabeto* da LPO é constituído por *variáveis*, *constantes*, *funções* e *predicados*, além de símbolos de pontuação e conectivos ou operadores lógicos:

- *Variáveis* representam elementos quaisquer do domínio.
- *Constantes* dão nome a elementos particulares do domínio.
- *Funções* representam operações sobre elementos do domínio
- *Predicados* representam propriedades ou relações entre elementos do domínio.
- *Conectivos* representam as operações lógicas de conjunção ($\wedge$), disjunção ($\vee$), negação ($\neg$), implicação ($\rightarrow$) e biimplicação ($\leftrightarrow$), além das operações de quantificação universal ($\forall$) e quantificação existencial ($\exists$).

Associado a cada símbolo para função ou predicado, temos um número inteiro não-negativo que indica a sua *aridade*, ou seja, o número de argumentos da função ou predicado. Assim um predicado pode ser representado como p/n, onde p é o nome do predicado e n a sua aridade.

Um *termo* é qualquer constante, variável ou função $f(t_1, t_2, ..., t_n)$, onde $t_1, t_2, ..., t_n$ são termos.

Uma *fórmula atômica* ou *átomo* é qualquer predicado $p(t_1, t_2, ..., t_n)$, onde $t_1, t_2, ..., t_n$ são termos. As fórmulas atômicas são as fórmulas mais simples da linguagem.

Uma *fórmula bem formada* (ou simplesmente *fórmula*) é uma fórmula atômica ou uma fórmula que assume alguma das seguintes configurações: $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \leftrightarrow B)$, $\forall X(A)$ e $\exists X(A)$, onde A e B são fórmulas e X é um conjunto de variáveis.

Um *literal* é um átomo (*literal positivo*) ou um átomo negado (*literal negativo*).

Uma *cláusula* é uma disjunção de literais onde todas as variáveis são quantificadas universalmente. Uma cláusula tem a forma:

$$\forall x_1, ..., \forall x_m (L_1 \vee L_2 \vee ... \vee L_n)$$

onde cada L_i, $i = 1, ..., n$ é um literal e os x_j, $j = 1, ..., m$ são as variáveis que ocorrem na cláusula.

Uma *cláusula de Horn* é uma cláusula que contém exatamente um literal positivo tendo a seguinte forma:

$$\forall x_1, ..., \forall x_m (B \vee \neg A_1 \vee ... \vee \neg A_n)$$

ou equivalentemente[1],

$$A_1 \wedge ... \wedge A_n \rightarrow B$$

e é denominada *regra de primeira ordem.* Em Programação Lógica esta regra geralmente é representada da forma abaixo:

$$B \leftarrow A_1, A_2, ..., A_n$$

onde B é denominado *cabeça* da regra e $A_1, A_2, ..., A_n$ é denominado de *corpo* ou *cauda*. As vírgulas no corpo denotam conjunção.

Um *fato* é uma cláusula que não contém literais negativos, ou seja, é uma cláusula definida com um corpo vazio.

Um *programa Prolog* é um conjunto de fatos e/ou regras que definem relações entre objetos.

Uma cláusula *cobre* ou *prova* um exemplo se o exemplo satisfaz todos os termos da cláusula, ou seja, se o exemplo é avaliado como verdadeiro por todos os termos.

Uma *substituição* é uma função que troca variáveis por termos em uma expressão. Dada uma substituição θ e um literal L, escreve-se $L\theta$ para indicar o resultado da aplicação da substituição θ em L.

Uma *substituição unificadora* de dois literais L_1 e L_2 é uma substituição θ tal que $L_1\theta = L_2\theta$.

Um método geral para dedução automática é a *regra de resolução* introduzida por ROBINSON [60]. Ela é a base da programação lógica.

Em LPO, a regra de resolução é aplicada seguindo os passos abaixo:

[1] Já que todas as variáveis estão quantificadas universalmente, os quantificadores podem ser omitidos.

- Dadas duas cláusulas C_1 e C_2, ache um literal L_1 da cláusula C_1 e um literal L_2 da cláusula C_2, e uma substituição θ tal que $L_1\theta = \neg L_2\theta$;

$$L_1\theta = \neg L_2\theta \tag{2.1}$$

- Forme o *resolvente* C incluindo todos os literais de $C_1\theta$ e $C_2\theta$, exceto para $L_1\theta$ e $\neg L_2\theta$. Mais precisamente, o conjunto de literais ocorrendo na conclusão C é

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta \tag{2.2}$$

Por exemplo, considere as cláusulas C_1 e C_2 abaixo:

$$C_1 = mortal(X) \leftarrow humano(X)$$
$$C_2 = humano(socrates)$$

Primeiramente expressemos C_1 na forma de disjunção:

$$C_1 = mortal(X) \vee \neg humano(X)$$

e aplicamos a regra de resolução. No primeiro passo, achamos o literal $L_1 = \neg humano(X)$ de C_1 e o literal $L_2 = humano(socrates)$ de C_2 . Se escolhemos a substituição unificadora $\theta = \{X/socrates\}$ então estes dois literais satisfazem $L_1\theta = \neg L_2\theta = \neg humano(socrates)$. Portanto, a conclusão C é a união de $(C_1 - \{L_1\})\theta = mortal(socrates)$ e $(C_2 - \{L_2\})\theta = \emptyset$, ou $C = mortal(socrates)$.

A Figura 2.1 ilustra a aplicação da regra para o exemplo acima.

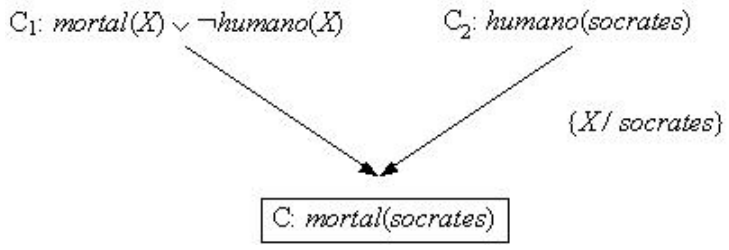


Figura 2.1: Aplicação da regra de resolução para inferir a cláusula C a partir das cláusulas C_1 e C_2.

2.2 Programação Lógica Indutiva (ILP)

Programação Lógica Indutiva (*Inductive Logic Programming* (ILP)) combina Aprendizado de Máquina com Programação Lógica [52, 53].

ILP tem ganho popularidade por 3 razões:

- oferece uma abordagem rigorosa para o problema geral de aprendizado indutivo baseado em conhecimento prévio;

- oferece algoritmos completos para induzir teorias gerais de primeira ordem a partir de exemplos, o que permite, portanto, aprender com sucesso em domínios onde algoritmos baseados em atributos são difíceis de aplicar. LPO é uma linguagem apropriada para descrever relacionamentos;

- produz hipóteses que são relativamente fáceis para interpretar.

Sistemas ILP têm sido usados com sucesso na extração de modelos relacionais de dados. Ao contrário de classificadores proposicionais ou numéricos e estatísticos, sistemas ILP produzem classificadores que possuem um grande poder de expressão, pois são baseados em LPO. Isto faz com que estes classificadores possam ser facilmente entendidos por especialistas em diversos domínios e têm sido empregados com êxito em áreas como Bioinformática [42], Engenharia [25], Processamento de Linguagem Natural [75], Meio Ambiente [21], Engenharia de Software [6], Aprendizado de Padrões e *Link Discovery* [19, 47] e *Alias Identification* [17].

ILP utiliza dados de um conjunto de exemplos de treinamento $E = E^+ \bigcup E^-$, onde E^+ e E^- são, respectivamente, conjuntos de exemplos positivos e negativos, e dados de conhecimento prévio (*background knowledge* (*BK*)), representados como conjuntos finitos de cláusulas.

A tarefa é achar uma *hipótese h* que satisfaça as restrições de linguagem (*language bias*) e que, quando conjunta com o *BK*, implique logicamente todos os exemplos positivos (*completude*) mas nenhum dos exemplos negativos (*consistência*). A hipótese é *correta* se for completa e consistente. Porém, para lidar com questões do mundo real, frequentemente relaxamos os requisitos, tal que *h* precise somente implicar significativamente mais exemplos positivos do que exemplos negativos.

Um exemplo ilustrativo simples é o aprendizado da relação $filha(X,Y)$, que estabelece que uma pessoa X é filha da pessoa Y, em termos do conhecimento prévio das relações *feminino* e *pais*. Estas relações são dadas na Tabela 2.1.

Tabela 2.1: Problema ILP simples: aprendizado da relação *filha*

Conhecimento prévio (*BK*)	
$pais(ana, maria).$	$feminino(ana).$
$pais(ana, jose).$	$feminino(maria).$
$pais(jose, carol).$	$feminino(carol).$
$pais(jose, carlos).$	

Exemplos de treinamento	
E^+	$filha(maria, ana).$
	$filha(carol, jose).$
E^-	$filha(jose, ana).$
	$filha(carol, ana).$

Assim, é possível formular a seguinte hipótese:

$$filha(X,Y) \leftarrow feminino(X), pais(Y,X).$$

que é interpretada como: X é filha de Y se X é do sexo feminino e Y é um dos pais de X.

O aprendizado em ILP pode ser visto como uma busca pelo espaço de hipóteses. Uma hipótese h_1 é dita mais geral que uma hipótese h_2, se h_1 cobrir um super-conjunto das instâncias cobertas por h_2. Desta forma h_1é uma *generalização* de h_2 e h_2 é uma *especialização* de h_1. Usando esta relação de generalidade, o espaço de hipóteses pode ser ordenado num reticulado (*lattice*) contendo uma hipótese mais geral (cobre todos os exemplos positivos) e uma hipótese mais específica (cobre a disjunção de todos os exemplos positivos). Seguindo a ordenação imposta pelo reticulado, os algoritmos de ILP podem fazer busca *top-down* (especialização), *bottom-up* (generalização) ou bidirecional.

Na abordagem *top-down*, o algoritmo de aprendizado começa com uma regra mais geral e gradualmente a especializa até que ela se adeque aos dados. Isto é essencialmente uma generalização do método de aprendizado de Árvores de Decisão, onde a árvore cresce gradualmente até ficar consistente com as observações. Em ILP usamos literais de primeira ordem ao invés de atributos nos nós da árvore, e a hipótese é um conjunto de cláusulas ao invés da árvore de decisão. Um exemplo de sistema ILP que atua desta forma é o FOIL [58].

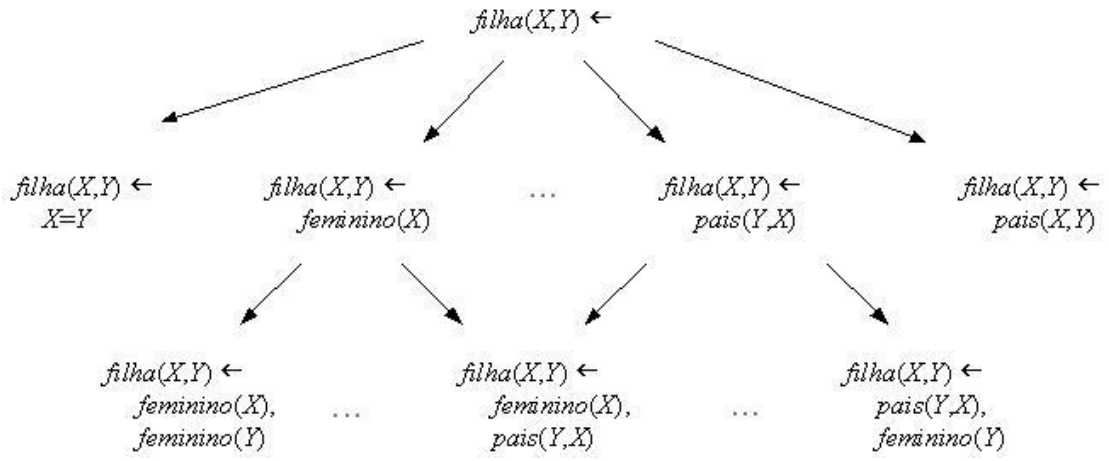


Figura 2.2: Exemplo de busca *top-down*. A busca é iniciada pela cláusula mais geral $filha(X,Y) \leftarrow$ que vai se especializando até encontrar a solução.

A Figura 2.2 mostra uma parte do grafo de refinamento da busca *top-down* para a relação $filha(X,Y)$ do exemplo da Tabela 2.1. A busca é iniciada pela cláusula $filha(X,Y) \leftarrow$, a qual cobre dois exemplos negativos. Os refinamentos da cláusula são considerados em

seguida, dos quais $filha(X,Y) \leftarrow feminino(X)$ e $filha(X,Y) \leftarrow pais(Y,X)$ cobrem apenas um exemplo negativo. As duas cláusulas tem um refinamento em comum $filha(X,Y) \leftarrow feminino(X), pais(Y,X)$ que não cobre os exemplos negativos, terminando assim a busca.

Um grafo de refinamento é tipicamente percorrido heuristicamente, usando heurísticas baseadas em números de exemplos positivos e negativos cobertos pela cláusula. Algumas das técnicas de busca que podem ser empregadas são *hill-climbing*, *A* best-first* e *breadthfirst* [43] .

A abordagem *bottom-up* é baseada na inversão da regra de resolução para inferência dedutiva. Esta é a base do sistema CIGOL [50].

Podemos derivar a *regra da resolução inversa* analiticamente, pela manipulação algébrica da Equação 2.2 que define a regra de resolução [46].

A substituição unificadora θ na Equação 2.2 pode ser fatorada em θ_1 e θ_2, onde $\theta = \theta_1\theta_2$, e θ_1 contém todas as substituições envolvendo variáveis da cláusula C_1, e θ_2 contém todas as substituições envolvendo variáveis da cláusula C_2. Usando esta fatoração de θ, podemos reescrever a Equação 2.2 como[2]

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2$$

Agora se restringirmos a resolução inversa para inferir apenas cláusulas C_2 que não contém literais em comum com C_1 (dando preferência para as cláusulas C_2 mais curtas), então podemos reexpressar a fórmula acima como

$$C - (C_1 - \{L_1\})\theta_1 = (C_2 - \{L_2\})\theta_2$$

Finalmente usando o fato que por definição da regra de resolução $L_2 = \neg L_1\theta_1\theta_2^{-1}$ (Equação 2.1), obtemos a seguinte solução para C_2:

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\} \tag{2.3}$$

A Equação 2.3 nos dá a regra da resolução inversa para a LPO.

A Figura 2.3 ilustra a aplicação da regra para o problema da Tabela 2.1 onde desejamos aprender a relação $filha(X,Y)$. Considere a cláusula no nível mais baixo da árvore da resolução inversa da figura. Aqui, nós fazemos a conclusão C ser o exemplo de treinamento $filha(maria,ana)$ e selecionamos a cláusula $C_1 = pais(ana,maria)$ do BK. Para aplicar a regra da resolução inversa nós temos uma única escolha para o literal L_1, ou seja, $pais(ana,maria)$. Suponha que escolhemos a substituição inversa $\theta_1 = \{\}$ e $\theta_2^{-1} = \{ana/Y\}$. Neste caso, a cláusula resultante C_2 é a união da cláusula $(C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} = (C\theta_1)\theta_2^{-1} = filha(maria,Y)$, e a cláusula $\{\neg L_1\theta_1\theta_2^{-1}\} = \neg pais(Y,maria)$. Logo o resultado é a cláusula $filha(maria,Y) \vee \neg pais(Y,maria)$, ou

[2]lembrando que o sinal "−" simboliza diferença de conjuntos

equivalentemente $filha(maria,Y) \leftarrow pais(Y,maria)$. Note que esta regra geral, junto com C_1 implica o exemplo de treinamento $filha(maria,ana)$.

De forma semelhante, esta cláusula inferida pode agora ser usada como a conclusão C para o próximo passo da resolução inversa. A cada passo existem vários resultados possíveis, dependendo das escolhas para as substituições. No exemplo da Figura 2.3, o conjunto particular de escolhas produz a cláusula final $filha(X,Y) \leftarrow feminino(X), pais(Y,X)$.

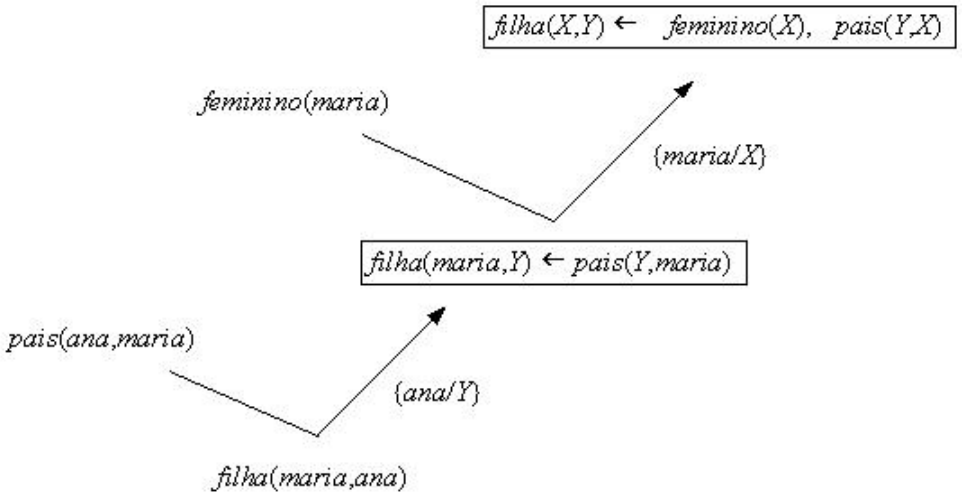


Figura 2.3: Passos da resolução inversa. Em cada passo, C é a cláusula de baixo, C_1 a cláusula à esquerda, e C_2, a cláusula dentro do retângulo à direita, que é o resultado da inferência. Em ambos os passos, θ_1 é a substituição vazia $\{\}$, e θ_2^{-1} é a substituição mostrada abaixo de C_2.

Embora a resolução inversa seja um método interessante para gerar hipóteses, na prática pode facilmente levar a uma explosão combinatória de hipóteses candidatas. Uma abordagem alternativa é usar implicação lógica inversa para gerar apenas a única hipótese mais específica que, junto com o conhecimento prévio, implica os dados observados. Esta hipótese mais específica pode ser usada para limitar uma busca *top-down* através do espaço de hipóteses similar à usada pelo FOIL, mas com a restrição adicional que as únicas hipóteses consideradas são as hipóteses mais gerais do que esta hipótese específica. Esta abordagem é empregada pelos sistemas Progol [49] e Aleph [71]. Este último será detalhado na próxima seção. A maioria dos sistemas ILP que utilizam esta abordagem usam um algoritmo guloso de cobertura para achar um conjunto de cláusulas que melhor modela os exemplos positivos. Este conjunto de cláusulas é chamado de *teoria*.

Estes algoritmos geram árvores, cujos nós correspondem às combinações de literais que podem aparecer no corpo de uma cláusula e são algoritmos de busca com complexidade exponencial no número de fatos e constantes associadas a estes fatos. Para contornar este problema e limitar o espaço de busca, pode-se utilizar as restrições de linguagem mencionadas anteriormente, porém ainda são necessárias formas de controlar o crescimento deste espaço de busca.

Várias técnicas têm sido aplicadas para melhorar a busca dos sistemas ILP e outras técnicas têm sido aplicadas para melhorar a qualidade dos classificadores produzidos por estes sistemas. Tais técnicas incluem:

- melhoramento do tempo de computação nos nós individuais [3, 64];
- melhor representação da busca [4, 12, 29, 31];
- amostragem do espaço de busca [69, 70, 74];
- paralelismo [22, 27, 32, 30, 36, 45, 72];
- utilização de métodos de *ensemble* [26, 63].

Na próxima seção será abordado o sistema ILP Aleph que foi utilizado em todo o decorrer deste trabalho. Aleph foi utilizado por causa de sua popularidade e por ser um sistema robusto e flexível, permitindo uma ampla variação de estratégias de busca, funções de avaliação de cláusulas, entre outros parâmetros.

2.2.1 Aleph

Aleph (*A Learning Engine for Proposing Hypotheses*) é um algoritmo ILP escrito completamente em Prolog desenvolvido na Universidade de Oxford.

Como entrada, Aleph toma informação de conhecimento prévio na forma de predicados, uma lista de tipos e modos declarando como estes predicados podem ser encadeados, e a designação de um predicado como o *predicado alvo*, a hipótese a ser aprendida. Também são requeridas listas de exemplos positivos e negativos do predicado alvo.

As declarações de modo são uma forma de definir (restringir) as relações entre diversos literais nas cláusulas geradas, como:

- Os literais que podem aparecer no corpo e na cabeça;
- O tipo de cada termo em um predicado;
- A definição de termos de entrada e saída em um predicado;
- O número máximo de literais com os quais um predicado pode se relacionar

De modo geral, Aleph gera cláusulas para os exemplos positivos selecionando um exemplo aleatório para ser uma semente. Este exemplo é *saturado* para criar a cláusula saturada (*bottom clause*), isto é, toda relação no *background knowledge* que pode ser alcançada a partir deste exemplo [49]. O processo de saturação funciona da seguinte forma: a partir do exemplo semente, Aleph busca na base de dados pelos fatos conhecidos como verdadeiros para aquele exemplo específico; a combinação destes fatos ou subconjunto destes

fatos cobre este exemplo e poderia ser possível generalizar esta combinação tal que ela também cobrisse outros exemplos.

A cláusula saturada restringe o espaço de busca para cláusulas mais gerais. Aleph heuristicamente faz a busca através do espaço de cláusulas possíveis até que a "melhor" cláusula é achada ou o tempo é esgotado. O modo padrão do Aleph é combinar as cláusulas aprendidas em uma teoria até quando cláusulas suficientes são aprendidas para cobrir quase todo o conjunto de exemplos positivos.

O procedimento do Aleph [71] pode ser descrito através dos passos apresentados no Algoritmo 1.

Algoritmo 1 Procedimento do Aleph

---

1. Selecionar exemplo: seleciona aleatoriamente um exemplo positivo para ser generalizado. Se não existir nenhum, pare, senão proceda ao próximo passo;

2. Construir cláusula mais específica: constrói a cláusula mais específica que vincule o exemplo selecionado dentro das restrições de linguagem. O resultado deste passo é uma cláusula com vários literais relacionados ao exemplo selecionado que é chamada de *cláusula saturada* (*bottom clause*). Este passo é chamado de *saturação*;

3. Busca: encontra uma cláusula mais geral que a cláusula saturada. Isto é feito procurando por algum subconjunto de literais da cláusula saturada que tenha a melhor avaliação segundo uma heurística escolhida. Este passo é chamado de *redução*;

4. Remover redundância: a cláusula com a melhor avaliação é adicionada à teoria corrente e todos os exemplos cobertos por ela são removidos. Além disso, a melhor cláusula pode também remover outras cláusulas redundantes. Este passo é chamado de *remoção de cobertura* (*cover-removal*);

5. Retornar ao passo 1.

No Aleph pode-se configurar várias estratégias de busca de cláusulas que melhor se adequem à base de dados. A estratégia de busca adotada neste trabalho foi a busca em largura. Esta busca enumera cláusulas mais curtas antes das mais longas. Dado um tamanho de cláusula, as cláusulas são reordenadas baseadas nas suas avaliações. Esta é a estratégia padrão do Aleph que favorece cláusulas mais curtas para evitar a complexidade de refinar cláusulas maiores.

Além de permitir a alteração da estratégia de busca, Aleph também permite que o usuário modifique vários outros parâmetros, como:

- Número máximo de literais em uma cláusula (*clauselength*): este parâmetro define o maior comprimento de uma cláusula.

- Função de avaliação (*evalfn*): este parâmetro é utilizado para medir a qualidade de uma cláusula;

- Encadeamento das variáveis (*i*): este parâmetro controla o encadeamento de variáveis durante a saturação. O valor do encadeamento de uma variável nova a ser colocada como um argumento de um literal L_i após o refinamento é calculado da seguinte forma: 1 + valor do encadeamento de todas as variáveis que aparecem em literais anteriores L_j, $j < i$;

- Número máximo de nós permitidos (*nodes*): este parâmetro corresponde ao número máximo de cláusulas a ser pesquisado no espaço de busca;

- Acurácia mínima da cláusula (*minacc*): este parâmetro define a acurácia mínima aceitável ao gerar uma nova cláusula.

Aleph permite a modificação de vários outros parâmetros, porém, em nossos experimentos, apenas controlamos os parâmetros *clauselength*, *evalfn*, *i*, *nodes* e *minacc* que variam o espaço de busca e modificam a avaliação das cláusulas.

Exemplos

Com o objetivo de ilustrar os passos de execução do Aleph, a seguir é mostrado o exemplo dos trens de Michalski. Dada uma sequência de trens (Figura 2.4) cada trem tem um número de vagões e cada vagão pode ter diferentes propriedades, indicadas pelo desenho de uma figura com um determinado formato. Também sabe-se se o trem está viajando para o Leste (*East*) ou para o Oeste (*West*). O problema é encontrar uma regra capaz de predizer, dada as propriedades dos seus vagões, se o trem está viajando para Leste; esse trem é chamado de *eastbound*.

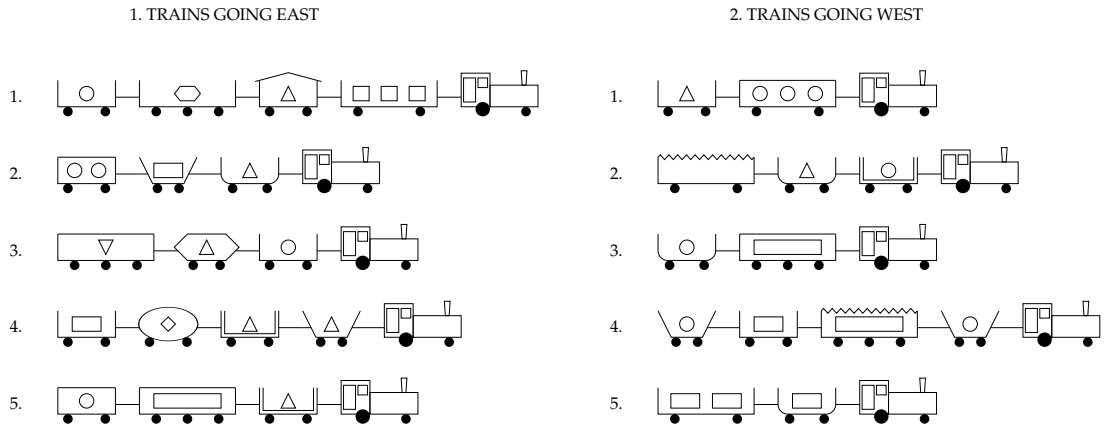


Figura 2.4: Trens que viajam para Leste e Oeste

Aleph utiliza dois arquivos distintos para armazenar os exemplos positivos e negativos. Abaixo são mostrados os conteúdos destes dois arquivos para o exemplo dos trens:

```
trens.f (arquivo com os exemplos positivos)

    eastbound(east1).
    eastbound(east2).
    eastbound(east3).
    eastbound(east4).
    eastbound(east5).

trens.n (arquivo com os exemplos negativos)

    eastbound(west6).
    eastbound(west7).
    eastbound(west8).
    eastbound(west9).
    eastbound(west10).
```

O próximo passo é definir os *modos* e *tipos* para a execução do Aleph e declarar o conhecimento prévio do problema. Isto é feito no arquivo chamado trens.b cujo conteúdo é apresentado abaixo:

```
% Declaração de modos
% modeh/2 define a relação na cabeça da regra
% modeb/2 define os literais utilizados no BK com seus tipos e modos
:- modeh(1,eastbound(+train)).
:- modeb(1,short(+car)).
:- modeb(1,closed(+car)).
:- modeb(1,long(+car)).
:- modeb(1,open_car(+car)).
:- modeb(1,double(+car)).
:- modeb(1,jagged(+car)).
:- modeb(1,shape(+car,#shape)).
:- modeb(1,load(+car,#shape,#int)).
:- modeb(1,wheels(+car,#int)).
:- modeb(*,has_car(+train,-car)).
```

O conceito a ser aprendido é denominado eastbound/1, portanto deve ser declarado com modeh/2, e os literais utilizados no *BK* com seus tipos e modos devem ser declarados com modeb/2. O símbolo "+" que aparece antes do tipo indica que esse argumento do predicado é uma variável. Assim, a cabeça pode ter a forma $eastbound(X)$, mas não $eastbound(east1)$. O símbolo "$-$" indica uma variável de saída e o símbolo "#" indica que esse argumento pode ser uma constante.

Os tipos devem ser especificados para cada argumento dos predicados utilizados na construção de uma hipótese. Para o Aleph os tipos são só nomes e a declaração de tipos

nada mais é que um conjunto de fatos. Para definir o tipo de dados car/1 enumeramos os possíveis valores que pode assumir, esses valores foram definidos considerando primeiro o trem e segundo o vagão. Por exemplo car(car_14) refere-se ao valor do tipo car para o trem 1 vagão 4. Também são definidos shape/1 que define os possíveis formatos das figuras relacionadas às propriedades dos vagões, bem como train/1 que define os trens.

```
% Definição de tipos
car(car_11). car(car_12). car(car_13). car(car_14).
car(car_21). car(car_22). car(car_23).
car(car_31). car(car_32). car(car_33).
car(car_41). car(car_42). car(car_43). car(car_44).
car(car_51). car(car_52). car(car_53).
car(car_61). car(car_62).
car(car_71). car(car_72). car(car_73).
car(car_81). car(car_82).
car(car_91). car(car_92). car(car_93). car(car_94).
car(car_101). car(car_102).
shape(elipse). shape(hexagon). shape(rectangle). shape(u_shaped).
shape(triangle). shape(circle). shape(nil).
train(east1). train(east2). train(east3).
train(east4). train(east5).
train(west6). train(west7). train(west8).
train(west9). train(west10).
```

Também, devem ser declarados os determination/2, especificando os predicados que podem aparecer na cabeça e no corpo das regras com essa cabeça, como mostrado a seguir:

```
% Determinations
% Predicados que podem ser usados para construir uma hipótese
:- determination(eastbound/1,short/1).
:- determination(eastbound/1,closed/1).
:- determination(eastbound/1,long/1).
:- determination(eastbound/1,open_car/1).
:- determination(eastbound/1,double/1).
:- determination(eastbound/1,jagged/1).
:- determination(eastbound/1,shape/2).
:- determination(eastbound/1,wheels/2).
:- determination(eastbound/1,has_car/2).
:- determination(eastbound/1,load/3).
```

O próximo passo consiste em definir cada um dos 10 trens. Por exemplo, o trem 1 é definido como:

```
% Definição do trem 1
short(car_12). % vagão 2 é curto
closed(car_12). % vagão 2 é fechado
long(car_11). % vagão 1 é longo
long(car_13). % vagão 3 é longo
short(car_14). % vagão 4 é longo
open_car(car_11). % vagão 1 é aberto
open_car(car_13). % vagão 3 é aberto
open_car(car_14). % vagão 4 é aberto
shape(car_11,rectangle). % vagão 1 tem forma rectangular
shape(car_12,rectangle). % vagão 2 tem forma rectangular
shape(car_13,rectangle). % vagão 3 tem forma rectangular
shape(car_14,rectangle). % vagão 4 tem forma rectangular
load(car_11,rectangle,3). % vagão 1 carrega 3 retângulos
load(car_12,triangle,1). % vagão 2 carrega 1 triângulo
load(car_13,hexagon,1). % vagão 3 carrega 1 hexágono
load(car_14,circle,1). % vagão 4 carrega 1 círculo
wheels(car_11,2). % vagão 1 tem 2 rodas
wheels(car_12,2). % vagão 2 tem 2 rodas
wheels(car_13,3). % vagão 3 tem 3 rodas
wheels(car_14,2). % vagão 4 tem 2 rodas
has_car(east1,car_11). % o vagão 1 é do trem east1
has_car(east1,car_12). % o vagão 2 é do trem east1
has_car(east1,car_13). % o vagão 3 é do trem east1
has_car(east1,car_14). % o vagão 4 é do trem east1
```

Os outros 9 (nove) trens devem ser descritos de maneira semelhante no arquivo trens.b.

Aleph executa então os quatro passos descritos no Algoritmo 1. No passo 1 seleciona um exemplo para ser generalizado, por exemplo, *eastbound(east1)*. No passo 2 é construída a cláusula saturada do exemplo, mostrada a seguir:

```
[bottom clause]
eastbound(A) :-

    has_car(A,B), has_car(A,C), has_car(A,D), has_car(A,E),
    short(B), short(D), closed(D), long(C),
    long(E), open_car(B), open_car(C), open_car(E),
    shape(B,rectangle), shape(C,rectangle),
    shape(D,rectangle),shape(E,rectangle),
    wheels(B,2), wheels(C,3), wheels(D,2), wheels(E,2),
```

```
    load(B,circle,1), load(C,hexagon,1),
    load(D,triangle,1), load(E,rectangle,3).
[literals] [25]
[saturation time] [0.01]
```

No passo 3, o algoritmo procura a cláusula mais geral através da busca *top-down* restringida pelos literais da cláusula saturada e pelas restrições de modos e tipos, guiada pelos parâmetros de busca.

```
eastbound(A).
[5/5]
eastbound(A) :-
   has_car(A,B).
...
[5/5]
eastbound(A) :-
   has_car(A,B), short(B).
[5/5]
eastbound(A) :-
   has_car(A,B), open_car(B).
[5/5]
eastbound(A) :-
   has_car(A,B), shape(B,rectangle).
...
eastbound(A) :-
   has_car(A,B), closed(B), shape(B,rectangle).
eastbound(A) :-
   has_car(A,B), closed(B), wheels(B,2).
eastbound(A) :-
   has_car(A,B), closed(B), load(B,triangle,1).
[2/0]
...
eastbound(A) :-
   has_car(A,B), short(B), closed(B).
[5/0]
...
[clauses constructed] [100]
[search time] [0.05]
[best clause]
eastbound(A) :-
```

```
   has_car(A,B), short(B), closed(B). [pos-neg] [5]
[atoms left] [0]
[theory]
[Rule 1] [Pos cover = 5 Neg cover = 0]
eastbound(A) :-
    has_car(A,B), short(B), closed(B).
```

Neste caso, a cláusula encontrada é a melhor cláusula (cobre os 5 exemplos positivos e nenhum negativo). Assim, ela é adicionada à teoria no passo 4, e os exemplos positivos cobertos são removidos como futuras escolhas de sementes. Como não há mais exemplos positivos, Aleph pára. Também são informados a matriz de confusão (ver Seção 2.5) da teoria induzida no conjunto de treinamento e outras estatísticas, como mostrado a seguir.

```
[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] [0.07]
[total clauses constructed] [100]
```

O exemplo dos trens é muito bom para mostrar todo o processo do Aleph. No entanto, ele não mostra todo o poder relacional do sistema ILP.

Para isso, será apresentado a seguir, de forma concisa, um outro exemplo retirado da base de dados *Protein* utilizada em nossos experimentos e descrita na Seção 3.2.

Considere o exemplo do gene “G235580” descrito da seguinte forma no BK:

```
chromosome('G235580','11').
complex('G235580','Respiration chain complexes').
essential('G235580','Non-Essential').
interaction('G235580','G236280','Physical','0.342639674').
motif('G235580','PS00504').
phenotype('G235580','"Auxotrophies, carbon and"').
```

Podemos interpretar o conhecimento prévio acima da seguinte forma: o gene está no cromossomo 11, pertence ao complexo da cadeia respiratória, é não-essencial, interage

fisicamente com o gene “G236280” com força 0,342639674, possui *motif* “PS00504” e tem fenótipo *“Auxotrophies, carbon and*”

A cláusula saturada gerada para este exemplo é a seguinte:

```
[metabolism(G235580)]
[bottom clause]
metabolism(A) :-
    essential(A,'Non-Essential'),
    complex(A,'Respiration chain complexes'),
    phenotype(A,'"Auxotrophies, carbon and"'),
    motif(A,'PS00504'), chromosome(A,'11'),
    interaction(A,B,C,D), essential(B,'Non-Essential'),
    intertype(C,'Physical'),
    phenotype(B,'"Auxotrophies, carbon and"'),
    chromosome(B,'2'), interaction(B,E,C,G),
    interaction(B,A,C,D), interaction(B,F,C,H),
    essential(E,'Non-Essential'), essential(F,'Non-Essential'),
    complex(E,'Respiration chain complexes'),
    complex(F,'Respiration chain complexes'),
    phenotype(E,'"Auxotrophies, carbon and"'),
    motif(E,'PS00197'), motif(F,'PS01000'),
    chromosome(E,'12'), chromosome(F,'11'),
    interaction(E,B,C,G), interaction(F,B,C,H).
[literals] [25]
```

A cláusula saturada descreve não só o gene identificado como “G235580”, mas também todos os outros genes que, no BK, interagem com este. Ou seja, cláusulas generalizadas geradas a partir desta cláusula saturada, podem representar mais do que a simples descrição de um gene - podem também representar as relações entre genes. Este tipo de relação é difícil de ser aprendido com sistemas proposicionais.

O processo de busca do Aleph restringido pela cláusula saturada e parâmetros de busca gera a seguinte cláusula:

```
metabolism(A) :-
    phenotype(A,'"Auxotrophies, carbon and"'),
    interaction(A,B,_,_), essential(B,'Non-Essential'),
    chromosome(B,'4').
```

que pode ser interpretada como: um gene *A* participa do metabolismo se *A* tem fenótipo *“Auxotrophies, carbon and*”, *A* interage com outro gene *B*, *B* é não-essencial e *B* está no cromossomo 4.

2.3 *Ensembles*

Um *ensemble* é um classificador formado por um conjunto de classificadores cujas decisões individuais são combinadas para classificar novos exemplos. *Ensembles* são freqüentemente mais acurados do que os seus classificadores individualmente, desde que estes sejam acurados e diversos [23]. Um classificador acurado é aquele cuja probabilidade de responder corretamente é maior que 50%. Dois classificadores são diversos se eles produzem seus erros em diferentes pontos do espaço de busca.

HANSEN e SALAMON [37] provaram que se a taxa média de erro dos classificadores individuais for abaixo de 50% e se eles forem independentes na classificação dos exemplos, a taxa de erro combinada pode ser reduzida a 0 quando o número de classificadores que compõem o *ensemble* tender a infinito.

DIETTERICH [23] fez um levantamento dos métodos para construção de *ensembles* e mostrou três razões fundamentais (estatística, computacional e representacional) do porquê métodos de *ensemble* são capazes de superar os classificadores que os compõem.

A geração dos *ensembles* é feita em duas fases:

1. treinamento dos classificadores individuais;
2. combinação das saídas dos classificadores.

Os métodos de *ensemble* variam de acordo com as restrições impostas na fase de treinamento e o tipo de combinação usada. Em geral, estes métodos trabalham combinando as predições de vários classificadores fracos (de baixa acurácia) para produzir um classificador final forte (de alta acurácia).

A Figura 2.5 mostra a estrutura de um *ensemble*. Os classificadores podem ser do mesmo tipo (classificadores de um mesmo modelo) ou de tipos diferentes (classificadores de diferentes modelos).

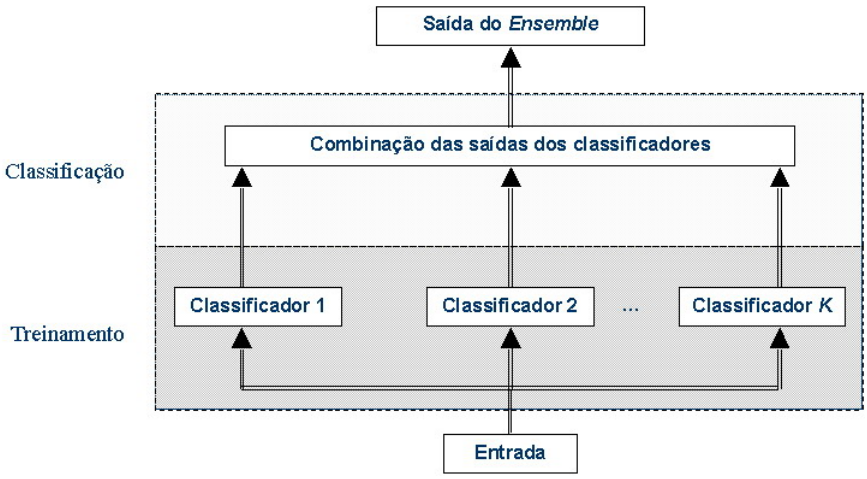


Figura 2.5: Estrutura de um *ensemble*

Para se obter bons *ensembles* devemos lidar com três tipos diferentes de questões [26]:

1. Como gerar os classificadores individualmente: qual o algoritmo de aprendizado será utilizado;

2. Quantos classificadores gerar: quantidade de classificadores individuais que integram o *ensemble*;

3. Como combinar as saídas dos classificadores individuais para gerar a saída do *ensemble*.

2.3.1 Criação dos classificadores individuais

Os métodos para criação dos classificadores individuais devem focar em produzir classificadores com algum grau de diversidade. Duas formas para que isso aconteça são: 1) variar a estrutura de cada classificador (topologia e/ou parâmetros), e 2) variar o conjunto dos dados de treinamento de cada classificador.

Conjuntos de Redes Neuronais, onde cada rede é inicializada com pesos diferentes [55] e de Árvores de Decisão, onde os atributos que dividem os nós internos de cada árvore são escolhidos aleatoriamente [24] são exemplos da primeira forma para gerar classificadores individuais para o *ensemble*.

Em ILP, DUTRA *et al.* [26] se beneficiam da busca aleatória baseada em semente do Aleph e simplesmente combinam diferentes cláusulas do experimento de diferentes exemplos semente, usando sempre o conjunto de treinamento original. Esta abordagem foi chamada *different seeds*.

Outra forma para criar diversidade entre os classificadores é obtê-los a partir de diferentes amostragens do conjunto de treinamento. Os dois métodos mais populares para isso são: *bagging* e *boosting* [44].

No método de *bagging* (termo surgido de ***b**ootstrap **agg**regati**ng***) [7], cada classificador é individualmente obtido através de uma amostra aleatória do conjunto de treinamento. São sorteados aleatoriamente N exemplos, com reposição, onde N é o tamanho do conjunto de treinamento original (processo chamado de *bootstrap replicate*).

Assim, muitos exemplos podem aparecer repetidos nos conjuntos de treinamento dos classificadores. Isto contribui para que os classificadores sejam diferentes e se concentrem no aprendizado de grupos de exemplos.

Dentre as principais vantagens deste método estão:

- é efetivo em "algoritmos de aprendizado instáveis" (ou seja, algoritmos que com pequenas mudanças no conjunto de treinamento apresentam grandes mudanças no classificador), como é o caso de ILP;

- é pouco suscetível à *overfitting* [44];

- pode ser trivialmente implementado em paralelo [26] uma vez que os conjuntos de treinamento são independentes entre si.

A Figura 2.6 mostra um esquema ilustrativo da geração do *ensembles* através do método de *bagging*.

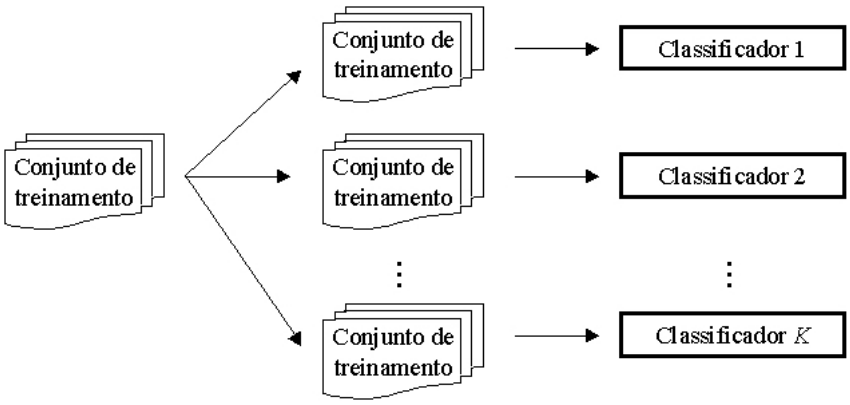


Figura 2.6: *Bagging*: cada classificador é individualmente obtido através de amostras aleatórias (com reposição) do conjunto de treinamento original

No método de *boosting*, o conjunto de treinamento usado para cada classificador é gerado baseado no desempenho do classificador anterior. Este método funciona atribuindo penalidades aos exemplos mal classificados, e refina a busca por cláusulas que tentam cobrir estes exemplos.

Diferentemente do método de *bagging*, a probabilidade de selecionar um exemplo não é igual para todo o conjunto de treinamento. São selecionados probabilisticamente, com reposição, N exemplos do conjunto de treinamento do classificador K para gerar o classificador $K+1$. Esta probabilidade depende do quão freqüente o exemplo foi mal classificado pelos K classificadores anteriores.

Na literatura podem ser verificados dois tipos de métodos de *boosting*: *arcing* e *ada-boosting*. Ambos inicializam um conjunto de probabilidades atribuindo para cada exemplo a probabilidade $1/N$. Estes métodos então recalculam estas probabilidades após cada classificador ser treinado e incluído ao *ensemble*. A diferença desses métodos está na maneira de fazer o cálculo das probabilidades em cada iteração.

No método *ada-boosting* [33], as probabilidades para a próxima geração são geradas pela multiplicação das probabilidades das instâncias incorretamente classificadas pelo fator

$$\beta = (1-\varepsilon_k)/\varepsilon_k$$

onde ε_k é a soma das probabilidades das instâncias mal classificadas do classificador corrente. Estas probabilidades são então normalizadas.

O método *arcing* atualiza as probabilidades um tanto diferentemente. A probabilidade para selecionar o exemplo i para ser parte do conjunto de treinamento do classificador $K+1$ é definida como

$$p_i = (1+m_i^4)/\sum_{j=1}^{N}(1+m_j^4)$$

onde m_i refere-se ao número de vezes que o exemplo foi mal classificado pelos K classificadores anteriores. BREIMAN [8] escolheu o valor da potência empiricamente após tentar diversos valores diferentes[3].

A Figura 2.7 mostra um esquema ilustrativo da geração de *ensembles* através do método de *boosting*.

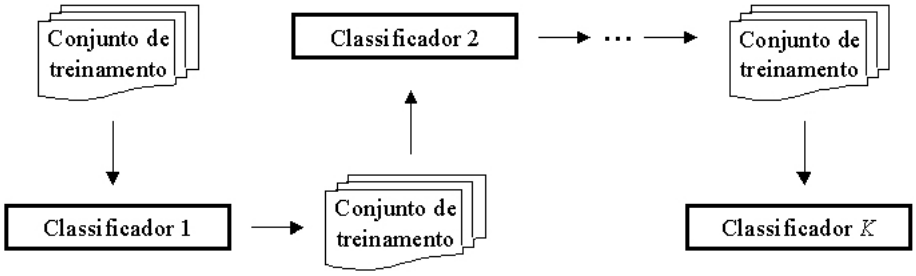


Figura 2.7: *Boosting*: cada classificador é obtido através de amostras probabilísticas baseadas no desempenho do classificador anterior

2.3.2 Tamanho do *ensemble*

A próxima questão está relacionada à escolha da quantidade de classificadores que devemos ter no *ensemble* (tamanho do *ensemble*). Pesquisas anteriores têm mostrado ganhos com até 25 classificadores [26, 37]. No próximo capítulo será apresentada uma abordagem de *ensemble* de cláusulas onde é possível ter ganhos com tamanhos de *ensembles* maiores

2.3.3 Combinação dos classificadores individuais

A última questão diz respeito à forma de combinar as saídas dos classificadores individuais. A forma mais comum de fazer esta combinação é através de métodos de votação.

O método *ada-boosting* combina os classificadores $C_1,...,C_k$ usando votação ponderada, onde C_k tem peso $\log(\beta_k)$. O método *arcing* combina seus classificadores por votação sem peso [44].

Uma das abordagens de votação bastante utilizada é a *votação com limiar* (*voting threshold*) assim definida: se o número de classificadores individuais que votam com

[3]Esta fórmula também é conhecida na literatura como *arcing-x4*

a mesma classificação para um exemplo está acima ou igual a um determinado limiar, dizemos que o exemplo possui tal classificação.

Os limiares variam de 1 até o tamanho do *ensemble*. Um limiar 1 de votação corresponde ao classificador que é a disjunção de todos os classificadores individuais. Um limiar de votação igual ao tamanho do *ensemble* corresponde a um classificador que é a conjunção de todos os classificadores individuais.

Por exemplo, para um *ensemble* de tamanho 3 e um exemplo com classificação binária (positivo ou negativo):

1. Selecione aleatoriamente 3 classificadores individuais do *ensemble*. Suponha que estes classificadores sejam c_1, c_2 e c_3.

2. Neste caso, o limiar varia de 1 a 3.

    2.1 Quando o limiar for 1, basta um dos classificadores individuais classificar o exemplo como positivo para que ele seja considerado positivo. Isto equivale a fazer: $(c_1 \vee c_2 \vee c_3)$

    2.2 Quando o limiar for 2, dois dos classificadores individuais devem classificar o exemplo como positivo para que ele seja considerado positivo. Isto equivale a fazer: $((c_1 \vee c_2) \wedge c_3) \vee ((c_1 \vee c_3) \wedge c_2) \vee ((c_2 \vee c_3) \wedge c_1)$

    2.3 Quando o limiar for 3, os três classificadores individuais têm que classificar o exemplo como positivo para que ele seja considerado positivo. Isto equivale a fazer: $(c_1 \wedge c_2 \wedge c_3)$

São realizadas votações para tamanhos de *ensembles* de 1 até K, onde K é o número total de classificadores do *ensemble*.

O procedimento de votação funciona como se fosse um algoritmo de força bruta para tentar todas as combinações de classificadores individuais que maximize a acurácia.

2.4 Máquinas de Vetor Suporte (SVM)

Máquinas de Vetor Suporte (*Support Vector Machines* (SVM)), foram introduzidas por VAPNIK *et al.* [5] e estão relacionadas a métodos de aprendizado supervisionado usados para classificação e regressão.

Elas pertencem à família de classificadores lineares generalizados. Nas tarefas mais simples de reconhecimento de padrões, SVM usa um hiperplano separador linear para criar um classificador com uma margem máxima. Para fazer isso, o problema de aprendizado é moldado como um problema de otimização não-linear com restrições. Nesta definição a função objetivo é quadrática e as restrições são lineares, ou seja, devemos resolver um problema de programação quadrática.

O hiperplano resultante será ótimo no sentido de ser um classificador de margem máxima com respeito aos dados de treinamento e é aquele que possui a melhor generalização [68]. Este tipo de hiperplano é conhecido como *hiperplano separador ótimo* ou *hiperplano de margem máxima* e é único. Na Figura 2.8 são mostrados dois exemplos de classificadores lineares.

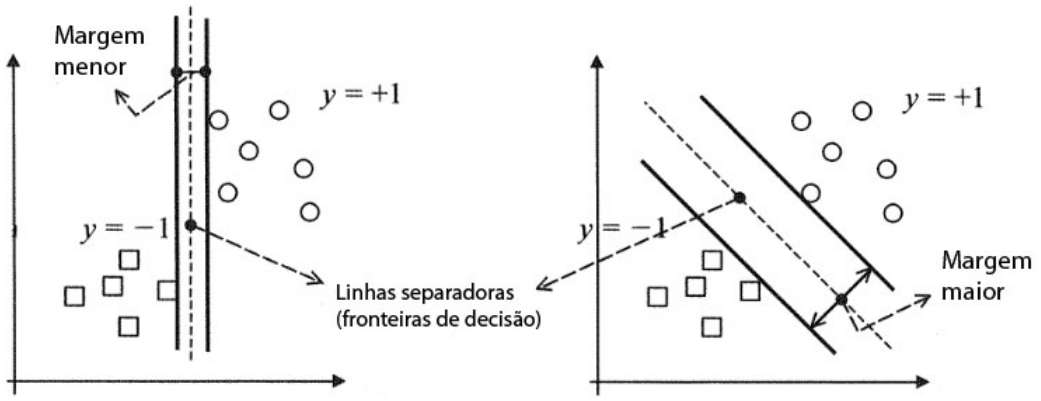


Figura 2.8: Dois classificadores lineares separando o mesmo conjunto de dados. O da direita, com margem maior, é considerado um bom classificador enquanto que o da esquerda é menos aceitável.

Em problemas onde hiperplanos lineares não são mais viáveis, funções não-lineares mapeiam os dados de entrada para um espaço transformado (espaço de características) de dimensão mais alta onde o hiperplano separador é construído.

No problema de reconhecimento de padrões, SVMs têm sido usadas para reconhecimento de dígitos manuscritos isolados [11, 14, 66, 67], reconhecimento de objetos [2], identificação de locutor [65], detecção de faces em imagens [54], categorização de texto [38] e bioinformática [9].

2.4.1 SVMs lineares

Considere o problema de classificação binária onde o conjunto de treinamento é dado por $T = \{(\mathbf{x}_i, y_i)\}, i = 1,...,l, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1,1\}$, onde cada $\mathbf{x}_i$ é um vetor real n-dimensional, usualmente normalizado[4] por valores nos intervalos $[0,1]$ ou $[-1,1]$ e y_i é uma constante $+1$ ou -1, denotando a classe a que o ponto $\mathbf{x}_i$ pertence.

SVMs lineares são utilizadas para classificação de conjuntos de dados linearmente separáveis e conjuntos de dados com sobreposição (não-linearmente separáveis). Nesta seção dividimos os dois tipos de classificação para uma melhor construção de conceitos.

[4] a escala é importante para precaver-se contra variáveis (atributos) com maior variância que poderiam dominar a classificação

2.4.1.1 Classificação de dados linearmente separáveis

Seja o hiperplano H definido pelo par $(\mathbf{w}, b)$ que separe os exemplos positivos dos exemplos negativos, chamado *hiperplano separador*. Os pontos $\mathbf{x}$ que pertencem ao hiperplano satisfazem a equação

$$H: \mathbf{w} \cdot \mathbf{x} + b = 0$$

onde $\mathbf{w}$ é o vetor normal ao hiperplano e ao se variar b obtemos uma família de hiperplanos paralelos (Figura 2.9).

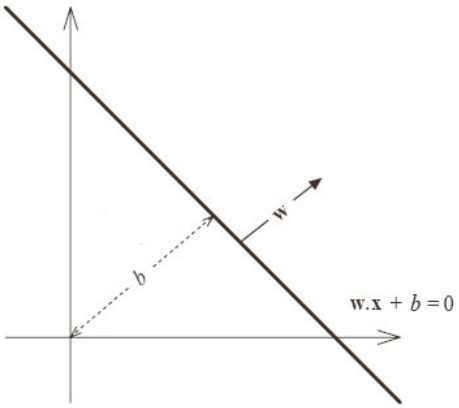


Figura 2.9: Hiperplano separador

A *margem M* do hiperplano separador é definida como sendo a distância entre o hiperplano e o ponto mais próximo da classe positiva somado à distância do hiperplano ao ponto mais próximo da classe negativa como é mostrado na Figura 2.10.

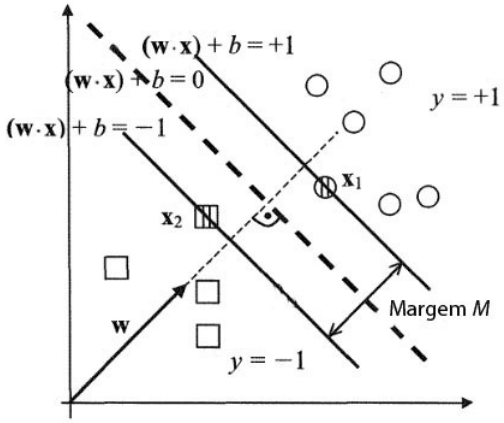


Figura 2.10: Margem do hiperplano separador

O *hiperplano separador ótimo* é o hiperplano que possui a maior margem para um dado conjunto finito de padrões de aprendizado. É ele que é computado na fase de treinamento pelo algoritmo de vetor suporte.

Suponha que todos os dados de treinamento satisfaçam as seguintes restrições (escalando $\mathbf{w}$ e b se necessário):

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1 \qquad \text{para } y_i = +1 \tag{2.4}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \qquad \text{para } y_i = -1 \tag{2.5}$$

Estas restrições podem ser combinadas em um único conjunto de desigualdades:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \forall i = 1, ..., l \tag{2.6}$$

Agora, considere os pontos que satisfazem a igualdade na Equação 2.4. Estes pontos passam no hiperplano $H_1 : \mathbf{w} \cdot \mathbf{x}_i + b = +1$. Similarmente, os pontos que satisfazem a igualdade na Equação 2.5 passam no hiperplano $H_2 : \mathbf{w} \cdot \mathbf{x}_i + b = -1$.

Sejam $\mathbf{x}_1$ e $\mathbf{x}_2$ dois pontos pertencendo aos hiperplanos H_1 e H_2, respectivamente, conforme mostrado na Figura 2.11. A margem M que é maximizada durante o estágio de treinamento pode ser vista como uma projeção do vetor $(\mathbf{x}_1 - \mathbf{x}_2)$ na direção do vetor normal do hiperplano separador.

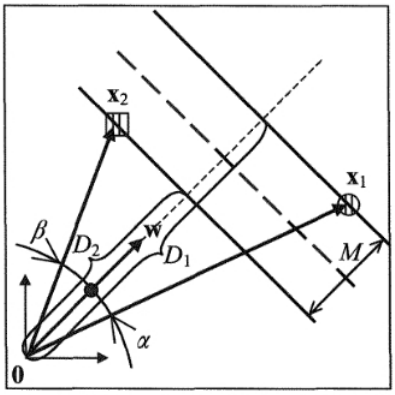


Figura 2.11: Cálculo da margem máxima

Podemos então calcular a margem M da seguinte forma:

$$D_1 = \|\mathbf{x_1}\| \cos(\alpha), D_2 = \|\mathbf{x_2}\| \cos(\beta), \tag{2.7}$$

$$M = D_1 - D_2, \tag{2.8}$$

onde α e β são os ângulos entre $\mathbf{w}$ e $\mathbf{x_1}$ e entre $\mathbf{w}$ e $\mathbf{x_2}$, respectivamente dados por

$$\cos(\alpha) = \frac{\mathbf{x}_1 \cdot \mathbf{w}}{\|\mathbf{x}_1\| \|\mathbf{w}\|} \quad \text{e} \quad \cos(\beta) = \frac{\mathbf{x}_2 \cdot \mathbf{w}}{\|\mathbf{x}_2\| \|\mathbf{w}\|}. \tag{2.9}$$

Substituindo 2.9 em 2.7 e 2.8 resulta em

$$M = \frac{\mathbf{x}_1 \cdot \mathbf{w} - \mathbf{x}_2 \cdot \mathbf{w}}{\|\mathbf{w}\|}, \tag{2.10}$$

e usando o fato que $\mathbf{x}_1$ e $\mathbf{x}_2$ pertencem, respectivamente, aos hiperplanos $\mathbf{w} \cdot \mathbf{x}_1 + b = 1$ e $\mathbf{w} \cdot \mathbf{x}_2 + b = -1$, finalmente obtemos

$$M = \frac{2}{\|\mathbf{w}\|}. \tag{2.11}$$

Assim, para maximizar a margem M, devemos minimizar a norma do vetor normal ao hiperplano, dada por $\|\mathbf{w}\| = \sqrt{\mathbf{w} \cdot \mathbf{w}} = \sqrt{w_1^2 + w_2^2 + ... + w_n^2}$.

Logo, para achar o hiperplano que possui a margem máxima, devemos resolver o seguinte problema de otimização:

$$\begin{aligned} &\text{minimizar} && \tfrac{1}{2}\|\mathbf{w}\|^2 \\ &\text{sujeito a} && y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0 \quad i = 1, ..., l \end{aligned} \tag{2.12}$$

O fator $1/2$ é usado por conveniência matemática.

Este é um problema clássico de *programação quadrática com restrições lineares* que pode ser resolvido através do seu primal e/ou dual lagrangiano.

Formulação de Lagrange

Nesta seção será feita a formulação de Lagrange do problema. Há duas razões para se fazer isso: 1) as restrições em 2.12 serão trocadas por restrições nos próprios multiplicadores de Lagrange, mais fácil de manipular e 2) nesta formulação do problema, os dados de treinamento irão aparecer somente na forma de produtos internos entre vetores. Esta é uma propriedade crucial que irá permitir generalizar o procedimento no caso das SVMs não-lineares.

Para formar o lagrangiano, nas restrições da forma $c_i \geq 0$, as desigualdades das equações de restrição são multiplicadas por *multiplicadores de Lagrange* (também chamados de *variáveis duais*) não-negativos e são subtraídas da função objetivo. Para as restrições de igualdade, os multiplicadores de Lagrange são não-restritos [10].

Logo, introduzindo multiplicadores de Lagrange $\alpha_i \geq 0$, $i = 1, ..., l$, nas restrições de 2.12, nos dá o seguinte lagrangiano primal:

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i (\mathbf{w} \cdot \mathbf{x}_i + b) + \sum_{i=1}^{l} \alpha_i \tag{2.13}$$

O tratamento lagrangiano para problemas de otimização convexo leva a uma descrição dual alternativa, que frequentemente torna-se mais fácil de resolver do que o problema primal uma vez que a manipulação direta das restrições de desigualdades é difícil [15].

Assim, o lagrangiano L_P deve ser minimizado com respeito às *variáveis primais* $\mathbf{w}$ e b e maximizado com respeito às *variáveis duais* α_i. Para isso, utilizamos as condições de KARUSH-KUHN-TUCKER (KKT) para o ótimo das funções restritas [28]. Neste caso, tanto a função objetivo quanto as restrições são convexas, e as condições KKT são necessárias e suficientes para um máximo em 2.13. Estas condições para o problema primal são as seguintes:

Condições de gradiente:

$$\frac{\partial}{\partial \mathbf{w}} L_P = 0 \quad \text{e} \quad \frac{\partial}{\partial b} L_P = 0, \tag{2.14}$$

o que leva a

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i \tag{2.15}$$

e

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{2.16}$$

Condição de complementariedade:

$$\alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0, \, i = 1, ..., l, \tag{2.17}$$

Condição de viabilidade:

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \, i = 1, ..., l, \tag{2.18}$$

Condição de não-negatividade:

$$\alpha_i \geq 0, \, i = 1, ..., l \tag{2.19}$$

Substituindo 2.15 e 2.16 em 2.13, obtemos o lagrangiano dual:

$$L_D = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{2.20}$$

Para achar o hiperplano ótimo, o lagrangiano dual L_D deve ser maximizado com respeito aos α_i e às restrições 2.16. O problema dual pode ser formulado da seguinte forma

$$\begin{aligned} &\text{maximizar} \quad L_D = \sum_{i=1}^{l} \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ &\text{sujeito a} \qquad\qquad \alpha_i \geq 0 \qquad\qquad i = 1, ..., l \end{aligned} \tag{2.21}$$

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

O lagrangiano dual L_D é expresso em termos dos dados de treinamento e depende somente dos produtos internos dos padrões de entrada $(\mathbf{x}_i \cdot \mathbf{x}_j)$. Este produto interno expressa a similaridade de dois pontos de treinamento em termos dos atributos dados. Esta propriedade de L_D será bastante útil mais tarde para o caso geral de SVM não-linear.

Existe um multiplicador de Lagrange α_i para cada ponto de treinamento. Na solução, os pontos onde $\alpha_i > 0$ são chamados *vetores suporte*, e situam-se em um dos hiperplanos H_1 ou H_2 definidos na seção anterior. Todos os outros pontos de treinamento são irrelevantes: suas restrições são automaticamente satisfeitas, e eles não aparecem em 2.15 uma vez que seus multiplicadores $\alpha_i = 0$.

Para as SVMs, os vetores suporte são os elementos críticos do conjunto de treinamento. Eles passam o mais próximo da fronteira de decisão; se todos os outros pontos de treinamento forem removidos (ou movidos, desde que não cruzem H_1 ou H_2), e o treinamento for repetido, o mesmo hiperplano separador deve ser encontrado.

O hiperplano separador ótimo $(\mathbf{w}, b)$ é definido pelo vetor $\mathbf{w}$ dada pela Equação 2.15 e o limiar b. Para calcular b, usamos a condição KKT da Equação 2.17, a expressão de $\mathbf{w}$ e a condição $\alpha_j > 0$, o que leva a

$$\sum_{i}^{l} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x}_j + b = y_j$$

Portanto, o limiar b pode ser obtido fazendo-se a média dos valores

$$b = y_j - \sum_{i}^{l} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x}_j$$

para todos os pontos onde $\alpha_j > 0$, ou seja, todos os vetores suporte.

Para classificar um novo padrão $\mathbf{x}$ usamos uma função sinal, denominada *função de decisão* dada por:

$$f(\mathbf{x}) = \text{sgn}(\sum_{i}^{Ns} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b) \tag{2.22}$$

onde Ns é o número de vetores suporte.

Padrões que não são vetores suporte ($\alpha_i = 0$) não influenciam na classificação de novos padrões. A função de decisão 2.22 tem uma importante vantagem pois para classificar um novo padrão $\mathbf{x}$, é apenas necessário calcular o produto interno entre $\mathbf{x}$ e todos os vetores suporte. Isto resulta em uma significante economia de tempo computacional uma vez que o número de vetores suporte é pequeno comparado com o número total de exemplos no conjunto de treinamento.

2.4.1.2 Classificação de dados não-linearmente separáveis

O procedimento de aprendizado apresentado na seção anterior é válido para dados linearmente separáveis, isto é, para conjuntos de dados de treinamento sem sobreposição. Tais problemas são raros na prática. Por outro lado, existem muitas instâncias onde hiperplanos separadores lineares podem ter boas soluções mesmo quando dados estão sobrepostos.

Contudo, a solução de programação quadrática apresentada anteriormente não pode ser usada no caso de sobreposição devido às restrições $y_i(\mathbf{w}\cdot\mathbf{x}_i+b)-1\geq 0, i=1,...,l$ não poderem ser satisfeitas. Para achar um classificador com uma margem máxima, este algoritmo deve ser mudado, permitindo alguns dados serem mal classificados, ou estarem no lado "errado" da fronteira de decisão [41].

Na prática nós permitimos uma *"margem flexível"* onde todos os dados dentro desta margem são negligenciados. A largura desta margem pode ser controlada por um parâmetro de penalidade que determina o compromisso entre a maximização da margem e a minimização do erro de treinamento. A Figura 2.12 mostra dois pontos, x_1 e x_2, dentro da margem, ou seja, estão mal classificados.

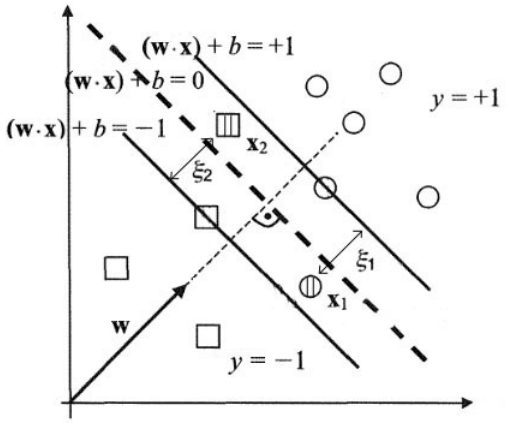


Figura 2.12: Dois pontos mal classificados dentro da margem

O algoritmo de margem ótima é generalizado para problemas não-separáveis [14] pela introdução de variáveis de folga não-negativas $\xi_i, i=1,...,l$, nas restrições 2.6. Agora, o hiperplano separador deverá satisfazer

$$\begin{aligned}\mathbf{w}\cdot\mathbf{x}_i+b &\geq +1-\xi_i \qquad \text{para } y_i=+1\\ \mathbf{w}\cdot\mathbf{x}_i+b &\leq -1+\xi_i \qquad \text{para } y_i=-1\\ \xi_i &\geq 0 \qquad \forall i\end{aligned}$$

ou

$$y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i,\ i=1,...,l,\ \xi_i\geq 0, \tag{2.23}$$

A função objetivo a ser minimizada irá incluir também um termo extra computando o custo dos erros de sobreposição. Desse modo, a nova forma do problema de otimização é dada por:

$$\begin{aligned} \text{minimizar} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \\ \text{sujeito a} \quad & y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i \quad i=1,...,l \\ & \xi_i\geq 0 \qquad\qquad \forall i \end{aligned} \tag{2.24}$$

O parâmetro C é escolhido pelo usuário, um C grande corresponde a assinalar penalidade maior para os erros, minimizando o número de padrões mal classificados. Um C pequeno maximiza a margem tal que o hiperplano separador é menos sensível aos erros no conjunto de treinamento.

O problema acima também é um problema quadrático de otimização e como no problema linearmente separável, introduzimos multiplicadores de Lagrange para as restrições e obtemos o seguinte lagrangiano primal

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i+b)-1+\xi_i) - \sum_{i=1}^{l}\beta_i\xi_i \tag{2.25}$$

onde α_i e β_i são multiplicadores de Lagrange para as restrições 2.23.

Como antes, uma solução no espaço dual é achada usando as condições KKT [28] como segue:

Condições de gradiente:

$$\frac{\partial}{\partial\mathbf{w}}L_P = 0 \ \Rightarrow\ \mathbf{w}=\sum_{i=1}^{l}\alpha_i y_i\mathbf{x}_i \tag{2.26}$$

$$\frac{\partial}{\partial b}L_P = 0 \ \Rightarrow\ \sum_{i=1}^{l}\alpha_i y_i = 0 \tag{2.27}$$

$$\frac{\partial}{\partial\xi_i}L_P = 0 \ \Rightarrow\ \alpha_i+\beta_i = C \tag{2.28}$$

Condição de complementariedade:

$$\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i+b)-1+\xi_i)=0,\ i=1,...,l \tag{2.29}$$

Condição de viabilidade:

$$y_i(\mathbf{w}\cdot\mathbf{x}_i+b)-1+\xi_i=0,\ i=1,...,l \tag{2.30}$$

Condição de não-negatividade:

$$\xi_i\geq 0,\ \alpha_i\geq 0,\ \beta_i\geq 0,\ \beta_i\xi_i=0,\ i=1,...,l \tag{2.31}$$

Substituindo 2.26 e 2.27 em 2.25, obtemos o lagrangiano dual:

$$L_D=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j\mathbf{x}_i\cdot\mathbf{x}_j \tag{2.32}$$

e problema dual pode ser formulado como

$$\begin{aligned}\text{maximizar}\quad & L_D=\sum_{i=1}^{l}\alpha_i-\tfrac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j\mathbf{x}_i\cdot\mathbf{x}_j \\ \text{sujeito a}\quad & 0\leq\alpha_i\leq C \qquad i=1,...,l \\ & \sum_{i=1}^{l}\alpha_i y_i=0\end{aligned} \tag{2.33}$$

Logo, o problema de otimização quadrático final é praticamente o mesmo que no caso separável, a única diferença é que os multiplicadores de Lagrange α_i agora tem um limite superior dado pelo parâmetro de penalização C, que é determinado pelo usuário.

Mais uma vez, o hiperplano separador é dado por $(\mathbf{w},b)$, onde

$$\mathbf{w}=\sum_{i=1}^{l}\alpha_i y_i\mathbf{x}_i \tag{2.34}$$

e, como antes, podemos utilizar as condições KKT, equações 2.29 e 2.31, para determinar o valor de b.

Finalmente, a expressão para a função de decisão $f(\mathbf{x})$ para o classificador com margem flexível, dada por 2.35, é a mesma para classes linearmente separáveis

$$f(\mathbf{x})=\text{sgn}(\sum_{i}^{Ns}\alpha_i y_i\mathbf{x}_i\cdot\mathbf{x}+b) \tag{2.35}$$

onde *Ns* é o número de vetores suporte.

2.4.2 SVMs não-lineares

Os métodos acima podem ser generalizados para o caso onde a função de decisão não é uma função linear dos dados. A extensão de SVM para conjuntos de dados não-lineares é

baseada no mapeamento das variáveis de entrada em um espaço de dimensão maior onde é realizada a classificação linear.

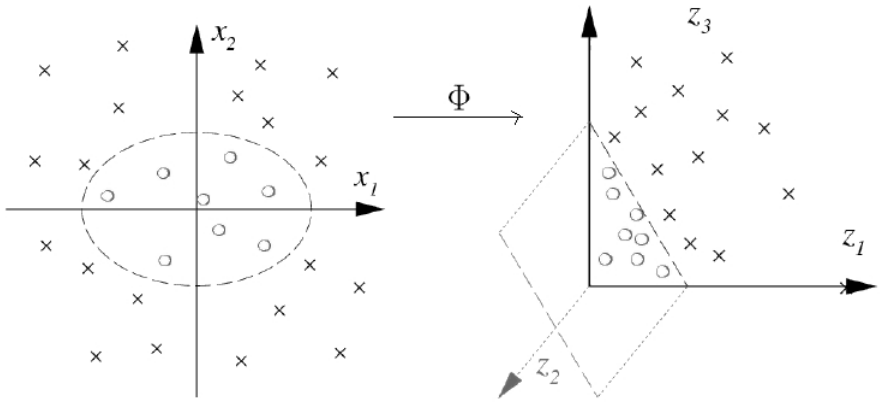


Figura 2.13: Separação linear de padrões no espaço de características

Por exemplo, considere o conjunto de padrões não-linearmente separáveis do lado esquerdo da Figura 2.13. Um classificador linear não é capaz de fazer a separação das classes neste espaço. Entretanto, se primeiro mapeamos os dados para outro espaço euclideano $\mathscr{H}$ de dimensão maior, usando uma função

$$\begin{array}{rccc} \Phi: & \mathbb{R}^n & \longrightarrow & \mathscr{H} \\ & \mathbf{x} & \longmapsto & \Phi(\mathbf{x}) \end{array}$$

tal que as duas classes passam a ser separáveis neste espaço $\mathscr{H}$, o uso do classificador linear fica permitido (lado direito da Figura 2.13). O espaço $\mathscr{H}$ é chamado de *espaço de características*.

Uma propriedade importante que a função Φ deve possuir é que a imagem do conjunto de treino dado por $\Phi(T)$ deve ser linearmente separável no espaço de características, mesmo quando o conjunto não é linearmente separável no espaço original. Esta propriedade é valiosa pois classificadores lineares são fáceis de computar, e todas as considerações nas seções anteriores se mantém, uma vez que ainda estamos fazendo uma separação linear, mas em um espaço diferente.

Além disso, nos classificadores SVM lineares apresentados, o único modo em que os dados aparecem no problema de treinamento é na forma de produtos internos, $\mathbf{x}_i \cdot \mathbf{x}_j$. Através da função Φ, o algoritmo de treinamento depende apenas dos dados através do produto interno em $\mathscr{H}$, isto é, em funções da forma $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. Esta propriedade dos classificadores não-lineares é importante pois mostra que nem precisamos conhecer a expressão analítica da função Φ.

BOSER *et al.* [5] mostraram que um "truque" um tanto antigo [1] pode ser usado de uma maneira relativamente direta para criar fronteiras de decisão não-lineares: uma

classe especial de funções, chamadas *kernels*, permite o cálculo do produto interno no espaço original definido pelos padrões de treinamento ao invés do espaço de características (Equação 2.36).

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j). \tag{2.36}$$

O *kernel K* deve ser uma função definida positiva simétrica, que satisfaça as condições de MERCER [73]:

$$K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x})$$

$$K^2(\mathbf{x}, \mathbf{y}) \leq K(\mathbf{x}, \mathbf{x}) K(\mathbf{y}, \mathbf{y})$$

$$\sum_{i=1}^{l} \sum_{j=1}^{l} \lambda_i \lambda_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0, \forall \mathbf{x}_l, \forall \lambda_l \in \mathbb{R}$$

Na Tabela 2.2 são apresentados os principais *kernels* para o reconhecimento de padrões. Estes *kernels* são os mais utilizados também nas principais implementações de SVM. O primeiro resulta em um classificador que é um polinômio de grau d nos dados, o segundo gera um classificador de função gaussiana de base radial, e o terceiro gera um tipo particular de uma rede neuronal sigmoidal de duas camadas. Um estudo completo sobre *kernels* e toda teoria subjacente ao assunto pode ser encontrada em [68].

Tabela 2.2: *Kernels* mais utilizados

Polinomial	$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d$
Gaussiano	$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$
Sigmóide	$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x} \cdot \mathbf{y} + c)$

Desta forma, problemas de classificação não-linear podem ser resolvidos com sucesso aplicando uma das funções de *kernel* no espaço de entrada que substitui o cálculo do produto interno requerido no espaço de características. Não precisamos nem definir explicitamente a função Φ.

O algoritmo de treinamento para SVM não-linear segue da construção de um hiperplano separador ótimo no espaço de características. A formulação matemática é similar àquela apresentada para a SVM linear, onde agora apenas os padrões de entrada $\mathbf{x}$ são trocados por funções $\Phi(\mathbf{x})$, e o produto interno para duas funções $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ é trocada pela função *kernel* $K(\mathbf{x}_i, \mathbf{x}_j)$.

No espaço de características $\mathscr{H}$, o lagrangiano dual, dado em 2.20 e 2.32, é

$$L_D = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

A partir da escolha de um *kernel* e de acordo com 2.36, o problema de otimização a ser resolvido fica

$$\begin{aligned} \text{maximizar} \quad & L_D = \sum_{i=1}^{l} \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{sujeito a} \quad & \alpha_i \geq 0 \qquad\qquad i = 1, ..., l \\ & \sum_{i=1}^{l} \alpha_i y_i = 0 \end{aligned}$$

No caso mais geral, considerando que os dados podem estar sobrepostos devido a ruídos ou artefatos da base de dados, podemos usar o classificador de margem flexível onde a única diferença é o limite superior C dos multiplicadores de Lagrange α_i, tornando esta restrição como

$$0 \leq \alpha_i \leq C, \, i = 1, ..., l$$

O hiperplano separador irá definir um classificador linear no espaço de características $\mathscr{H}$, e criar uma *hipersuperfície* de separação não-linear no espaço original de entrada.

A função de decisão é dada por 2.37:

$$f(x) = \text{sgn}(\sum_{i=1}^{Ns} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b) \tag{2.37}$$

Mais uma vez, a soma não é efetuada sobre por todos os dados de treinamento mas apenas sobre os vetores suporte pois somente para eles os multiplicadores de Lagrange diferem de zero.

Os parâmetros do hiperplano de margem máxima são derivados pela solução do problema de otimização. Existem vários algoritmos especializados para resolver rapidamente o problema de programação quadrática que surge das SVMs, a maioria baseado em heurísticas para quebrar o problema em pedaços menores e mais tratáveis.

Um método comum para resolver o problema de programação quadrática é o algoritmo *Sequential Minimal Optimization* (SMO) de PLATT [56], que quebra o problema em sub-problemas de duas dimensões que podem ser resolvidos analiticamente, eliminando a necessidade de um algoritmo de otimização numérica tal como métodos de gradiente conjugado. O pseudocódigo deste algoritmo pode ser encontrado no apêndice A de [15]. Outros métodos populares que também funcionam fazendo a decomposição do problema

são SVM-Light [39] e LIBSVM [13].

JOACHIMS [40] apresenta um algoritmo de plano de corte para o treinamento de SVMs chamado SVM-Perf. Ele é baseado numa formulação alternativa do problema de otimização de SVM e mostrou-se ser muito mais rápido que os métodos tradicionais de decomposição para bases de dados grandes.

2.5 Medidas de desempenho de um classificador

Para finalizar este capítulo de fundamentos, apresentamos nesta seção as principais medidas de desempenho consideradas para um classificador de duas classes, foco deste trabalho.

As métricas mais comumente utilizadas são baseadas na *matriz de confusão* mostrada na Tabela 2.3. Cada linha da matriz representa exemplos na classe predita, enquanto que cada coluna representa exemplos na classe real. As classes são denominadas positiva e negativa. Os valores da matriz são:

tp (*true positives* - positivos verdadeiros) é o número de exemplos positivos classificados corretamente

tn (*true negatives* - negativos verdadeiros) é o número de exemplos negativos classificados corretamente

fp (*false positives* - falso-positivos) o número de exemplos negativos classificados incorretamente como exemplos positivos

fn (*false negatives* - falso-negativos) o número de exemplos positivos incorretamente classificados como exemplos negativos

O total de exemplos classificados é a soma: $tp+fn+fp+tn$

Tabela 2.3: Matriz de confusão

Exemplos	Realmente Positivos	Realmente Negativos
Positivos preditos	tp	fp
Negativos preditos	fn	tn
Total	$tp+fn$	$fp+tn$

Uma das vantagens da matriz de confusão é que é fácil visualizar quando o classificador está confundindo duas classes, isto é, classificando exemplos de uma classe em outra.

As medidas de desempenho mais comuns calculadas a partir da matriz de confusão são:

Acurácia (padrão): mede a taxa de acertos de um classificador. É a razão do total de exemplos positivos e negativos classificados corretamente pelo total de exemplos. É dada por:

$$\frac{tp+tn}{totalExemplos}$$

Acurácia balanceada: mede a taxa de acertos de um classificador considerando o desbalanceamento na quantidade de exemplos positivos e negativos. Calcula a média entre sensibilidade e especificidade. É dada por:

$$\frac{1}{2}(\frac{tp}{tp+fn}+\frac{tn}{tn+fp})$$

***Recall* (sensibilidade):** mede o quanto um classificador pode reconhecer exemplos positivos, ou seja, quantos exemplos positivos são classificados corretamente dentre todos os exemplos positivos. É dada por:

$$\frac{tp}{tp+fn}$$

***Precision* (precisão):** mede o quanto um classificador pode reconhecer exemplos positivos que realmente são positivos, ou seja, quantos exemplos positivos são classificados corretamente dentre os exemplos classificados como positivos. É dada por:

$$\frac{tp}{tp+fp}$$

***Specificity* (especificidade):** mede o quanto um classificador pode reconhecer exemplos negativos, ou seja, é a taxa de acerto de exemplos negativos. É dada por:

$$\frac{tn}{tn+fp}$$

***F-measure*:** medida que leva em conta ambos *Recall* e *Precision*. É dada por:

$$\frac{2*Recall*Precision}{Recall+Precision}$$

Capítulo 3

Ensemble de cláusulas

3.1 Introdução

Ao construir uma teoria utilizando o algoritmo ILP padrão, o conjunto de exemplos positivos do conjunto de treinamento é manipulado de modo que cada cláusula a ser gerada considera um subconjunto de exemplos. No algoritmo de cobertura-remoção (*cover-removal*) os exemplos considerados como semente para gerar uma cláusula são cada vez mais exíguos. À medida que o algoritmo avança a probabilidade de encontrar cláusulas de cobertura alta diminui até o ponto onde aparecem cláusulas que cobrem somente um exemplo.

Além disso, a busca por uma teoria dentro do sistema ILP consome muito tempo e frequentemente produz classificadores excessivamente complexos (com muitas cláusulas) o que dificulta para a interpretação de um especialista.

Vários autores têm se interessado em usar técnicas de *ensembles* como alternativa para gerar melhores classificadores em ILP. O trabalho original nesta área é o de QUINLAN [59] no uso de *boosting* no sistema FOIL.

MORELLI [48] analisa o desempenho do método de *boosting* em ILP usando o sistema Aleph e introduz os *ensembles* de cláusulas.

Em SALVINI *et al.* [63] modificamos e estendemos o trabalho feito por MORELLI e comparamos *ensembles* de cláusulas com *ensembles* de teorias para gerar classificadores ILP usando o método de *bagging*.

Neste capítulo apresentamos um estudo empírico de métodos de *ensembles* usando ILP. Dois tipos de *ensembles* foram gerados:

1. *ensembles* cujos classificadores individuais são teorias, e
2. *ensembles* cujos classificadores individuais são cláusulas.

Argumentamos que aprender uma simples cláusula por vez ao invés de aprender teorias inteiras por vez, pode ser mais eficiente computacionalmente e produzir classificadores

mais simples e legíveis.

Achar uma teoria consome muito tempo e pode produzir classificadores muito complexos, no entanto, aprender simples cláusulas e usar métodos de *ensembles* para combiná-las poderia produzir classificadores melhores em muito menos tempo.

Consideramos neste trabalho uma teoria como um *classificador forte* (de alta acurácia) enquanto que uma única cláusula é considerada como sendo um *classificador fraco* (de baixa acurácia).

Resultados em DUTRA *et al.* [26] já mostravam que o benefício dos *ensembles* de teorias nas aplicações era limitado. Uma possível razão para isso é que uma teoria por si só já seria um *ensemble*. Um estudo em GOADRICH *et al.* [35] mostra que limitando o tamanho das teorias, ou seja, limitando o número de cláusulas da teoria, produz resultados melhores na área sob a curva de precisão-sensibilidade (*AUC precision-recall*) e aponta como limite 25 cláusulas por teoria.

Nas próximas seções apresentamos a metodologia e os resultados deste primeiro estudo.

3.2 Bases de dados

O conjunto de dados utilizados em nosso primeiro experimento é composto de três bases de dados que correspondem a aplicações não-triviais em ILP, que também são usadas em outros trabalhos[1]. Abaixo são descritas as características de cada base de dados e suas aplicações ILP associadas.

***Amine*:** Nossa primeira tarefa de aprendizado é predizer a inibição da recaptação de aminas para a descoberta de novas drogas para a doença de Alzheimer. São dados alguns exemplos positivos de drogas que são efetivas contra a doença, e alguns exemplos negativos. A tarefa é construir um modelo para uma droga ativa que distingue entre drogas ativas e inativas.

***Choline*:** Esta base de dados também está relacionada com a descoberta de drogas para o mal de Alzheimer. A tarefa de aprendizado neste caso é identificar a inibição da enzima *acetil-colina-esterase*.

***Protein*:** Nossa última base de dados consiste de um banco de dados de genes e características dos genes ou das proteínas para as quais eles codificam, junto com informação sobre quais proteínas interagem entre si e correlações entre padrões de expressão de genes. A tarefa de aprendizado focaliza na função "metabolismo" para predizer quais genes codificam proteínas envolvidas em metabolismo.

[1] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/applications.html

A Tabela 3.1 mostra um quadro resumo destas bases de dados. As colunas E^+, E^- e BK denotam, respectivamente, a quantidade de exemplos positivos, negativos e o tamanho do conhecimento prévio (medido como a quantidade de literais presentes em todos os exemplos).

Tabela 3.1: Quadro resumo dos dados das bases de dados utilizadas

Bases de dados	Total exemplos	E^+	E^-	BK
Amine	686	343	343	232
Choline	1326	663	663	232
Protein	862	172	690	6913

3.3 Metodologia

Para cada base de dados, primeiramente executamos o experimento para gerar o *ensemble* de teorias e depois o experimento para gerar o *ensemble* de cláusulas.

Foi empregada validação cruzada em 5 *folds* (5-*fold cross-validation*). Os conjuntos de treinamento e teste foram gerados usando *bagging* diretamente em cada *fold* independentemente.

Testamos a eficácia de diferentes tamanhos de *ensembles*. Em cada *fold* consideramos *ensembles* com tamanho variando de 1 a 25 no experimento onde cada classificador individual é uma teoria e, para o experimento onde o classificador individual é uma cláusula, variamos o tamanho do *ensemble* de 1 até o tamanho médio da teoria gerada no experimento anterior.

Foi usado o sistema ILP Aleph, visto no capítulo anterior, sendo executado no Yap Prolog [2]. Aleph é chamado uma vez para cada conjunto de treinamento.

Em todas as bases de dados, os parâmetros do Aleph utilizados foram:

- Número máximo de literais em uma cláusula (*clauselength*): 5
- Função de avaliação (*evalfn*): *coverage*. A utilidade da cláusula medida por esta função é dada por $(P-N)$, sendo P e N o número de exemplos positivos e negativos cobertos pela cláusula, respectivamente;
- Encadeamento das variáveis (*i*): 5. O valor padrão é 2, aumentando este valor obtemos relações mais complexas entre literais numa cláusula;
- Número máximo de nós permitidos (*nodes*): 100.000
- Valor mínimo da acurácia aceitável de uma cláusula (*minacc*): 0,9

[2] Disponível em http://www.ncc.up.pt/~vsc/Yap/

Foi usado o mesmo conjunto de parâmetros usados por DUTRA *et al.* [26] para comparar os resultados.

O Algoritmo 2 mostra como são gerados os *ensembles* para cada base de dados. Primeiramente é gerado o *ensemble* de teorias e só depois o *ensemble* de cláusulas para uma dada base de dados. Este algoritmo é executado em cada *fold* da validação cruzada.

Algoritmo 2 Algoritmo de treinamento dos *ensembles*

1. Se *ensemble* de teorias então
   numClassificadores = 25;
2. Senão (*ensemble* de cláusulas)
   numClassificadores = tamanho médio das teorias geradas no experimento com teorias;
3. Criar *numClassificadores* conjuntos de treino usando o método de *bagging;*
4. Para cada conjunto de treino criado faça
   4.1 Gerar classificador individual (teoria ou cláusula) através do Aleph.

A avaliação dos classificadores foi feita sobre os conjuntos de testes dos *folds*. A métrica utilizada para avaliar a qualidade do *ensemble* foi a acurácia padrão.

Utilizamos a técnica de votação com limiar, vista no capítulo anterior, para testar diferentes tamanhos de *ensembles*. Para cada *ensemble* os limiares variaram de 1 até o tamanho do *ensemble*.

Além disso, quando se trabalha com este tipo de votação, é necessário usar um conjunto de ajuste ou validação (*tuning set*), porque o limiar de votação usado no conjunto de teste é escolhido no conjunto de validação. Ou seja, no conjunto de teste não são usados todos os limiares de votação e é escolhido aquele que deu a melhor acurácia. Ao invés disso, aceita-se o melhor limiar de votação que vem do conjunto de validação para ser usado no conjunto de teste, mesmo que este não seja o melhor neste conjunto.

Devido aos resultados poderem ser distorcidos por uma escolha particular, ruim ou boa dos classificadores individuais, para cada tamanho de *ensemble*, repetimos este processo de seleção 30 vezes para que permita um resultado significativamente estatístico, e fazemos a média dos resultados.

O procedimento completo do processo de votação e avaliação é mostrado no Algoritmo 3. Este algoritmo é executado em cada *fold* da validação cruzada.

O Algoritmo 3 computa os pontos necessários para produzir a curva de acurácia, onde cada ponto é relativo a um tamanho de *ensemble*. Os passos 1.1.1 e 1.1.2 do algoritmo são repetidos 30 vezes onde, a cada iteração, os classificadores individuais (teorias ou cláusulas) são selecionados aleatoriamente dentre os que foram gerados. Isto constrói

Algoritmo 3 Algoritmo de avaliação dos *ensembles*

1. Para *tamEnsemble* de 1 até *numClassificadores* faça

   1.1 Para *limiar* de 1 até *tamEnsemble* faça

      1.1.1 Selecionar aleatoriamente *tamEnsemble* classificadores individuais;

      1.1.2 Para cada exemplo *e* do conjunto de teste faça

         1.1.2.1 Calcular
         votosPositivo = quantidade de classificadores individuais selecionados que consideram *e* como sendo da classe positiva;

         1.1.2.2 Se $votosPositivo \geq limiar$ então
         o exemplo *e* é considerado positivo pelo *ensemble*;

         1.1.2.3 Senão
         o exemplo *e* é considerado negativo pelo *ensemble*;

      1.1.3 Calcular
      $acuraciaMedia[tamEns, limiar]$.

uma tabela por *fold*, por tamanho de *ensemble*, onde cada linha representa a acurácia padrão média de 30 iterações, para cada limiar de votação.

Isto é repetido duas vezes, uma para o conjunto de validação (*tuning set*) de onde é extraído o melhor limiar para cada tamanho de *ensemble* e novamente na fase de treinamento onde o limiar de votação usado é aquele que foi o melhor na fase de validação. Os resultados finais são obtidos fazendo a média das acurácias através de todos os *folds*.

3.4 Resultados

As acurácias apresentadas representam a média calculada para todos os *folds* e são relacionadas aos conjuntos de teste.

Para cada tamanho de *ensemble*, valores de acurácia para os conjuntos de teste são obtidos variando o limiar de votação. O resultado traçado usa o limiar que produziu a melhor acurácia no conjunto de validação onde foram usados 3 dos 4 conjuntos de treinamento para cada *fold* para obter o melhor limiar.

As Figuras 3.1, 3.2 e 3.3 mostram as acurácias médias para cada tamanho de *ensemble* nas 3 aplicações utilizadas neste experimento. Nestas figuras são comparados o aprendizado dos *ensembles* de teorias e cláusulas.

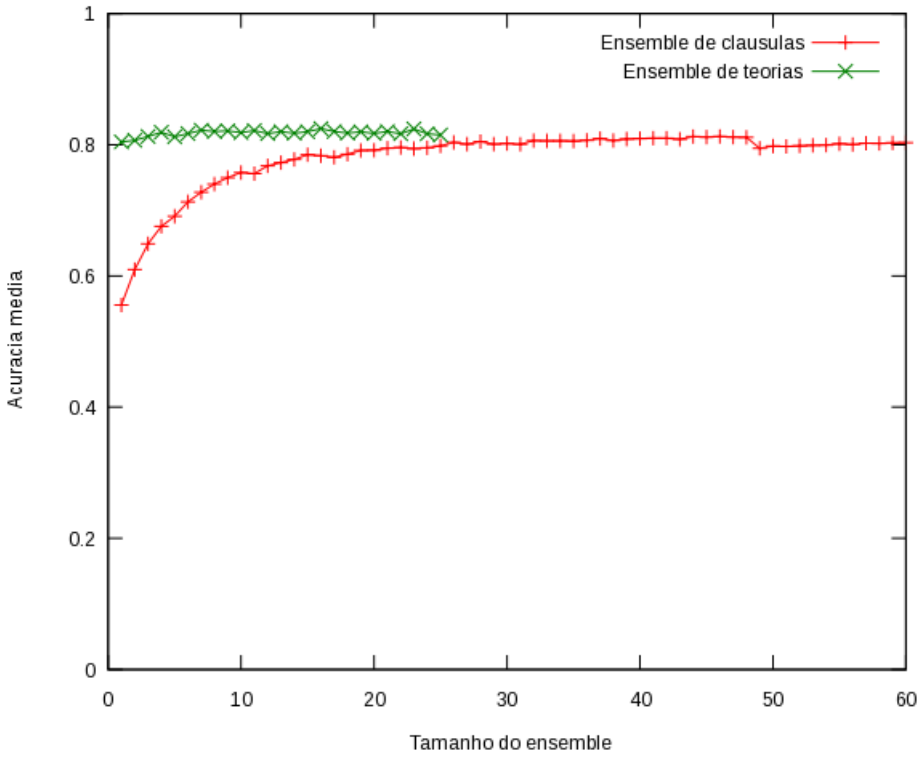


Figura 3.1: Acurácias médias para diferentes tamanhos de ensembles de teorias e cláusulas utilizando a base de dados *Amine*.

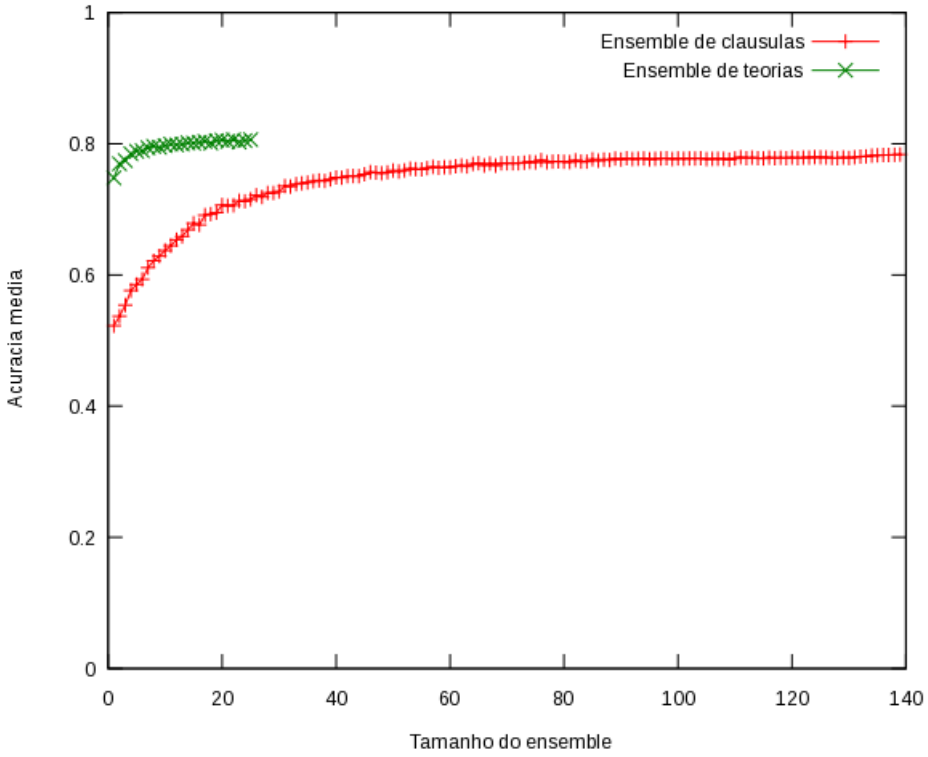


Figura 3.2: Acurácias médias para diferentes tamanhos de ensembles de teorias e cláusulas utilizando a base de dados *Choline*.

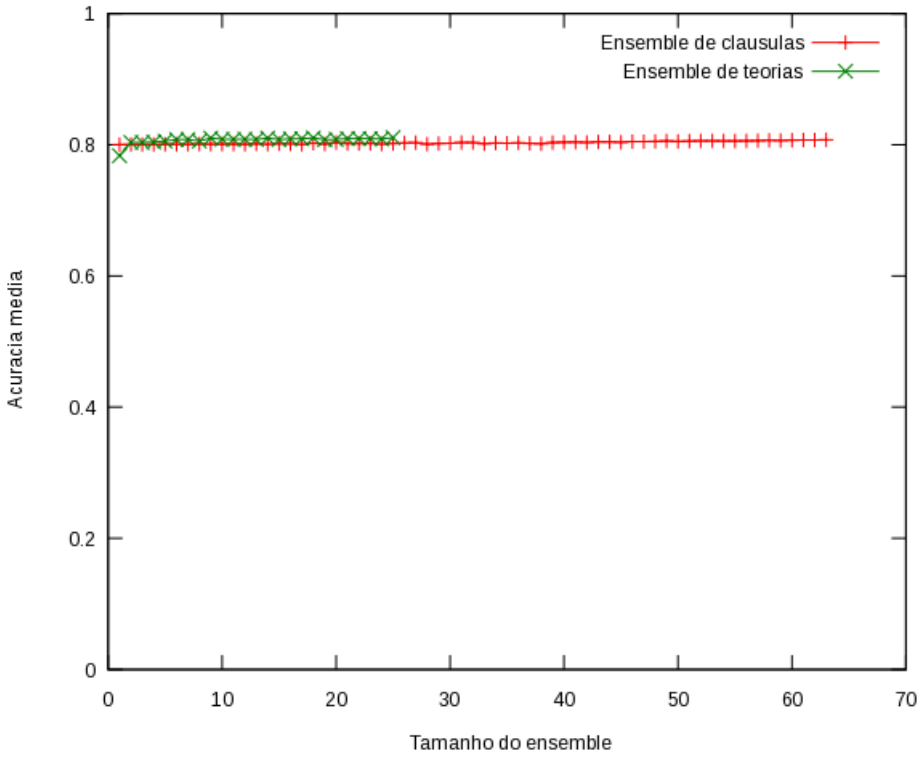


Figura 3.3: Acurácias médias para diferentes tamanhos de ensembles de teorias e cláusulas utilizando a base de dados *Protein.*

Com relação aos dois tipos de *ensembles* gerados podemos destacar:

- Apenas duas bases de dados, *Choline* (Figura 3.2) e *Protein* (Figura 3.3), tiveram uma modesta, mas não significante, melhora quando comparamos a acurácia de um *ensemble* de teorias de tamanho 1 com *ensembles* de tamanhos maiores. A base de dados *Amine* (Figura 3.1) não mostra praticamente nenhuma melhora quando nós compomos teorias em *ensembles* de tamanhos maiores.

- Em contraste, *ensembles* de cláusulas têm uma clara melhora em duas aplicações (*Amine* e *Choline*) à medida que aumentamos o tamanho deste tipo de *ensemble*, como é mostrado nas Figuras 3.1 e 3.2.

Os tamanhos dos *ensembles* de cláusulas variam de acordo com o tamanho médio das teorias geradas anteriormente pelos *ensembles* de teorias para cada aplicação. Estes valores são apresentados na Tabela 3.2.

Quando nós geramos o *ensemble* de teorias, o classificador pode se tornar muito complexo e difícil de interpretar pois o número total de cláusulas geradas pelo *ensemble* é igual ao número de cláusulas geradas por cada classificador individual (tamanho da teoria de cada classificador) multiplicado pelo número de classificadores individuais (tamanho do *ensemble*).

Tabela 3.2: Tamanho médio das teorias aprendidas no *ensemble* de teorias

Aplicação	Núm. Cláusulas
Amine	60
Choline	139
Protein	63

Por exemplo, na aplicação *Choline* (Figura 3.2) temos no total aproximadamente 3475 cláusulas no *ensemble* de teorias de tamanho 25. No caso do aprendizado baseado em cláusula, cada classificador individual é composto por apenas uma única cláusula. Logo, o número total de cláusulas geradas no *ensemble* de cláusulas de tamanho 139 é de apenas 139.

Na aplicação *Protein* (Figura 3.3) não houve melhora na acurácia com a utilização de *ensembles* de cláusulas de tamanhos maiores. Porém, podemos notar que apenas 2 cláusulas são suficientes (*ensemble* de cláusulas de tamanho 2) para produzir uma acurácia pouco pior do que um *ensemble* de teorias de tamanho 25 (onde cada teoria possui em média 63 cláusulas).

Da mesma forma, na aplicação *Amine* (Figura 3.1), para obter uma acurácia pouco melhor, um *ensemble* de teorias de tamanho 16 (com 60 cláusulas em média por teoria) produz uma acurácia pouco melhor que um *ensemble* de cláusulas de tamanho 46 com 46 cláusulas ao total. Além disso, há uma enorme diferença nos tempos gastos para gerar os *ensembles*: 3 min. para o *ensemble* de cláusulas contra 641 min. para o *ensemble* de teorias.

Nas Tabelas 3.3 e 3.4 são mostrados os tempos de execução e melhor acurácia alcançada nas três bases de dados nos dois tipos de *ensembles*. Os tempos de execução correspondem ao tempo total para produzir os *ensembles* de todos os tamanhos utilizando validação cruzada. Não estamos contando o tempo para gerar o classificador final através do mecanismo de votação. Isto adicionaria um tempo extra proporcional em todos os campos da tabela.

Podemos observar que as acurácias obtidas com *ensembles* de teorias são pouco melhores que as obtidas com *ensembles* de cláusulas, contudo, o tempo de execução para obter as teorias é muito maior. Por outro lado, podemos obter acurácias com *ensembles* de cláusulas muito próximas às melhores alcançadas pelo *ensemble* de teorias em muito menos tempo.

Tabela 3.3: Tabela com o tempo de execução, acurácia e total de cláusulas no *ensemble* de teorias

Bases	Tempo	Acurácia	Tam. *ens.*	Tam. teoria	Total cls.
Amine	641 min.	0,82	16	60	960
Choline	303 min.	0,81	25	139	3475
Protein	222 min.	0,81	25	63	1575

Tabela 3.4: Tabela com o tempo de execução, acurácia e total de cláusulas no *ensemble* de cláusulas

Bases	Tempo	Acurácia	Tam. *ens.*	Total cls.
Amine	3 min.	0,81	46	46
Choline	34 min.	0,78	139	139
Protein	24 min.	0,81	63	63

Os resultados demonstram vários fatos importantes:

- *Ensembles* de cláusulas com ILP são poderosos métodos para se obter boas acurácias num curto período de tempo;
- O método de *bagging* é eficaz para obter classificadores de boa qualidade, mesmo quando os classificadores individuais são fracos como simples cláusulas;
- Tamanhos de *ensembles* maiores que 25 podem produzir melhores acurácias para métodos que aprendem cláusulas;
- Teorias não se beneficiam muito dos *ensembles*. Possivelmente por uma teoria por si só já ser um ensemble;
- Classificadores muito fracos como simples cláusulas podem se beneficiar significativamente dos *ensembles* e produzir um classificador final que leva tempo de muitas ordens de magnitude menor que o tempo gasto para aprender teorias;
- Cláusulas são classificadores mais simples que teorias. Conseqüentemente um *ensemble* de cláusulas é mais simples que um *ensemble* de teorias, o que pode ajudar um especialista a interpretar melhor os resultados.

Todos os experimentos foram executados no mesmo computador com processador AMD Athlon 2025MHz com 1024MB de memória RAM utilizando o sistema operacional Mandriva Linux 2007 de 32 bits. Os programas foram desenvolvidos utilizando as linguagens Python (versão 2.4) e Prolog (Yap Prolog versão 5). O sistema ILP usado foi o Aleph (versão 3).

Capítulo 4

Classificador "Nata"

4.1 Introdução

Conforme visto no capítulo anterior, trabalhar com cláusulas geradas independentemente ao invés de teorias gera classificadores com desempenho comparável em muito menos tempo.

Na abordagem que propomos, cada exemplo do conjunto de treinamento é a base para construir uma cláusula. Deste modo incorporamos as seguintes vantagens com respeito da estratégia padrão:

1. todos os exemplos do conjunto de treinamento compartilham o mesmo espaço de busca para construir sua cláusula respectiva;

2. a cobertura e a acurácia da classificação de cada cláusula gerada podem ser usados como indicativos da contribuição relativa de conhecimento que cada exemplo possui;

3. é possível identificar aqueles exemplos que, devido à dificuldade em gerar uma cláusula a partir deles, representam casos a serem considerados separadamente.

Após gerar todas as cláusulas, eliminamos as cláusulas redundantes (sintática e semanticamente) e combinamos as cláusulas remanescentes por meio de uma disjunção.

Com isso, obtemos o menor conjunto de cláusulas que melhor modela o conjunto de exemplos, considerando como exceções aqueles que não foram generalizados. Chamamos este conjunto de cláusulas de *Nata*.

O nosso intuito é maximizar a cobertura de exemplos; entretanto, pode acontecer de uma cláusula que cubra poucos exemplos tenha uma acurácia (ou qualquer outra medida de desempenho) muito ruim. No entanto essa cláusula será incluída na teoria se estes poucos exemplos são cobertos somente por ela. Esta abordagem foi apresentada inicialmente em SALVINI *et al.* [62].

4.2 Bases de dados

Para avaliar esta nova metodologia acrescentamos, às três bases de dados anteriores, outras duas bases de dados clássicas utilizadas em ILP. São elas:

***Carcinogenesis*:** A tarefa ILP consiste na predição de testes de carcinogenicidade (capacidade de produzir câncer) em roedores. Esta base de dados tem várias características atrativas: é um importante problema prático; o conhecimento prévio consiste de um grande número de definições de predicados não-determinados; experiências anteriores sugerem que um espaço de busca muito grande precisa ser examinado para se obter uma boa cláusula.

***Mutagenesis*:** A determinação da capacidade de alguma substância causar mutação é importante devido à sua relevância para a compreensão e predição da formação do câncer. A tarefa ILP para esta base de dados consiste em predizer a capacidade de compostos aromáticos e hetero-aromáticos causar mutação.

A Tabela 4.1 mostra as características de todas as bases de dados utilizadas nos experimentos. As colunas E^+, E^- e BK denotam, respectivamente, a quantidade de exemplos positivos, negativos e o tamanho do conhecimento prévio (medido como a quantidade de literais presentes em todos os exemplos).

Tabela 4.1: Bases de dados utilizadas nos experimentos

Bases de dados	Total exemplos	E^+	E^-	BK
Amine	686	343	343	232
Carcinogenesis	330	182	148	24988
Choline	1326	663	663	232
Mutagenesis	188	125	63	15113
Protein	862	172	690	6913

4.3 Metodologia

Na nossa metodologia, utilizamos o conjunto total de exemplos de uma das classes da base de dados (por exemplo, os exemplos positivos) como sementes do processo de geração de cláusulas: geramos tantas cláusulas quanto exemplos da classe escolhida.

Porém, ao invés de trabalhar com o conjunto total de cláusulas geradas, usamos a "nata" das cláusulas, obtida através da eliminação das cláusulas redundantes ou equivalentes, que do ponto de vista lógico não acrescentam nada. Além disso, cláusulas especializadas que cobrem somente um único exemplo (o exemplo semente gerador da cláusula) também foram descartadas. O classificador resultante é constituído pela disjunção das cláusulas que sobraram após a depuração.

A seleção das cláusulas é feita através de uma *matriz de cobertura*. Cada linha desta matriz representa um exemplo da base de dados e cada coluna uma cláusula gerada. Se a cláusula cobrir um exemplo, a posição respectiva da matriz é preenchida com "1", caso contrário, é preenchida com "0".

Tabela 4.2: Exemplo de matriz de cobertura. As linhas representam os exemplos enquanto que as colunas representam as cláusulas geradas como os atributos dos exemplos. O valor "1" indica que o exemplo da linha i é coberto pela cláusula da coluna j.

	C_1	C_2	C_3	C_4
e_1^+	1	0	0	1
e_2^+	1	1	1	1
e_3^-	0	0	1	0
e_4^-	0	0	0	0

A Tabela 4.2 mostra um exemplo de matriz de cobertura. Nesta matriz está representada a cobertura de 4 exemplos por 4 cláusulas. Os 2 primeiros exemplos estão rotulados como sendo da classe positiva e os 2 últimos estão rotulados como sendo da classe negativa. O exemplo e_1^+ é coberto pelas cláusulas C_1 e C_4. O exemplo e_2^+ é coberto por todas as cláusulas. O exemplo e_3^- é coberto somente pela cláusula C_3. Já o exemplo e_4^- não é coberto por nenhuma cláusula.

Os passos utilizados para a geração do classificador Nata são apresentados no Algoritmo 4.

Algoritmo 4 Passos para a geração do classificador Nata

1. Para cada exemplo positivo e^+ da base de dados faça

   1.1 Gerar uma cláusula usando e^+ como semente;

2. Retirar do conjunto de cláusulas formado, aquelas que só cobrem um exemplo;
3. Retirar do conjunto de cláusulas formado, as cláusulas sintaticamente iguais;
4. Gerar a matriz de cobertura das cláusulas remanescentes considerando todos os exemplos (positivos e negativos);
5. Para cada cláusula $\bar{C}$ do conjunto de cláusulas faça

   5.1 Retirar toda cláusula C cuja cobertura esteja incluída na cobertura de $\bar{C}$;

6. Combinar as cláusulas remanescentes por meio de uma disjunção.

Os passos 1 a 3 do Algoritmo 4 foram implementados diretamente usando a linguagem Prolog e o sistema Aleph. Os passos 4 a 6 foram implementados usando a linguagem Python. As cláusulas foram selecionadas através do Algoritmo 5.

Algoritmo 5 Seleção das cláusulas Nata

1. Para cada par de cláusulas C_i e C_j da matriz de cobertura faça

   1.1 Calcular
   $|C_i|$ = quantidade de exemplos positivos cobertos pela cláusula C_i;
   $|C_j|$ = quantidade de exemplos positivos cobertos pela cláusula C_j;
   $C_i \cdot C_j$ = produto interno de C_i e C_j;

   1.2 Se $|C_i| = |C_j| = C_i \cdot C_j$ então
   C_i e C_j são equivalentes (com relação à cobertura de positivos) e é retirada aquela que cobre menos exemplos negativos;

   1.3 Senão se $|C_i| = C_i \cdot C_j$ então
   a cobertura de C_i está contida na de C_j e C_i é retirada;

   1.4 Senão se $|C_j| = C_i \cdot C_j$ então
   a cobertura de C_j está contida na de C_i e C_j é retirada;

   1.5 Senão
   nenhuma das duas cláusulas é retirada.

Para ilustrar o funcionamento desta implementação seguem alguns exemplos de cada situação que pode ocorrer. Considere as cláusulas C_1 e C_2 com suas respectivas coberturas de exemplos positivos, ou seja, colunas da matriz de cobertura apenas considerando a cobertura dos exemplos positivos.

Exemplo 1: $C_1 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$ e $C_2 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$

Assim, $|C_1| = 2$, $|C_2| = 2$, $C_1 \cdot C_2 = 2$

Logo, as cláusulas C_1 e C_2 são equivalentes e é retirada a que cobre menos exemplos negativos (não colocado no exemplo).

Exemplo 2: $C_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$ e $C_2 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$

Assim, $|C_1| = 1$, $|C_2| = 2$, $C_1 \cdot C_2 = 1$

Logo, a cláusula C_1 é retirada.

Exemplo 3: $C_1 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$ e $C_2 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$

Assim, $|C_1| = 3$, $|C_2| = 2$, $C_1 \cdot C_2 = 2$

Logo, a cláusula C_2 é retirada.

Exemplo 4: $C_1 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$ e $C_2 = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}^T$

Assim, $|C_1| = 2$, $|C_2| = 2$, $C_1 \cdot C_2 = 1$

Logo, nenhuma das cláusulas é retirada.

4.4 Resultados

Em todas as bases de dados as cláusulas foram geradas utilizando os mesmos valores de parâmetros do Aleph usados no capítulo anterior para que os resultados possam ser comparados.

Com o objetivo de mostrar a vantagem da abordagem Nata para descrever um domínio usando uma quantidade mínima de cláusulas, nosso primeiro experimento com esta abordagem foi aplicar o Algoritmo 4 usando todos os exemplos positivos das bases de dados para treinamento. Na Tabela 4.3 são apresentados os resultados obtidos. A acurácia obtida é a mesma quando consideramos o conjunto total de cláusulas geradas e o conjunto de cláusulas Nata uma vez que apenas as cláusulas que são redundantes foram retiradas.

Tabela 4.3: Tabela de resultados obtidos no conjunto de treinamento com as cláusulas geradas

Bases de dados	Cláusulas geradas	Tempo	Cláusulas Nata	Acurácia
Amine	343	5 min.	38	0,93
Carcinogenesis	182	52 min.	91	0,91
Choline	663	12 min.	50	0,83
Mutagenesis	125	47 min.	15	0,89
Protein	172	5 min.	69	0,94

A Tabela 4.3 sugere que um especialista procurando extrair conhecimento deverá considerar um conjunto de regras pelo menos 50% menor (base de dados *Carcinogenesis*), se for escolhida a abordagem Nata sendo que esta redução pode chegar a 92,5% (base de dados *Choline*).

A seguir são apresentados os resultados obtidos através de validação cruzada em 10 *folds*. Em cada *fold*, o Algoritmo 4 foi aplicado utilizando os exemplos positivos do conjunto de treino. As Tabelas 4.4, 4.5 e 4.6 comparam o classificador Nata com *ensembles* de cláusulas e teorias sob as perspectivas de acurácia do classificador, quantidade de cláusulas que formam o classificador e tempo para a geração do classificador. Os resultados apresentados representam a média calculada nos conjuntos de teste para todos os *folds* da validação cruzada.

Tabela 4.4: Comparação das acurácias médias obtidas na validação cruzada pelo classificador Nata e pelos métodos de *ensemble*

Bases de dados	Classificador Nata	*Ensemble* teorias	*Ensemble* cláusulas
Amine	0,84	0,82	0,81
Carcinogenesis	0,60	-	-
Choline	0,79	0,81	0,78
Mutagenesis	0,80	-	-
Protein	0,79	0,81	0,81

Tabela 4.5: Comparação da quantidade média de cláusulas consideradas na validação cruzada pelo classificador Nata e pelos métodos de *ensemble*

Bases de dados	Classificador Nata	*Ensemble* teorias	*Ensemble* cláusulas
Amine	36	960	46
Carcinogenesis	80	-	-
Choline	49	3475	139
Mutagenesis	15	-	-
Protein	62	1575	63

Tabela 4.6: Comparação dos tempos gastos para a geração do classificador Nata e os métodos de *ensemble*

Bases de dados	Classificador Nata	*Ensemble* teorias	*Ensemble* cláusulas
Amine	10 min.	641 min.	3 min.
Carcinogenesis	88 min.	-	-
Choline	28 min.	303 min.	34 min.
Mutagenesis	78 min.	-	-
Protein	11 min.	222 min.	24 min.

A Tabela 4.4 compara as acurácias dos classificadores. A abordagem Nata consegue melhores resultados que ambos os *ensembles* na base de dados *Amine* e no ensemble de cláusulas da base de dados *Choline* mas um resultado um pouco pior na base de dados *Protein*.

Além dos resultados de acurácia serem comparáveis, a quantidade média de cláusulas do classificador Nata também é sempre menor daquela correspondente aos *ensembles* (Tabela 4.5), o que reforça a afirmação de Nata ser uma metodologia adequada para um especialista.

O tempo total consumido para gerar todas as cláusulas e selecionar aquelas que compõem o classificador Nata é apresentado na Tabela 4.6. Apenas na base de dados *Amine* foi mais rápido gerar um *ensemble* de cláusulas do que as cláusulas Nata. Sendo que sempre são geradas todas as cláusulas em todos os *folds* da validação cruzada. Conforme já mencionado no capítulo anterior, o custo em tempo para gerar o *ensemble* de teorias é muito maior daquele consumido para gerar as cláusulas independentemente.

Capítulo 5

Classificador híbrido

5.1 Introdução

Trabalhos recentes mostram que uma abordagem eficaz é tratar cada cláusula aprendida como um atributo em um classificador proposicional que é quem vai determinar a classificação final dos exemplos.

A matriz de cobertura apresentada no capítulo anterior constitui uma representação numérica dos exemplos mapeados no espaço de cláusulas, onde cada cláusula representa uma dimensão para o exemplo, convertendo assim o problema em um problema proposicional. Esta representação alternativa permite aplicar classificadores proposicionais cujas características intrínsecas permitem fazer uma melhor generalização dos exemplos.

Classificadores externos à ILP, principalmente baseados em Redes de Bayes, como Naive Bayes e Tree Augmented Naive Bayes (TAN), têm sido usados com sucesso como parte da estrutura de aprendizado [16, 17, 18].

Esta metodologia define um processo em duas fases:

1. Um algoritmo ILP aprende um conjunto de cláusulas;

2. Um segundo classificador, do tipo proposicional, combina as classificações feitas pelas cláusulas aprendidas.

O primeiro trabalho usando esta metodologia é de POMPE e KNONENKO [57] onde foi aplicada uma Rede Naive Bayes para combinar cláusulas.

DAVIS *et al.* [16] utilizaram classificadores Bayesianos para combinar as classificações feitas pelas cláusulas; eles fizeram um estudo que compara as técnicas probabilísticas com o método de votação. Os autores mostraram que Naive Bayes, e principalmente Tree Augmented Naive Bayes (TAN), levam vantagem em relação ao método de votação na manipulação de possíveis regras imprecisas geradas. Métodos Bayesianos associam uma probabilidade para cada predição; esta probabilidade representa o grau de confiança do classificador individual (cláusula) na classificação final.

Neste capítulo apresentamos a abordagem, denominada de *híbrida*, que utiliza SVM para combinar as classificações feitas pelas cláusulas. SVM foi utilizada por ser um algoritmo de aprendizado que gera um classificador binário ótimo.

Esta abordagem é semelhante à de MUGGLETON *et al.* [51]. Naquele trabalho, cláusulas são geradas usando o sistema ILP CProgol. Todas as cláusulas com medida de compressão positiva são selecionadas de todos os *folds* e este conjunto é então transformado em uma matriz de cobertura que, por sua vez, serve como entrada para um sistema SVM. Esta abordagem é semelhante à que utilizamos, porém a metodologia aplicada é diferente. Os autores sugerem uma metodologia, onde fazem validação cruzada antes de submeter as cláusulas ao classificador SVM. Além disso, propõem selecionar as cláusulas com compressão positiva onde é selecionado um número grande de claúsulas comparado à nossa metodologia. A abordagem híbrida gera todas as cláusulas possíveis, geradas a partir de cada exemplo selecionado como semente para o sistema ILP Aleph e só então fornece estas cláusulas ao classificador proposicional. Esta metodologia claramente apresenta vantagens, pois o classificador proposicional fica com maior flexibilidade para gerar um classificador mais forte, já que os nossos classificadores são mais fracos do que os utilizados por MUGGLETON *et al.*

Nossa abordagem foi testada nas cinco bases de dados anteriores e mais uma base de dados relacional real de mamografias, extraída do "*National Mammography Database*" (NMD) americano em um trabalho conjunto com a Universidade de Wisconsin-Madison.

5.2 Metodologia

A abordagem híbrida que propomos utiliza SVMs para combinar as decisões das cláusulas individuais como alternativa à disjunção. O Algoritmo 6 descreve esta abordagem.

Algoritmo 6 Algoritmo de aprendizado para gerar classificadores híbridos.

1. Para cada exemplo positivo e^+ da base de dados faça

   1.1 Gerar uma cláusula usando e^+ como semente;

2. Retirar do conjunto de cláusulas formado, as cláusulas sintaticamente iguais;
3. Gerar a matriz de cobertura das cláusulas remanescentes considerando todos os exemplos (positivos e negativos);
4. Utilizar a matriz de cobertura para treinar um classificador SVM.

Os passos 1 e 2 do Algoritmo 6 descrevem como gerar e selecionar as cláusulas. A partir destas cláusulas o passo 3 do algoritmo gera a matriz de cobertura para todos os

exemplos. As linhas desta matriz representam os pontos que serão usados pelo algoritmo de aprendizado responsável pela criação do classificador SVM (passo 4).

Diferentemente da abordagem Nata, nesta abordagem mantemos as cláusulas que cobrem somente um exemplo e as cláusulas cuja cobertura já esteja abrangida por outras cláusulas, ou seja, apenas as cláusulas repetidas foram retiradas. Estas cláusulas que mantemos podem acrescentar dimensões que tornam o exemplo mais representativo para o algoritmo de aprendizado do SVM.

Para classificar um novo exemplo, primeiramente precisamos encontrar a sua representação no espaço de cláusulas. Esta representação é formada pela classificação que cada uma das cláusulas dá para o exemplo. Estas classificações são combinadas para formar um vetor binário que constitui a representação numérica do exemplo no espaço de cláusulas que é classificado pelo SVM.

A abordagem híbrida busca uma metodologia que melhor combine as saídas de classificadores individuais. Para mostrar isto, comparamos o classificador híbrido com o classificador formado por disjunção de cláusulas.

Foi empregada validação cruzada em 10 *folds*. Com o intuito de maximizar a significação estatística das comparações, em ambos os experimentos, usamos os mesmos exemplos nos conjuntos de treino e de teste em cada *fold*. A avaliação dos classificadores é feita calculando a acurácia média nos conjuntos de teste.

Para gerar o classificador híbrido, executamos os passos 1 ao 3 do Algoritmo 6. Os exemplos da matriz de cobertura obtida foram separados em grupos para a formação dos conjuntos de treino e teste em cada *fold*. O passo 4 do Algoritmo 6 foi executado em cada *fold* com o respectivo conjunto de treinamento. Utilizamos a implementação SVM-Perf[1] de JOACHIMS [40] mantendo os valores padrões dos parâmetros do SVM em todos os experimentos.

Para gerar o classificador disjunção, geramos as cláusulas individualmente para cada exemplo positivo do conjunto de treinamento de cada *fold* e criamos o classificador fazendo disjunção das cláusulas.

5.3 Resultados

Na Tabela 5.1 comparamos as acurácias médias obtidas pelo classificador híbrido e pelos *ensembles* de teorias e cláusulas na validação cruzada.

[1] http://svmlight.joachims.org/svm_perf.html

Tabela 5.1: Comparação das acurácias médias obtidas na validação cruzada pelo classificador híbrido e pelos métodos de *ensemble*

Bases de dados	Classificador Híbrido	*Ensemble* teorias	*Ensemble* cláusulas
Amine	0,88	0,82	0,81
Carcinogenesis	0,64	-	-
Choline	0,81	0,81	0,78
Mutagenesis	0,84	-	-
Protein	0,84	0,81	0,81

Nas três bases de dados usadas nos experimentos com *ensembles*, o classificador híbrido obtém uma acurácia melhor em todas, quando comparado com *ensemble* de cláusulas.

Comparado ao *ensemble* de teorias, o classificador híbrido obtém um melhor resultado nas bases de dados *Amine* e *Protein*, e uma acurácia igual na base de dados *Choline*.

Comparamos também a diferença na acurácia quando combinamos as cláusulas através do SVM (abordagem híbrida) e quando combinamos as cláusulas fazendo a disjunção delas (tal qual numa teoria). Os resultados são mostrados na Tabela 5.2. As acurácias são apresentadas com o seus respectivos intervalos de confiança de 95%. Na última coluna incluímos o grau de confiança associado à diferença de acurácias, calculado usando o teste-t e a distribuição de Student.

Tabela 5.2: Comparação das acurácias médias com os respectivos intervalos de confiança de 95% entre a abordagem híbrida e a simples disjunção de cláusulas.

Bases de dados	Híbrido	Disjunção	Diferença acurácia	teste-t
Amine	$0,88 \pm 0,02$	$0,84 \pm 0,03$	0,04	**99,56%**
Carcinogenesis	$0,64 \pm 0,05$	$0,60 \pm 0,07$	0,04	81,03%
Choline	$0,81 \pm 0,01$	$0,79 \pm 0,01$	0,02	**98,67%**
Mutagenesis	$0,84 \pm 0,04$	$0,80 \pm 0,04$	0,04	**95,91%**
Protein	$0,84 \pm 0,01$	$0,79 \pm 0,03$	0,05	**99,89%**

A partir dos resultados apresentados na Tabela 5.2 podemos afirmar que os classificadores híbridos ultrapassam significativamente o desempenho dos classificadores ILP puros.

Entretanto, a comparação feita entre as duas formas de combinar as decisões das cláusulas (SVM versus disjunção) está intimamente relacionada com a maneira que o sistema ILP gerou as cláusulas. Teoricamente existe a possibilidade que para um outro conjunto de cláusulas a abordagem híbrida não seja mais vantajosa. Aleph aceita muitas configurações de parâmetros, cada uma das quais deriva num conjunto diferente de cláusulas geradas. Aos efeitos de mostrar que o ganho da abordagem híbrida independe de como

as cláusulas são geradas, exploramos vários conjuntos de configurações possíveis para o funcionamento do Aleph. Para cada aplicação, comparamos os desempenhos de ambas as abordagens para 7 configurações diferentes dos parâmetros do Aleph. Os parâmetros escolhidos foram os seguintes:

- Número máximo de literais em uma cláusula (*clauselength*): 5 (padrão), 4 e 6 (variações)
- Função de avaliação (*evalfn*): *coverage* (padrão) e *m-estimate* (variação) cuja medida é a estimativa *m* descrita em [20]. O valor de *m* foi automaticamente ajustado para ser a estimativa de máxima verosemelhança do melhor valor de *m*.
- Encadeamento das variáveis (*i*): 5 (padrão), 4 e 6 (variações)
- Valor mínimo da acurácia aceitável de uma cláusula (*minacc*): 0,9 (padrão), 0,7 e 1,0 (variações)

Ainda que o Aleph permita a modificação de vários outros parâmetros, apenas controlamos os parâmetros *clauselength*, *evalfn*, *i*, *nodes* e *minacc* pois eles variam o espaço de busca e modificam a maneira como as cláusulas são selecionadas.

As Tabelas 5.3 a 5.7 comparam as acurácias médias para todas as combinações dos parâmetros considerados. As acurácias são apresentadas junto com os seus respectivos intervalos de confiança de 95% . Na última coluna incluímos o grau de confiança associado à diferença de acurácias, calculado usando o teste-t e a distribuição de Student.

Tabela 5.3: Acurácias médias com os respectivos intervalos de confiança de 95% para a abordagem híbrida e para a disjunção na base de dados *Amine*.

Parâmetro variado	*Amine*		Diferença acurácia	teste-t
	Híbrido	Disjunção		
$clauselenght = 4$	$0,85 \pm 0,02$	$0,84 \pm 0,03$	$0,01$	$75,10\%$
$clauselenght = 6$	$0,88 \pm 0,02$	$0,84 \pm 0,03$	$0,04$	$\mathbf{99,04\%}$
$evalfn = mestimate$	$0,88 \pm 0,02$	$0,85 \pm 0,03$	$0,03$	$\mathbf{99,03\%}$
$i = 4$	$0,88 \pm 0,02$	$0,84 \pm 0,03$	$0,04$	$\mathbf{99,56\%}$
$i = 6$	$0,88 \pm 0,02$	$0,83 \pm 0,03$	$0,05$	$\mathbf{99,56\%}$
$minacc = 0,7$	$0,82 \pm 0,03$	$0,73 \pm 0,03$	$0,09$	$\mathbf{99,93\%}$
$minacc = 1,0$	$0,86 \pm 0,01$	$0,83 \pm 0,03$	$0,03$	$\mathbf{96,74\%}$

Tabela 5.4: Acurácias médias com os respectivos intervalos de confiança de 95% para a abordagem híbrida e para a disjunção na base de dados *Carcinogenesis*.

Parâmetro variado	*Carcinogenesis*		Diferença acurácia	teste-t
	Híbrido	Disjunção		
$clauselenght = 4$	$0,66 \pm 0,05$	$0,59 \pm 0,06$	$0,07$	$\mathbf{95,59}\%$
$clauselenght = 6$	$0,64 \pm 0,05$	$0,59 \pm 0,05$	$0,05$	$89,55\%$
$evalfn = mestimate$	$0,65 \pm 0,02$	$0,59 \pm 0,08$	$0,06$	$90,06\%$
$i = 4$	$0,64 \pm 0,05$	$0,60 \pm 0,07$	$0,04$	$81,03\%$
$i = 6$	$0,64 \pm 0,05$	$0,60 \pm 0,07$	$0,04$	$81,03\%$
$minacc = 0,7$	$0,62 \pm 0,05$	$0,60 \pm 0,04$	$0,02$	$74,42\%$
$minacc = 1,0$	$0,66 \pm 0,04$	$0,58 \pm 0,08$	$0,08$	$\mathbf{98,94}\%$

Tabela 5.5: Acurácias médias com os respectivos intervalos de confiança de 95% para a abordagem híbrida e para a disjunção na base de dados *Choline*.

Parâmetro variado	*Choline*		Diferença acurácia	teste-t
	Híbrido	Disjunção		
$clauselenght = 4$	$0,76 \pm 0,01$	$0,73 \pm 0,01$	$0,03$	$\mathbf{99,39}\%$
$clauselenght = 6$	$0,82 \pm 0,01$	$0,79 \pm 0,02$	$0,03$	$\mathbf{98,97}\%$
$evalfn = mestimate$	$0,81 \pm 0,01$	$0,79 \pm 0,01$	$0,02$	$\mathbf{99,80}\%$
$i = 4$	$0,81 \pm 0,01$	$0,79 \pm 0,01$	$0,02$	$\mathbf{98,67}\%$
$i = 6$	$0,81 \pm 0,01$	$0,79 \pm 0,01$	$0,02$	$\mathbf{98,67}\%$
$minacc = 0,7$	$0,80 \pm 0,01$	$0,70 \pm 0,02$	$0,10$	$\mathbf{99,99}\%$
$minacc = 1,0$	$0,74 \pm 0,01$	$0,70 \pm 0,01$	$0,04$	$\mathbf{99,64}\%$

Tabela 5.6: Acurácias médias com os respectivos intervalos de confiança de 95% para a abordagem híbrida e para a disjunção na base de dados *Mutagenesis*.

Parâmetro variado	*Mutagenesis*		Diferença acurácia	teste-t
	Híbrido	Disjunção		
$clauselenght = 4$	$0,84 \pm 0,05$	$0,79 \pm 0,04$	$0,05$	$\mathbf{95,03}\%$
$clauselenght = 6$	$0,84 \pm 0,04$	$0,80 \pm 0,04$	$0,04$	$\mathbf{95,91}\%$
$evalfn = mestimate$	$0,87 \pm 0,05$	$0,80 \pm 0,04$	$0,07$	$\mathbf{98,53}\%$
$i = 4$	$0,84 \pm 0,04$	$0,80 \pm 0,04$	$0,04$	$\mathbf{95,91}\%$
$i = 6$	$0,84 \pm 0,04$	$0,80 \pm 0,04$	$0,04$	$\mathbf{95,91}\%$
$minacc = 0,7$	$0,85 \pm 0,05$	$0,72 \pm 0,01$	$0,13$	$\mathbf{99,99}\%$
$minacc = 1,0$	$0,83 \pm 0,05$	$0,80 \pm 0,03$	$0,03$	$93,80\%$

Tabela 5.7: Acurácias médias com os respectivos intervalos de confiança de 95% para a abordagem híbrida e para a disjunção na base de dados *Protein*.

Parâmetro variado	*Protein*		Diferença acurácia	teste-t
	Híbrido	Disjunção		
$clauselenght = 4$	$0,83 \pm 0,02$	$0,80 \pm 0,03$	$0,03$	$\mathbf{99,25}\%$
$clauselenght = 6$	$0,85 \pm 0,02$	$0,76 \pm 0,04$	$0,09$	$\mathbf{99,97}\%$
$evalfn = mestimate$	$0,84 \pm 0,02$	$0,79 \pm 0,02$	$0,05$	$\mathbf{99,94}\%$
$i = 4$	$0,84 \pm 0,01$	$0,79 \pm 0,03$	$0,05$	$\mathbf{99,89}\%$
$i = 6$	$0,84 \pm 0,01$	$0,79 \pm 0,03$	$0,05$	$\mathbf{99,89}\%$
$minacc = 0,7$	$0,82 \pm 0,02$	$0,75 \pm 0,04$	$0,07$	$\mathbf{99,96}\%$
$minacc = 1,0$	$0,84 \pm 0,02$	$0,79 \pm 0,03$	$0,05$	$\mathbf{99,90}\%$

Os resultados apresentados nas tabelas anteriores permitem concluir que na maioria dos casos, independentemente da configuração de parâmetros empregada para gerar as cláusulas, a abordagem híbrida ultrapassa significativamente em performance à alternativa de fazer a disjunção de cláusulas. Portanto, os resultados obtidos na Tabela 5.2 são representativos da vantagem que decorre de usar o algoritmo de aprendizado híbrido com respeito de usar unicamente ILP.

A comparação dos resultados obtidos nestas bases de dados com outros trabalhos na literatura é um tanto difícil. Parte desta dificuldade se deve aos algoritmos serem muito sensíveis na distribuição de exemplos nos *folds* para a validação cruzada.

Por exemplo, em uma outra implementação de SVM, chamada de PyML[2], onde a validação cruzada é automática, chegamos a obter as acurácias apresentadas na Tabela 5.8.

Tabela 5.8: Acurácia no PyML

Bases de dados	Acurácia
Amine	0,92
Carcinogenesis	0,76
Choline	0,81
Mutagenesis	0,86
Protein	0,88

Base de dados de mamografias

O conjunto de dados mais desafiador do nosso estudo foi um banco de dados relacional real de mamografias, extraída do "*National Mammography Database*" (NMD) americano em um trabalho conjunto com a Universidade de Wisconsin-Madison.

Este conjunto consiste de 47669 exames mamográficos de 18270 pacientes realizados entre o período de 5 de Abril de 1999 a 9 de Fevereiro de 2004, no *Froedter College* e

[2]http://pyml.sourceforge.net/

*Breast Imaging Cente*r da Escola de Medicina de Wisconsin. A base de dados contém 435 anormalidades malignas e 65365 anormalidades benignas (uma mamografia pode conter múltiplas anormalidades), num total de 65800 exemplos. A tarefa de aprendizado é predizer se uma dada anormalidade é maligna.

Estes dados foram utilizados por DAVIS *et al.* [18] para construir modelos probabilísticos de banco de dados relacionais chamado de Aprendizado Relacional Estatístico (SRL) que integra aprendizado de atributos, agregados e regras.

A metodologia que eles utilizaram foi a seguinte: os dados foram divididos em 10 conjuntos, cada um com aproximadamente 1/10 de anormalidades malignas e 1/10 de anormalidades benignas para fazer a validação cruzada em 10 *folds*. Todas as anormalidades pertencentes a um mesmo paciente ficaram no mesmo conjunto. O classificador foi construído em 2 etapas. Na primeira etapa, 4 *folds* de dados foram usados para aprender regras ILP. Depois, os restantes 5 *folds* foram usados para aprender a estrutura e parâmetros de uma Rede de Bayes e o último *fold* foi usado para teste.

Foram geradas milhares de cláusulas em cada *fold*. Estas cláusulas foram depois processadas através de um algoritmo guloso onde foram selecionadas as cláusulas com maior pontuação na cobertura dos primeiros novos exemplos. Ao todo, ficaram 8067 cláusulas em todos os *folds*.

Uma segunda seleção de cláusulas foi efetuada, em cada *fold*, onde as 50 melhores cláusulas foram selecionadas baseado em 3 critérios: elas deveriam ser 1) multi-relacionais, 2) distintas e 3) cobrir um número significativo de casos malignos.

As matrizes de cobertura das 50 cláusulas selecionadas foram geradas em cada *fold* e adicionadas como atributos para a construção de uma rede TAN (*Tree Augmented Naive Bayes*) [34] que é uma rede Bayesiana que pode expressar relações parciais de dependência entre atributos.

Para mostrar a eficiência da metodologia híbrida e compará-la com a metodologia Bayesiana, utilizamos as saídas das cláusulas geradas como entrada para o classificador SVM.

Fizemos a matriz de cobertura dos 65800 exemplos considerando todas as 8087 cláusulas que foram primeiramente selecionadas. Esta matriz continha muitas linhas repetidas (exemplos com a mesma classificação) e conseguimos reduzi-la para 8067 linhas (exemplos) retirando apenas as repetições que do ponto de vista do algoritmo de aprendizado SVM não faz diferença pois representam o mesmo ponto de dado no espaço.

Dividimos a matriz de cobertura em conjuntos de treino e teste para validação cruzada em 10 *folds* e fizemos o treinamento utilizando o SVM.

A Tabela 5.9 mostra os resultados do valores médios de acurácia padrão, *precision*, *recall*, área sob a curva *precision-recall* e tempo total gasto. O valor da área sob a curva *precision-recall* refere-se a *recalls* acima de 50% para compararmos com [18].

Estes resultados ultrapassam o desempenho de DAVIS *et al*. [18], no que diz respeito à

Tabela 5.9: Resultados da abordagem híbrida para a base de mamografias

Medida	Média na validação cruzada
Acurácia	0,94
Precision	0,36
*Re*call	0,35
AUC-PR-50	0,11
Tempo	7 min.

área sob a curva *precision-recall* (Figura 5.1). Naquele trabalho, é reportado um resultado máximo pouco acima de 0,07 de área para a melhor metodologia empregada, onde as cláusulas eram entrada para um classificador Bayesiano. Com a metodologia híbrida, obtemos o valor 0,11 para a mesma área calculada.

Outra vantagem que obtemos foi em relação ao tempo de execução. Na nossa metodologia, o tempo total de execução para a geração e teste do classificador SVM para todos os 10 *folds* da validação cruzada foi de apenas 7 min. (considerando que as cláusulas já estavam geradas). Por outro lado, a construção da rede TAN chegava a levar mais de um dia para ser feita.

Um outro diferencial da nossa abordagem foi o uso de todas as 8087 cláusulas primeiramente selecionadas para construir a matriz de cobertura que foi utilizada como entrada para o SVM enquanto na rede TAN o número de cláusulas foi limitada a 50 cláusulas por *fold*.

Nesta base de dados, o experimento foi feito usando um computador com processador Intel Core Duo 6300, com 3 GB de memória RAM, na plataforma Linux 32 bits. As versões do Prolog, Aleph e SVM foram as mesmas dos experimentos anteriores.

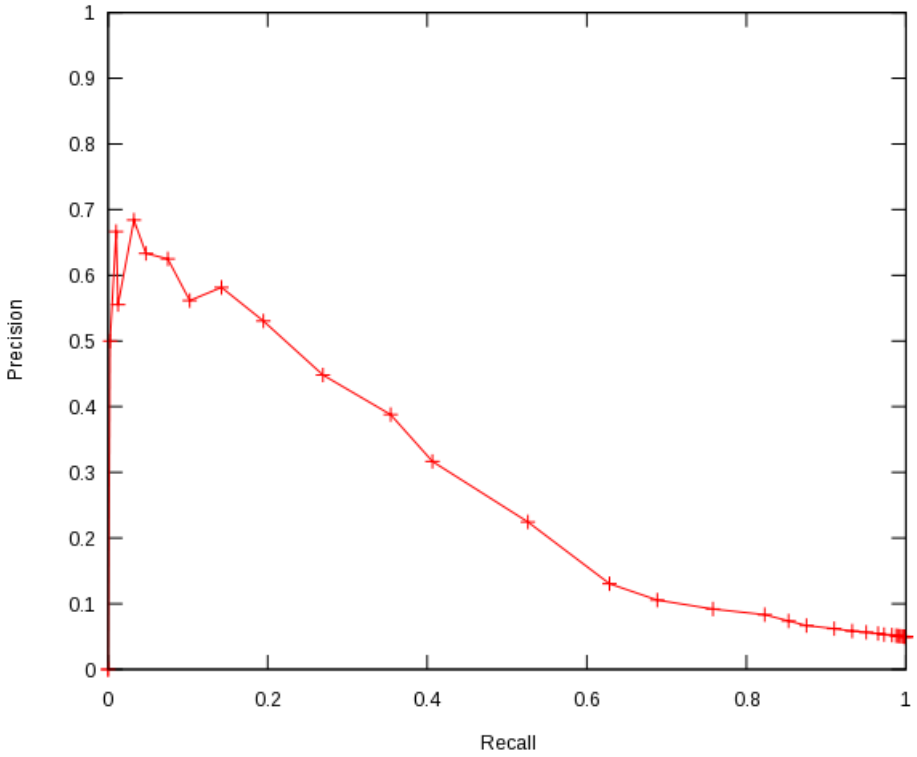


Figura 5.1: Curva *Precision-Recall* para a base de mamografias

Capítulo 6

Conclusão

Programação Lógica Indutiva (*Inductive Logic Programming* (ILP)) combina aprendizado de máquina com o poder de expressão da Lógica de Primeira Ordem. Como consequência, ILP traz duas grandes vantagens: produz classificadores que são de fácil entendimento por especialistas e consegue resolver problemas de aprendizado multi-relacional, em contraste com outros métodos numéricos, probabilísticos ou proposicionais tais como Redes Neuronais, Bayesianas e Árvores de Decisão.

A tarefa de um sistema ILP é, dados um conjunto de exemplos (observações) rotulados como positivos e negativos e a descrição destes exemplos, gerar um classificador que melhor separe os exemplos positivos dos negativos, através da geração de um modelo representado na linguagem de Lógica de Primeira Ordem. Este modelo consiste de um conjunto de cláusulas e se denomina teoria.

Os sistemas ILP empregam estratégias gulosas para a geração das cláusulas. Uma consequência desta característica é que a precisão das cláusulas diminui conforme elas são geradas; assim, a acurácia da teoria também diminui globalmente, principalmente devido ao aumento do número de exemplos erroneamente classificados como positivos.

Além disso, a busca por uma teoria dentro do sistema ILP consome muito tempo e frequentemente produz classificadores excessivamente complexos (com muitas cláusulas) o que dificulta a interpretação por parte de um especialista.

Tradicionalmente uma teoria é construída em um algoritmo iterativo de remoção de cobertura (*cover-removal*) na qual uma cláusula gerada condiciona a construção das seguintes.

No presente trabalho consideramos uma estratégia diferente na geração das cláusulas que conformam a teoria. A diferença da abordagem clássica é que consideramos a criação das cláusulas de forma independente, ou seja, o algoritmo deixa de ser iterativo para produzir sequências de cláusulas. Em seguida, estudamos diferentes modos de combinar as classificações que cada cláusula faz de um exemplo dado, com o intuito de aumentar a robustez do classificador conjunto, e apresentamos três abordagens para gerar, selecionar e combinar cláusulas geradas pelo sistema ILP Aleph.

Concluímos que o aprendizado de cláusulas, que são classificadores mais simples e mais rápidos de se obter, de forma independente, produz classificadores tão bons ou melhores do que com o aprendizado de teorias, onde cláusulas são aprendidas condicionadas às geradas anteriormente. Utilizamos, para isto, um método de *ensemble* clássico na literatura, chamado de *bagging*, para gerar os conjuntos de treinamento para cada base de dados. Mostramos que *ensemble* de cláusulas, combinadas através de votação com limiar, atinge acurácia comparada com teorias reduzindo o tempo de obtenção do classificador em mais de 90% no melhor caso, além de apresentar um classificador muito mais simples (com menos cláusulas).

A partir desta conclusão, exploramos outras alternativas de combinação de cláusulas e apresentamos mais duas contribuições do trabalho.

Uma das contribuições é o classificador que denominamos de "Nata". Nesta abordagem é feita a seleção do menor conjunto de cláusulas que maximiza a cobertura de exemplos positivos. Esta combinação de cláusulas gera um classificador mais sucinto e menos complexo que pode ser entendido mais facilmente. Além disso, por usar linguagem da Lógica de Primeira Ordem, esta abordagem é mais sucinta do que as regras que são geradas, por exemplo, através de Árvores de Decisão. Os classificadores gerados usando esta abordagem são particularmente interessantes do ponto de vista do especialista pois consideram o menor número de cláusulas facilitando assim a tarefa de extração de conhecimento de um domínio.

A outra contribuição é a abordagem que denominamos de híbrida, a qual utiliza SVM para combinar as classificações feitas pelas cláusulas gerando um classificador mais preciso que obteve melhores resultados sobre cinco bases de dados clássicas utilizadas na literatura para aferir classificadores baseados em ILP. Além disso, esta abordagem também foi bem sucedida em uma base de dados relacional real de mamografias, extraída do "*National Mammography Database*" (NMD) americano.

Os resultados obtidos nos experimentos computacionais mostram a eficiência destas estratégias. Em particular podemos destacar:

- *Ensembles* de cláusulas com ILP são poderosos métodos para se obter boas acurácias num curto período de tempo;
- O método de *bagging* é eficaz para obter classificadores de boa qualidade, mesmo quando os classificadores individuais são fracos como simples cláusulas;
- O tamanho do classificador final obtido com cláusulas é bastante reduzido com relação à teorias.
- O tempo total gasto para gerar os classificadores baseados em cláusulas é bastante inferior.

- A acurácia obtida, principalmente quando combinamos os resultados das cláusulas com SVM, é significativamente superior à simples disjunção.
- Ao utilizar SVM para combinar as saídas dadas pelas cláusulas para os exemplos temos duas vantagens:
  - Preservamos todas as componentes originais;
  - Aduzimos novas componentes (derivadas das cláusulas) com forte poder discriminatório.

Como trabalhos futuros, gostaríamos de:

- Fazer os experimentos feitos com SVM e disjunção com outras formas de combinar as decisões das cláusulas, por exemplo, usando abordagens Bayesianas.
- Relacionar os vetores de suporte com os exemplos e ver o tipo de cláusula que estes geraram para: a) consideração do especialista e b) tentar caracterizar esses exemplos de maneira que possa ser criada uma heurística para a seleção de sementes na construção de teorias do Aleph.
- Estudar o comportamento e a eficiência do algoritmo SVM alterando seus parâmetros na construção do classificador.

Referências Bibliográficas

[1] AIZERMAN, M. A., BRAVERMAN, E. M., ROZONOER, L. I., "Theoretical foundations of the potential function method in pattern recognition learning", *Automation and Remote Control*, v. 25, pp. 821–837, 1964.

[2] BLANZ, V., SCHÖLKOPF, B., BULTHOFF, H., *et al.*, "Comparison of view-based object recognition algorithm using realistic 3d models", In: *Artificial Neural Networks - ICANN 96* (VON DER MALSBURG, C., VON SEELEN, W., VORBRUGGEN, J. C., *et al.*, eds.), v. 1112 of *Lecture Notes in Computer Science*, (Berlin), pp. 251–256, Springer, 1996.

[3] BLOCKEEL, H., DEHASPE, L., DEMOEN, B., *et al.*, "Executing query packs in ILP", In: *Proceedings of the 10th International Conference on Inductive Logic Programming* (CUSSENS, J., FRISCH, A., eds.), v. 1866 of *Lectures Notes in Artificial Intelligence*, pp. 60–77, Springer-Verlag, 2000.

[4] BLOCKEEL, H., DEMOEN, B., JANSSENS, G., *et al.*, "Two advanced transformations for improving the efficiency of an ILP system", In: *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming* (CUSSENS, J., FRISCH, A., eds.), pp. 43–59, 2000.

[5] BOSER, B. E., GUYON, I. M., VAPNIK, V. N., "A training algorithm for optimal margin classifiers", In: *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory* (HAUSSLER, D., ed.), (Pittsburgh, PA), pp. 144–152, ACM Press, 1992.

[6] BRATKO, I., GROBELNIK, M., "Inductive learning applied to program construction and verification", pp. 279–292, 1993.

[7] BREIMAN, L., "Bagging Predictors", *Machine Learning*, v. 24, n. 2, pp. 123–140, 1996.

[8] BREIMAN, L., "Bias, variance, and arcing classifiers", technical report 460, UC-Berkeley, Berkeley, CA, 1996.

[9] BROWN, M. P. S., GRUNDY, W. N., LIN, D., *et al.*, "Knowledge-based analysis of microarray gene expression data by using support vector machines", In: *Proceedings of the National Academy of Sciences*, v. 97, pp. 262–267, 2000.

[10] BURGES, C. J. C., "A Tutorial on Support Vector Machines for Pattern Recognition", *Data Mining and Knowledge Discovery*, v. 2, n. 2, pp. 121–167, 1998.

[11] BURGES, C. J. C., SCHÖLKOPF, B., "Improving th accuracy and speed of support vector learning machines", In: *Advances in Neural Information Processing Systems* (MOZER, M., JORDAN, M., PETSCHE, T., eds.), v. 9, (Cambridge, MA), pp. 375–381, MIT Press, 1997.

[12] CAMACHO, R., FONSECA, N. A., COSTA, V. S., "ILP :- Just Trie It", In: *Proceedings of the 2007 International Conference on Inductive Logic Programming* (BLOCKEEL, H., ed.), n. 4894 in Lecture Notes in Artificial Intelligence, pp. 78–87, Springer-Verlag, 2008. Revised version of the paper appearing in the Work-in-progress proceedings.

[13] CHANG, C., LIN, C., "Libsvm: a library for support vector machines". Software disponível em http://www.csie.ntu.edu.tw/ cjlin/libsvm, 2001.

[14] CORTES, C., VAPNIK, V., "Suport vector networks", *Machine Learning*, v. 20, pp. 273–297, 1995.

[15] CRISTIANINI, N., SHAWE-TAYLOR, J., *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[16] DAVIS, J., COSTA, V. S., ONG, I. M., *et al.*, "Using Bayesian Classifiers to Combine Rules", In: *3rd Workshop on Multi-Relational Data Mining*, (Seattle, USA), Aug. 2004.

[17] DAVIS, J., DUTRA, I. C., PAGE, D., *et al.*, "Establishing identity equivalence in multi-relational domains", In: *Proceedings of the International Conference on Intelligence Analysis, McLean, VA, USA, May 2–6*, May 2005. Awarded best technical paper.

[18] DAVIS, J., BURNSIDE, E., DUTRA, I. C., *et al.*, "View Learning for Statistical Relational Learning: With an Application to Mammography", In: *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005.

[19] DAVIS, J., PAGE, D., BURNSIDE, E., *et al.*, *Introduction to Statistical Relational Learning*, ch. Learning a New View of a Database: With an Application in Mammography. MIT Press, 2007.

[20] DŽEROSKI, S., BRATKO, I., "Handling noise in inductive logic programming", In: *Proceedings of Second International Workshop on Inductive Logic Programming* (MUGGLETON, S., FURUKAWA, K., eds.), (Tokyo, Japan), Institute for New Generation Computer Technology, 1992.

[21] DŽEROSKI, S., DEHASPE, L., RUCK, B., *et al.*, "Classification of river water quality data using machine learning", In: *Proceedings of the 5th International Conference on the Development and Application of Computer Techniques to Environmental Studies*, 1995.

[22] DEHASPE, L., DE RAEDT, L., "Parallel inductive logic programming", In: *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.

[23] DIETTERICH, T. G., "Ensemble Methods in Machine Learning", *Lecture Notes in Computer Science*, v. 1857, pp. 1–15, 2000.

[24] DIETTERICH, T. G., "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization", *Machine Learning*, v. 40, n. 2, pp. 139–157, 2000.

[25] DOLSAK, B., MUGGLETON, S., "The application of ILP to finite element mesh design", In: *Proceedings of the First International Workshop on Inductive Logic Programming (ILP-91)* (MUGGLETON, S., ed.), pp. 225–242, 1991.

[26] DUTRA, I. C., PAGE, D., COSTA, V. S., *et al.*, "An empirical evaluation of bagging in inductive logic programming", In: *Proceedings of the 12th International Conference on Inductive Logic Programming* (MATWIN, S., SAMMUT, C., eds.), v. 2583 of *Lectures Notes in Artificial Intelligence*, pp. 48–65, Springer-Verlag, September 2002.

[27] DUTRA, I. C., PAGE, D., COSTA, V. S., *et al.*, "Toward automatic management of embarrassingly parallel applications", In: *Euro-Par03, Klagenfurt, Austria, August 26 - 29*, pp. 509–516, 2003.

[28] FLETCHER, R., *Practical Methods of Optimization*. 2 ed. John Wiley and Sons, 1987.

[29] FONSECA, N. A., CAMACHO, R., ROCHA, R., "Compile the hypothesis space: do it once, use it often", *Fundamenta Informaticae, Special Issue on Multi-Relational Data Mining (to appear)*, 2008.

[30] FONSECA, N. A., CAMACHO, R., SILVA, F., "Strategies to Parallelize ILP Systems", In: *Proceedings of the 15th International Conference on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence, Springer-Verlag, 2005.

[31] FONSECA, N. A., COSTA, V. S., ROCHA, R., "Improving the Efficiency of ILP Systems", *Software, Practice & Experience (Epub ahead of print)*, 2008.

[32] FONSECA, N. A., SILVA, F., COSTA, V. S., *et al.*, "A pipelined data-parallel algorithm for ILP", In: *Proceedings of 2005 IEEE International Conference on Cluster Computing*, IEEE, September 2005.

[33] FREUND, Y., SHAPIRE, R., "Experiments with a new boosting algorithm", In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 148–156, Morgan Kaufman, 1996.

[34] FRIEDMAN, N., GEIGER, D., GOLDSZMIDT, M., "Bayesian network classifiers", *Machine Learning*, v. 29, n. 2-3, pp. 131–163, 1997.

[35] GOADRICH, M., OLIPHANT, L., SHAVLIK, J., "Learning ensembles of first-order clauses for recall-precision curves: A case study in biomedical information extraction", In: *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP)*, (Porto, Portugal), 2004.

[36] GRAHAM, J., PAGE, D., WILD, A., "Parallel inductive logic programming", In: *Proceedings of the Systems, Man, and Cybernetics Conference*, 2000.

[37] HANSEN, L., SALAMON, P., "Neural Network Ensembles", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 12, n. 10, pp. 993–1001, 1990.

[38] JOACHIMS, T., "Text categorization with support vector machines", technical report ls viii number 23, University of Dortmund, 1997.

[39] JOACHIMS, T., "Making large-scale svm learning practical", In: *Advances in Kernel Methods - Support Vector Learning* (SCHÖLKOPF, B., SMOLA, A., eds.), (Cambridge, MA), pp. 169–184, MIT Press, 1999.

[40] JOACHIMS, T., "Training linear svms in linear time", In: *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 217–226, ACM, 2006.

[41] KECMAN, V., *Learning and Soft Computing: Support Vector Machines, Neural Networks, Fuzzy Logic Systems Models*. Cambridge, MA, MIT Press, 2001.

[42] KING, R., MUGGLETON, S., STERNBERG, M., "Predicting protein secondary structure using inductive logic programming", *Protein Engineering*, v. 5, pp. 647–657, 1992.

[43] LAVRAČ, N., DŽEROSKI, S., *Relational Data Mining*. Berlin, Springer-Verlag, 2001.

[44] MACLIN, R., OPITZ, D., "An empirical evaluation of bagging and boosting", In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, (Cambridge, MA), pp. 546–551, AAAI Press/MIT Press, 1997.

[45] MATSUI, T., INUZUKA, N., SEKI, H., *et al.*, "Parallel induction algorithms for large samples", In: *Proceedings of the First International Conference on Discovery Science* (ARIKAWA, S., MOTODA, H., eds.), v. 1532 of *Lecture Notes in Artificial Intelligence*, pp. 397–398, Springer-Verlag, December 1998.

[46] MITCHELL, T. M., *Machine Learning*. McGraw-Hill, 1997.

[47] MOONEY, R. J., MELVILLE, P., TANG, R. L., *et al.*, *Data Mining: Next Generation Challenges and Future Directions*, ch. Relational Data Mining with Inductive Logic Programming for Link Discovery, pp. 239–254. AAAI Press, 2004.

[48] MORELLI, V. A., *Implementação e Análise de Técnicas de Ensemble em Programação Lógica Indutiva*, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2003.

[49] MUGGLETON, S., "Inverse Entailment and Progol", *New Generation Computing, Special issue on Inductive Logic Programming*, v. 13, n. 3-4, pp. 245–286, 1995.

[50] MUGGLETON, S., BUNTINE, W., "Machine invention of first order predicates by inverting resolution", In: *Proceedings of the Fifth International Conference on Machine Learning (ICML-88)*, pp. 339–351, 1988.

[51] MUGGLETON, S., LODHI, H., AMINI, A., *et al.*, "Support Vector Inductive Logic Programming", In: *Proceedings of the 8th International Conference on Discovery Science*, LNAI 3735, pp. 163–175, Springer-Verlag, 2005.

[52] MUGGLETON, S., "Inductive logic programming", *New Generation Computing*, v. 8, n. 4, pp. 295–318, 1991.

[53] MUGGLETON, S., RAEDT, L. D., "Inductive logic programming: Theory and methods", *Journal of Logic Programming*, v. 19-20, pp. 629–679, 1994.

[54] OSUNA, E., FREUND, R., GIROSI, F., "Training support vector machines: an application to face detection", In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 130–136, 1997.

[55] PARMANTO, B., MUNRO, P. W., DOYLE, H. R., "Improving commitee diagnosis with resampling techniques", In: *Advances in Neural Information Processing Systems* (TOURETZKY, D. S., MOZER, M. C., HESSELMO, M. E., eds.), v. 8, (Cambridge, MA), pp. 882–888, MIT Press, 1996.

[56] PLATT, J., "Fast training of support vector machines using sequential minimal optimization", In: *Advances in Kernel Methods - Support Vector Learning* (SCHÖLKOPF, B., SMOLA, A., eds.), (Cambridge, MA), pp. 185–208, MIT Press, 1999.

[57] POMPE, U., KONONENKO, I., "Naive Bayesian classifier within ILP-R", In: *Proceedings of the 5th International Workshop on Inductive Logic Programming* (DE RAEDT, L., ed.), pp. 417–436, Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[58] QUINLAN, J. R., "Learning Logical Definitions from Relations", *Machine Learning*, v. 5, n. 3, pp. 239–266, 1990.

[59] QUINLAN, J. R., "Boosting first-order learning", In: *Proceedings of the 7th International Workshop on Algorithmic Learning Theory* (ARIKAWA, S., SHARMA, A., eds.), v. 1160 of *Lecture Notes in Computer Science*, pp. 143–155, Springer, 1996.

[60] ROBINSON, J., "A Machine Oriented Logic Based on the Resolution Principle", *Journal of the ACM*, v. 12, n. 1, pp. 23–41, 1965.

[61] RUSSEL, S., NORVIG, P., *Artificial Intelligence: A Modern Approach*. 2 ed. Prentice Hall, 2003.

[62] SALVINI, R., AGUILAR, E., DUTRA, I. C., "Skimmed classifiers", In: *Work-in-Progress Proceedings of the 2007 International Conference on Inductive Logic Programming*, (Oregon, USA), June 19-21 2007.

[63] SALVINI, R., DUTRA, I. C., MORELLI, V. A., "Simple and effective classifiers to model biological data", In: *Proceedings of the Second International Symposium on Mathematical and Computational Biology*, pp. 379–394, World Scientific Publishing, 2005.

[64] SANTOS COSTA, V., SRINIVASAN, A., CAMACHO, R., "A note on two simple transformations for improving the efficiency of an ILP system", In: *Proceedings of the 10th International Conference on Inductive Logic Programming* (CUSSENS, J., FRISCH, A., eds.), v. 1866 of *Lectures Notes in Artificial Intelligence*, pp. 225–242, Springer-Verlag, 2000.

[65] SCHMIDT, M. S., "Identifying speakers with support vector networks", In: *Interface '96 Proceedings*, 1996.

[66] SCHÖLKOPF, B., BURGES, C. J. C., VAPNIK, V., "Extracting support data for a given task", In: *Proceedings of the First International Conference on Knowledge and Data Mining* (FAYYAD, U. M., UTHURUSAMY, R., eds.), (Mento Park, CA), AAAI Press, 1995.

[67] SCHÖLKOPF, B., BURGES, C. J. C., VAPNIK, V., "Incorporating invariances in support vector learning machines", In: *Artificial Neural Networks - ICANN 96* (VON DER MALSBURG, C., VON SEELEN, W., VORBRUGGEN, J. C., *et al.*, eds.), v. 1112 of *Lecture Notes in Computer Science*, (Berlin), pp. 47–52, Springer, 1996.

[68] SCHÖLKOPF, B., SMOLA, A. J., *Learning with Kernels*. Cambridge, MA, MIT Press, 2002.

[69] SEBAG, M., ROUVEIROL, C., "Tractable induction and classification in first-order logic via stochastic matching", In: *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 888–893, 1997.

[70] SRINIVASAN, A., "A study of two sampling methods for analysing large datasets with ILP", *Data Mining and Knowledge Discovery*, v. 3, n. 1, pp. 95–123, 1999.

[71] SRINIVASAN, A., *The Aleph Manual*, 2001.

[72] STRUYF, J., BLOCKEEL, H., "Efficient cross-validation in ILP", In: *Proceedings of the 11th International Conference on Inductive Logic Programming* (ROUVEIROL, C., SEBAG, M., eds.), v. 2157 of *Lectures Notes in Artificial Intelligence*, pp. 228–239, Springer-Verlag, September 2001.

[73] VAPNIK, V., *The Nature of Statistical Learning Theory*. New York, Springer-Verlag, 1995.

[74] ZELEZNY, F., SRINIVASAN, A., PAGE, D., "Lattice-search runtime distributions may be heavy-tailed", In: *Proceedings of the 12th International Conference on Inductive Logic Programming*, Springer Verlag, July 2002.

[75] ZELLE, J., MOONEY, R., "Learning semantic grammars with constructive inductive logic programming", (Washington, D.C.), pp. 817–822, AAAI Press/MIT Press, July 1993.