Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação Automática de Páginas Web Multi-label via MDL e Support Vector Machines

Rodrigo de La Rocque Ormonde

Brasília
2009

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação Automática de Páginas Web Multi-label via MDL e Support Vector Machines

Rodrigo de La Rocque Ormonde

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Marcelo Ladeira

Brasília
2009

CIP — Catalogação Internacional na Publicação

Ormonde, Rodrigo de La Rocque.

Classificação Automática de Páginas Web Multi-label via MDL e Support Vector Machines / Rodrigo de La Rocque Ormonde. Brasília : UnB, 2009.
122 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2009.

1. Classificação de Páginas Web, 2. Classificação Multi-label,
3. Support Vector Machines, 4. SVM, 5. MDL, 6. Codificação de Huffman

CDU 004.8

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação Automática de Páginas Web Multi-label via MDL e Support Vector Machines

Rodrigo de La Rocque Ormonde

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Marcelo Ladeira (Orientador)
CIC/UnB

Nelson Francisco F. Ebecken
COPPE/UFRJ

Célia Ghedini Ralha
CIC/UnB

Li Weigang
Coordenador do Mestrado em Informática

Brasília, 28 de Agosto de 2009

Agradecimentos

Agradeço a todos os que me apoiaram ao longo de todo este período em que estive desenvolvendo esta pesquisa. Em especial, agradeço à minha esposa Roberta, que sempre me apoiou e me incentivou, mesmo quando eu não lhe podia dar a atenção que ela necessitava e merecia.

Abstract

In this research, it is developed the extension of a new classification algorithm, called CAH+MDL, previously conceived to deal only with binary or multi-class classification problems, to treat directly multi-label classification problems. Its accuracy is then studied in the classification of a database comprised of Web sites in Portuguese and English, divided into seven multi-label categories. This algorithm is based on the principle of the Minimum Description Length (MDL), used together with the Huffman Adaptive Coding. It has already been studied for binary classification in SPAM detection and has presented good results, however, to the best of my knowledge, it had never been studied before for the multi-label case, which is much more complex. In order to evaluate its performance, its results are compared with the results obtained in the classification of the same database by a linear SVM, which is the algorithm that usually presents the best results in pattern classification and, specially, in text classification.

Resumo

Nesta pesquisa é feita a extensão de um novo algoritmo de classificação, chamado de CAH+MDL, anteriormente desenvolvido para lidar apenas com problemas de classificação binários ou multiclasse, para tratar diretamente também problemas de classificação multi-label. Foi estudado então seu desempenho para a classificação de uma base de páginas Web em Português e Inglês, divididas em sete categorias multi-label. Este algoritmo é baseado no princípio da *Minimum Description Length* (MDL), utilizado juntamente com a Codificação Adaptativa de Huffman e foi anteriormente estudado para a classificação binária na detecção de SPAM, tendo apresentado bons resultados. Não foram encontradas citações na literatura, entretanto, de sua utilização para o caso multi-label, que é bem mais complexo. Para avaliar seu desempenho, os resultados são comparados com os resultados obtidos na classificação da mesma base de dados por uma SVM linear, que é o algoritmo que normalmente apresenta os melhores resultados na classificação de padrões e, especialmente, na classificação de textos.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

CAH: *Codificação Adaptativa de Huffman* (ver Seção 5.2).

DDAG: *Decision Directed Acyclic Graph*. Em português *Grafo de Decisão Acíclico Orientado* (ver Seção 2.2.3).

DF: *Document Frequency*, i.e., a freqüência que um termo aparece nos documentos de uma base de dados (ver definição 3.4).

HTML: *Hypertext Markup Language*. É a linguagem mais usada para a criação de páginas Web.

IR: *Information Retrieval*, i.e., recuperação de informação.

KKT: *Karush-Kuhn-Tucker*. São condições que possuem um papel central na teoria e prática de otimizações limitadas (ver Seção 4.6.1).

MDL: *Minimum Description Length* (ver Seção 5.1).

RBF: *Radial Basis Function*. É um tipo de kernel que pode ser utilizado com SVMs não lineares (ver Seção 4.3).

SMO: *Sequential Minimal Optimization*. É um dos algoritmos mais eficientes para treinamento de SVMs. (ver Seção 4.6.4).

SVM: *Support Vector Machine*.

TF: *Term Frequency* (ver Seção 3.1.2).

TFIDF: *Term Frequency-Inverse Document Frequency* (ver Seção 3.1.3).

URL: *Uniform Resource Locator*, i.e., Localizador de Recursos Universal. Ele representa o endereço de um recurso disponível em uma rede. No caso da Internet, ele designa um servidor e um caminho dentro deste servidor.

1. Introdução

Com o crescimento explosivo que a Internet apresenta desde seu lançamento como rede comercial e em especial nos últimos anos, estão se tornando cada vez mais necessários mecanismos que facilitem o acesso às informações desejadas, bem como aqueles que impeçam o acesso de certos usuários a sites com conteúdos considerados impróprios. Como exemplo desta última situação, está o caso de um pai que não deseja que seu filho pré-adolescente tenha acesso a sites de conteúdo pornográfico ou que estimulem a violência ou o vício.

Para desempenhar ambas as tarefas, e em especial a segunda, métodos de classificação automática de páginas Web vem sendo estudados há vários anos [DC00, KR01, GTL+02, CWZH06, CM07, DS07]. A razão básica é que o número de páginas Web disponíveis na rede mundial não pára de crescer e atinge números assombrosos, inviabilizando completamente qualquer tipo de classificação manual.

De acordo com os dados disponíveis em Setembro de 2008, o número de servidores Web na Internet ultrapassou a marca de 180 milhões [Web08]. O número de URLs diferentes indexadas pela base de dados do Google, passou de 26 milhões em 1998, para 1 bilhão em 2000 e atingiu em Julho de 2008 a marca de 1 trilhão [AH08].

Apesar de não haver nenhum estudo atualizado que apresente o número de páginas Web escritas no idioma Português, ele é o sexto de uso mais freqüente na Internet, representando 3,6% dos usuários [Wor08]. Se utilizarmos esta proporção para fazer uma estimativa, chegaríamos ao número aproximado de 36 milhões de páginas. Apesar disso, muito poucos estudos na área de classificação de textos ou de páginas Web foi feito levando-se em conta as particularidades desta língua.

1.1 Definição do Problema

Esta pesquisa aborda o problema da classificação automática de páginas Web, utilizando técnicas de aprendizagem de máquina. Será comparada a acurácia de um novo algoritmo, baseado no princípio da *Minimum Description Length (MDL)*, com o algoritmo de *Support Vector Machines* . SVM é freqüentemente apontado como o algoritmo que produz os melhores resultados na classificação de padrões e, em especial, na classificação de textos [CV95, DPHS98, YL99, Joa02, CCO03, ZL03], na classificação de sites Web em Português e Inglês.

Este problema de categorização ou simplesmente classificação, como mais freqüentemente é denominado, pode ser visto como o processo de atribuição de forma dinâmica de uma ou mais categorias para páginas da Web. Ele é normalmente classificado como aprendizado supervisionado [Mit97], no qual um conjunto de dados pré-categorizados é

utilizado para treinar um classificador, que, a partir destes dados, consegue inferir a classificação de dados futuros.

A classificação de páginas Web pode ser abordada de várias formas diferentes: através de seu conteúdo textual, imagens, hyperlinks ou combinações destes elementos, dentre outras possibilidades. Aqui o problema será abordado como uma particularização da classificação de textos, onde serão utilizados não apenas o conteúdo textual das páginas a serem classificadas, mas também informações não disponíveis nos problemas de classificação de textos em geral, como metadados e informações semi-estruturadas, de modo a melhorar a precisão do classificador.

1.2 Justificativa do Tema

Métodos de classificação automáticos, adequados para a categorização de páginas Web são cada vez mais necessários em virtude do explosivo crescimento da Internet, como mencionado na Seção 1.1.

O algoritmo SVM é considerado atualmente como o estado da arte na classificação de dados e regressão. Entretanto, este algoritmo ainda demanda grande poder computacional para realizar seu treinamento e não o realiza de forma incremental. Isso pode ser um problema no caso de classificação multi-label (ver Seção 2.1) com um grande número de categorias, já que se necessita treinar um classificador para cada uma das categorias existentes e isso pode demandar grande quantidade de tempo.

O algoritmo baseado no princípio da *Minimum Description Length* com a codificação adaptativa de Huffman (CAH+MDL) possui a vantagem de exigir poucos recursos computacionais, tanto para o treinamento quanto para a classificação de dados, e permite a realização de treinamento incremental. Este algoritmo, baseado na abordagem proposta por [BFC+06], foi utilizado com sucesso para a identificação de SPAM em [BL07], porém ele ainda não foi estudado para a classificação de dados multi-label.

1.3 Contribuição Científica Esperada

As principais contribuições científicas deste trabalho são:

- Proposição e avaliação de um classificador multi-label, baseado no algoritmo CAH+MDL, compatível com a hipótese de mundo aberto, i.e., onde não é necessária a atribuição obrigatória de ao menos uma categoria para cada documento.
- Proposição de uma extensão do algoritmo de classificação CAH+MDL para a classificação multi-label e sua avaliação na classificação de uma base de dados de sites Web, tomando como base o desempenho de um classificador baseado em *Support Vector Machines* linear.
- Criação de uma base de sites Web multi-label, em Português e Inglês, composta de sites manualmente classificados em 7 categorias, que posteriormente poderá ser utilizada na realização de experimentos de classificação em geral.

- Identificação das combinações de ponderações de termos extraídos de partes específicas de páginas Web que produzem os melhores resultados na classificação de páginas Web, tanto no algoritmo CAH+MDL quanto em uma SVM linear.
- Como contribuição secundária, é feita uma apresentação unificada das técnicas utilizadas para classificação de dados multi-label, em especial para o caso de textos e páginas Web.

1.4 Organização deste documento

Nos primeiros capítulos, é feita inicialmente uma análise do estado da arte da classificação de sites Web de forma incremental.

No Capítulo 2 é analisado o problema da classificação de padrões de forma geral, mostrando os diferentes tipos que ele pode assumir (binário, multiclasse e multi-label), as técnicas utilizadas para se decompor um problema multiclasse ou multi-label em subproblemas e as medidas de análise de acurácia de classificadores.

O Capítulo 3 descreve o problema da classificação de textos e de páginas Web, mostrando as suas principais especificidades e as técnicas que podem ser utilizadas para lidar melhor com estes problemas.

O Capítulo 4 explica em detalhes o funcionamento do algoritmo SVM, mostrando em detalhes sua evolução desde sua concepção original até a formulação atual. Também são discutidas as SVM multi-classe e os algoritmos concebidos para realizar seu treinamento.

O Capítulo 5 introduz o princípio do *Minimum Description Length*, a Codificação Adaptativa de Huffman e o funcionamento do algoritmo de classificação CAH+MDL.

Após a análise do estado da arte, apresentada nos Capítulos de 2 a 5, o Capítulo 6 descreve em detalhes o problema que será abordado neste trabalho, a metodologia utilizada, o domínio e como os resultados serão avaliados.

O Capítulo 7 descreve os resultados obtidos e a análise dos dados, enquanto que o Capítulo 8 apresenta a conclusão e descreve possíveis trabalhos futuros que podem ser desenvolvidos a partir desta pesquisa.

Por fim, o Apêndice A documenta a ferramenta, desenvolvida como parte desta pesquisa, que implementou os classificadores CAH+MDL e SVM.

2. Classificação de Dados

Neste capítulo será abordado o estado da arte sobre a resolução do problema de classificação de dados em geral. A Seção 2.1 mostra os subproblemas de classificação conhecidos enquanto a Seção 2.2 descreve técnicas que permitem que problemas mais complexos, i.e., multiclasse e multi-label, sejam resolvidos por classificadores binários através de técnicas de decomposição. A Seção 2.3 descreve abordagens para tratar o problema de classificação multi-label, alvo desta pesquisa. Por último, a Seção 2.4 descreve como os resultados dos algoritmos de classificação podem ser avaliados e comparados.

2.1 Tipos de classificação

Os problemas de aprendizagem de máquina e classificação de dados em geral podem ser divididos em três abordagens distintas [ZG09]:

1. **Supervisionada:** Neste caso, na fase de treinamento, o algoritmo de aprendizagem recebe como entrada um conjunto de dados pré-classificados (geralmente por algum especialista humano), onde cada exemplo está associado a um ou mais rótulos. Finalizado o treinamento, o algoritmo consegue generalizar a partir dos exemplos recebidos e pode, com algum grau de confiança, associar rótulos a novos dados, até então desconhecidos.

2. **Não-Supervisionada:** Neste tipo de treinamento, o algoritmo recebe um conjunto de dados sem nenhum rótulo e seu objetivo é determinar como estes dados estão organizados. Tipicamente esta abordagem é utilizada para a realização de agrupamento (*clustering*) de dados.

3. **Semi-Supervisionada:** Esta abordagem é um meio termo entre a supervisionada e a não supervisionada. A idéia é que o algoritmo receba como treinamento um pequeno conjunto de dados rotulados e um conjunto grande sem rótulos. Vários algoritmos já foram propostos para que com o auxílio dos dados não rotulados se produza um aumento da acurácia do classificador em comparação ao caso onde apenas o pequeno conjunto já classificado fosse usado. Informações sobre este tópico podem ser encontradas em [BM98, LDG00, LLYL02, YHcC04].

Nesta pesquisa será abordado apenas o caso da aprendizagem supervisionada, que pode ser enunciada, de maneira genérica, como se segue:

Seja um conjunto de p exemplos, extraídos de forma independente e identicamente distribuídos (i.i.d), de acordo com uma distribuição de probabilidade desconhecida mas

fixa, a serem utilizados para treinar o algoritmo de aprendizagem. Estes exemplos são informados ao algoritmo associados, cada um, a um conjunto de rótulos, atribuídos por um especialista humano:

$$(x_1, y_1), (x_2, y_2), \cdots, (x_p, y_p) \tag{2.1}$$

Em sua forma mais freqüente, cada exemplo x_i consiste de um vetor que descreve um determinado documento. O formato dos rótulos y_i varia de acordo com o tipo de classificação a ser efetuada, conforme descrito nas seções a seguir, porém ele sempre indica a(s) classe(s) que um documento pertence.

Definição 2.1. Uma *classe* consiste em uma abstração para um grupo de documentos que contém determinadas características em comum. Cada classe é representada, para fins de classificação, por um valor discreto distinto.

2.1.1 Binária

O problema de classificação binária é o mais simples de todos, porém ainda assim é considerado o mais importante. Com algumas pequenas suposições, qualquer um dos problemas mais complexos pode ser reduzido a ele (veja a Seção 2.2).

No caso binário, existem exatamente duas classes. Assim sendo, os rótulos são definidos da seguinte maneira: $y_i \in \{-1, 1\}$ (outros valores como $y_i \in \{0, 1\}$ poderiam ser usados, mas nesta pesquisa se supõe sempre a primeira forma).

A interpretação do significado, entretanto, de cada uma das duas classes pode variar bastante. Um exemplo bastante usual é o caso da filtragem de SPAM, onde o algoritmo teria que decidir se uma determinada mensagem é ($y = 1$) ou não SPAM ($y = -1$). Para filtragem de páginas Web, poderíamos interpretar uma das categorias como permitida e outra como não permitida.

2.1.2 Multiclasse

Muitos dos problemas de classificação envolvem mais de uma classe. Podemos facilmente pensar no caso de páginas Web onde poderíamos ter, por exemplo, as categorias de "Material Adulto", "Páginas Infantis" e "Esportes". Uma classificação de documentos textuais também poderia estar agrupada por assunto, de forma semelhante à classificada em uma biblioteca, onde cada assunto estaria atribuído a exatamente uma categoria diferente.

No caso geral, vamos considerar que existam p categorias diferentes. Desta forma, sem perda de generalidade, vamos assumir que $y_i \in \{1, \cdots, p\}$. As categorias são assumidas como independentes e isso significa que a ordem na qual elas se apresentam é arbitrária.

Existem algumas versões de algoritmos de classificação, em especial de *Support Vector Machines* (leia a Seção 4.5), que permitem que se resolva este problema diretamente, entretanto, em geral elas são computacionalmente ineficientes [Joa02] e ainda são objetos de estudos mais aprofundados [Gue07].

O mais usual para se resolver problemas de classificação multiclasse (e também multi-label) é decompor o problema em vários subproblemas de classificação binária. Isso está descrito na Seção 2.2.

2.1.3 Multi-label

Vários dos problemas de classificação, especialmente textuais e de imagens, são problemas multi-label, ou seja, que podem pertencer a mais de uma categoria simultaneamente. Por exemplo, um artigo de jornal falando sobre o jogo Brasil X Alemanha na final da copa do mundo poderia estar nas categorias "Esportes", "Brasil" e "Alemanha", asim como uma reportagem sobre um novo pacote econômico lançado por um determinado governo poderia pertencer as categorias de "Política" e "Economia".

Formalmente, a definição de multi-label significa que, ao invés de estar restrito a uma única categoria, cada documento pode pertencer a várias, a uma ou a nenhuma. Sendo assim, temos que para um conjunto de p categorias, os rótulos pertenceriam ao conjunto das partes deste conjunto de rótulos, i.e., $y_i \in 2^{\{1,\cdots,p\}}$.

Apesar de serem muito freqüentes, problemas deste tipo foram menos estudados que os dois anteriores e são considerados como mais complexos. Desta forma, embora existam alguns poucos algoritmos que possam tratar o caso multi-label diretamente, como os propostos por [dCGT03] (decision-trees), [Zha06] (redes-neurais) e [ZZ07] (K-NN), eles ainda são objetos de estudos recentes e não foram utilizados em problemas práticos.

Desta forma, o método mais comum de resolução de problemas de classificação multi-label é o de separação do classificador geral em vários sub-classificadores, cada um capaz de identificar uma determinada classe. Isso está descrito na Seção 2.2.

2.2 Métodos de Decomposição

Esta seção descreve os métodos que podem ser utilizados para quebrar um problema de classificação multiclasse ou multi-label em vários problemas menores, tratados por classificadores binários. Isso é útil quando o algoritmo de classificação a ser utilizado não suporta diretamente o problema de classificação da forma como é especificado ou quando, para que ele suporte, são necessárias modificações que o tornam ineficiente sob algum aspecto.

2.2.1 Um contra todos

Este é o mais simples e antigo método de decomposição de problemas. Ele consiste na criação de k classificadores binários, cada um capaz de identificar uma classe em relação às demais. Desta forma, para se classificar um determinado documento, ele é apresentado aos k classificadores e o(s) rótulo(s) atribuído(s) ao documento vão depender do tipo de problema de classificação:

- Multiclasse: Neste caso, deve-se atribuir um único rótulo ao documento. Normalmente usa-se a abordagem de atribuir ao documento a classe cujo respectivo classificador produziu a maior nota.
- Multi-label: Para este tipo de problema, em geral usa-se a abordagem de atribuir ao documento todas as classes que foram reconhecidas pelos k classificadores. Caso nenhuma tenha sido identificada, o documento será marcado como "desconhecido". Uma variação deste método é atribuir no mínimo uma classe a todo documento,

mesmo que nenhuma tenha sido identificada. Neste caso, a classe com a maior nota será utilizada.

Apesar deste método de decomposição ter sido criticado por alguns autores por apresentar resultados inferiores aos de outros métodos [Für02, wHjL02], ele foi analisado posteriormente por Rifkin e Klautau [RK04], que concluíram que ele apresenta resultados estatísticos tão bons quanto os demais métodos de decomposição, desde que usado com SVMs bem parametrizadas. Uma importante ressalva em relação a estes estudos, entretanto, é que todos foram realizados apenas para a classificação multiclasse, não tendo, nenhum deles, analisado as particularidades dos problemas multi-label.

A principal vantagem desta abordagem em relação à "Um contra um" e "Grafos de Decisão Acíclicos Orientados" é que o número de classificadores a serem treinados é linear em relação ao número de classes, enquanto nos outros casos é quadrático. Além disso, a decomposição "DDAG" não pode ser usada no caso multi-label e a "Um contra um", apesar de poder ser utilizada, precisa de uma definição melhor de como as classes serão atribuídas aos documentos.

2.2.2 Um contra um

Este método de decomposição, originalmente desenvolvido por Knerr et al. [KPD90], consiste em utilizar um classificador para cada dupla de categorias, i.e., o classificador $K(i,j), 1 \leq i < j \leq k$ saberia distinguir a i-ésima categoria da j-ésima. Desta forma, cada classificador $K(i,j)$ seria treinado apenas com dados das classes i e j, produzindo, portanto, $\frac{k(k-1)}{2}$ classificadores distintos.

Para se classificar um determinado documento, ele é apresentado aos $\frac{k(k-1)}{2}$ classificadores e o(s) rótulo(s) atribuído(s) ao documento vão depender do tipo de problema de classificação:

- Multiclasse: Neste caso, deve-se atribuir um único rótulo ao documento. Normalmente usa-se a estratégia "Max Wins", onde se atribui ao documento a classe que obteve a maior soma dos votos dos classificadores.
- Multi-label: Não existem estudos conhecidos tratando especificamente a distribuição de rótulos para a decomposição "Um contra um" em um problema de classificação multi-label. Uma das possibilidades é definir um número mínimo de votos para que um dado rótulo seja atribuído a um documento. Outra é atribuir a um documento todos os rótulos que estiverem a uma distância mínima do mais votado. De qualquer forma, este tópico ainda é um estudo em aberto.

O principal problema deste método de decomposição é que ele cresce de forma quadrática em relação ao número de categorias. Desta forma, se este número for alto, sua utilização se torna inviável. Além disso, conforme apontado por Platt et al. [PCSt00], se os classificadores individuais não forem "cuidadosamente" regularizados (como são no caso das SVMs), o sistema de classificadores tende a produzir *overfitting*.

2.2.3 Grafos de Decisão Acíclicos Orientados

Esta abordagem foi proposta por Platt et al. [PCSt00] e consiste na disposição de classificadores em um grafo binário, direcionado, acíclico e com uma raiz, i.e., um nó único para o qual não existem arcos apontando. Para a criação deste grafo, inicialmente se decompõe o problema de classificação em $\frac{k(k-1)}{2}$ classificadores distintos, onde cada classificador é capaz de distinguir entre duas classes, da mesma forma que descrito na decomposição "Um contra um" (Seção 2.2.2).

Para se avaliar o resultado de classificação de um determinado documento, inicia-se no nó raiz e avalia-se a função de decisão: se ela for negativa prossegue-se pela aresta esquerda do grafo, caso contrário prossegue-se pela direita. No nó seguinte do grafo o procedimento é repedido, até que se chegue a uma de suas folhas. O valor do rótulo associado ao documento será o valor referente à folha atingida.

Um grafo, retirado de [PCSt00], para a separação de 4 classes está mostrado na Figura 2.1.

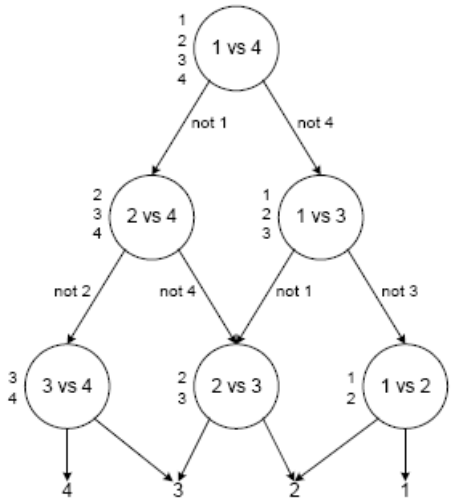


Figura 2.1: Grafo de decisão para encontrar a melhor entre 4 classes

A vantagem deste método é que o número de funções de decisão consultadas para a classificação de um determinado documento é igual a $k-1$, onde k é o número de possíveis classes, já que para cada classe apenas um classificador é consultado. Já na estratégia "Um contra um" o número de classificadores consultados é $\frac{k(k-1)}{2}$. A "Um contra todos" consulta k classificadores, porém como cada um deles foi treinado com mais dados, pode ocorrer (dependendo do algoritmo utilizado) que o tempo levado para avaliar todas estas k funções de decisão seja maior que no caso dos grafos de decisão.

O grande problema, entretanto, do uso desta estratégia é que com ela não é possível se resolver problemas de classificação multi-label, apenas multiclasse.

2.3 Treinamento de Classificadores Multi-label

Normalmente, exceto para os poucos algoritmos que o suportem diretamente, um problema de classificação de documentos multi-label é separado em vários sub-problemas,

conforme mostrado na Seção 2.2. A questão é decidir como realizar o treinamento destes sub-classificadores, de modo a resolver o problema inicial de classificação e obter os melhores resultados possíveis.

Esta seção descreve as várias abordagens diferentes que existem para realizar esta classificação, conforme levantado por [BLSB04, TK07]. Para melhor exemplificar cada um dos métodos, será utilizada a Tabela 2.1, que descreve a distribuição de rótulos em uma hipotética base de dados de documentos multi-label.

	Classe A	Classe B	Classe C	Classe D
Doc. 1	√			√
Doc. 2			√	√
Doc. 3	√			
Doc. 4		√	√	
Doc. 5	√	√	√	

Tabela 2.1: Exemplo de um conjunto de documentos multi-label com 4 classes

2.3.1 Rótulo Único

Esta abordagem, que é uma das primeiras a serem usadas, consiste em transformar os dados a serem treinados de multi-label para multiclasse, i.e., cada documento com um rótulo único. Desta forma, para cada documento que possui mais de um rótulo, apenas um deles é escolhido para lhe ser atribuído. Esta escolha pode ser feita de modo arbitrário ou através de algum critério, possivelmente subjetivo, que caracteriza determinado documento como mais característico de uma classe que de outra.

Usando esta abordagem, o conjunto de documentos da Tabela 2.1 seria transformado, entre outras possibilidades, na Tabela 2.2, mostrada a seguir:

	Classe A	Classe B	Classe C	Classe D
Doc. 1	√			
Doc. 2				√
Doc. 3	√			
Doc. 4		√		
Doc. 5			√	

Tabela 2.2: Uma possível transformação da Tabela 2.1 de multi-label para multiclasse

Esta abordagem possui dois grandes problemas:

1. Muitas vezes é complicado se definir um critério de atribuição da classe única para cada documento, já que, quase sempre, ele é subjetivo. Além disso, se a atribuição for aleatória ou automática (sempre pegar a menor classe, por exemplo), a distribuição de classes nos documentos pode ser seriamente afetada.

2. Se boa parte dos documentos a serem classificados possuir mais de uma classe, esta abordagem vai descartar uma boa quantidade da informação presente no problema original.

Na prática, esta abordagem só pode ser utilizada em conjuntos onde um percentual bastante pequeno de documentos possui mais de um rótulo (ou seja, seriam na verdade problemas de classificação multiclasse com alguns poucos documentos multi-label).

2.3.2 Ignorar Multi-label

Esta abordagem também visa transformar um problema multi-label em multiclasse, como a de rótulo único (Seção 2.3.1), porém a forma de se realizar esta transformação é mais radical: todos os documentos multi-label são simplesmente descartados ao se realizar o treinamento.

Usando esta abordagem, o conjunto de documentos da Tabela 2.1 seria transformado na Tabela 2.3, mostrada a seguir:

	Classe A	Classe B	Classe C	Classe D
Doc. 3	$\surd$			

Tabela 2.3: Transformação da Tabela 2.1 de multi-label para multiclasse ignorando os documentos multi-label

O grande problema desta abordagem é que ela produz resultados desastrosos se os documentos a serem treinados possuirem em sua maioria mais de um rótulo. Este é o caso do exemplo da Tabela 2.3 onde praticamente todos os documentos seriam eliminados, já que ele é um conjunto onde quase todos os documentos possuem mais de um rótulo.

Na prática, ainda mais que à de rótulo único, esta abordagem só pode ser utilizada em conjuntos onde um percentual insignificante de documentos possui mais de um rótulo (ou seja, seriam na verdade problemas de classificação multiclasse com muito poucos documentos multi-label).

2.3.3 Nova Classe

Este método também visa transformar um problema de classificação multi-label em multiclasse, como os dois apresentados anteriormente, porém seu objetivo é fazer esta transformação sem perda de informação. Desta forma, ao invés de atribuir uma determinada classe para cada documento multi-label ou simplesmente descartá-los (como nos métodos anteriores), a idéia desta abordagem é criar uma nova classe para cada interseção de classes existente no conjunto de documentos original. Estas novas classes seriam então treinadas da mesma forma que as originalmente existentes.

Com esta abordagem, o conjunto de documentos da Tabela 2.1 seria transformado na Tabela 2.4:

Esta abordagem possui dois grandes problemas:

1. Muito comumente os dados pertencentes às novas classes criadas são demasiadamente esparsos, i.e., muito pouco freqüentes, para que se possa utilizá-los para o treinamento. Neste caso, o efeito prático resultante desta abordagem seria o mesmo de que ignorar os documentos multi-label ou, pelo menos, um subconjunto deles.

	Classe A	Classe A+D	Classe C+D	Classe B+C	Classe A+B+C
Doc. 1		√			
Doc. 2			√		
Doc. 3	√				
Doc. 4				√	
Doc. 5					√

Tabela 2.4: Transformação da Tabela 2.1 de multi-label para multiclasse, criando uma nova classe para cada intersecção de classes existente na tabela original

2. Ainda que os dados não sejam esparsos, pode ocorrer de existirem muitas interseções distintas entre classes. Desta forma, o número total de classes a serem treinadas pode aumentar exponencialmente, inviabilizando o uso desta abordagem.

Na prática, esta abordagem pode ser utilizada sempre que o número de interseções entre classes do conjunto de treinamento não seja demasiadamente grande e que exista um número significativo de exemplos em cada uma destas interseções, para que o treinamento possa ser realizado.

2.3.4 Treinamento Cruzado

Esta abordagem, que visa resolver os problemas das abordagens anteriores e evitar a perda de informação, foi proposta por Boutell et al. [BLSB04] e, segundo seus estudos, apresentou melhores resultados do que as anteriores na classificação de imagens multi-label por Support Vector Machines, apesar da diferença em geral ser pequena (em torno de 2 pontos percentuais) em relação aos demais métodos.

O treinamento cruzado consiste em dividir o conjunto de dados em k subconjuntos, onde k é o número de classes distintas, e realizar o treinamento separado de k classificadores binários. Documentos serão usados como exemplos positivos durante o treinamento das classes às quais eles fazem parte e como exemplos negativos para as demais.

Com esta abordagem, o conjunto de documentos da Tabela 2.1 seria transformado em 4 subconjuntos, um para realizar o treinamento independente de cada uma das classes originais, conforme mostrado na Tabela 2.5:

	Classe A	¬ Classe A
Doc. 1	√	
Doc. 2		√
Doc. 3	√	
Doc. 4		√
Doc. 5	√	

a) Treinamento para o classificador 1

	Classe B	¬ Classe B
Doc. 1		√
Doc. 2		√
Doc. 3		√
Doc. 4	√	
Doc. 5	√	

b) Treinamento para o classificador 2

Esta abordagem também pode ser utilizada da mesma forma para a separação de um problema de classificação multiclasse em vários sub-classificadores binários. A única diferença é que na classificação multiclasse é necessário escolher o rótulo reportado por

	Classe C	¬ Classe C
Doc. 1		√
Doc. 2	√	
Doc. 3		√
Doc. 4	√	
Doc. 5	√	

c) Treinamento para o classificador 3

	Classe D	¬ Classe D
Doc. 1	√	
Doc. 2	√	
Doc. 3		√
Doc. 4		√
Doc. 5		√

d) Treinamento para o classificador 4

Tabela 2.5: Separação do conjunto original da Tabela 2.1 em 4 subconjuntos para o treinamento de classificadores independentes

um dos classificadores como o que será atribuído para o documento sendo classificado, enquanto no multi-label se necessita um método para escolher o conjunto de rótulos a atribuir a tal documento.

2.4 Medidas de Desempenho

Para a avaliação do desempenho de classificadores, não se pode simplesmente utilizar o número de erros como uma medida. Isso é causado em virtude de que freqüentemente as bases a serem classificadas são desbalanceadas. Sendo assim, um suposto classificador que respondesse sempre -1 (ou não pertence) para cada documento a ser classificado e tivéssemos uma base onde apenas 10% dos documentos efetivamente pertencessem à dada classe, teríamos um classificador com uma excelente taxa de acertos, 90%, porém que é completamente inútil.

Desta forma, para a avaliação da acurácia de classificadores, tradicionalmente utilizam-se as medidas de Precisão, Recall e F-Measure, que é uma média harmônica ponderada das duas primeiras. Aqui serão apresentadas as definições tradicionais, que em seguida serão estendidas, de acordo com [GS04], para tratar o caso de classificação multi-label.

Para a definição das medidas em sua forma tradicional, binária, será usada a Tabela 2.6, que é normalmente conhecida como "Matriz de Confusão". Nesta tabela, a diagonal principal representa os acertos, enquanto as demais células representam os erros de classificação.

	Rótulo = 1	Rótulo = -1
Previsão = 1	V_P (Verdadeiros Positivos)	F_P (Falsos Positivos)
Previsão = -1	F_N (Falsos Negativos)	V_N (Verdadeiros Negativos)

Tabela 2.6: Matriz de Confusão de uma classificação binária

2.4.1 Precisão

Precisão é a probabilidade de um dado documento, escolhido aleatoriamente, que esteja classificado como pertencente a uma dada categoria, seja realmente membro desta categoria.

Algebricamente, para o caso binário, ela pode ser definida para uma determinada categoria da seguinte forma:

$$\text{Precisão } = \frac{V_P}{V_P + F_P} \tag{2.2}$$

Para o caso multi-label, é importante considerar que não existem apenas acerto e erro absolutos, como no caso binário ou multiclasse, e sim uma graduação de erros. Como ilustração está o caso da classificação de um hipotético documento pertencente às classes 1 e 2, no qual alguns possíveis resultados desta classificação estão mostrados na Tabela 2.7:

$f(x) = \{1, 2\}$	(correto)
$f(x) = \{1\}$	(parcialmente correto)
$f(x) = \{2\}$	(parcialmente correto)
$f(x) = \{1, 3\}$	(parcialmente correto)
$f(x) = \emptyset$	(incorreto)
$f(x) = \{3, 4\}$	(incorreto)

Tabela 2.7: Resultados possíveis da classificação de um documento pertencente às classes 1 e 2

Desta forma, estendemos a definição da Precisão para o caso da classificação multi-label de uma determinada categoria da seguinte forma:

$$\text{Precisão } = \frac{1}{n} \sum_{i=1}^{n} \frac{\mid y_i \cap z_i \mid}{\mid z_i \mid} \tag{2.3}$$

Onde n é o número de documentos classificados, $y_i \subseteq 2^{\{1,\cdots,p\}}$ é o conjunto de classes atribuído ao i-ésimo documento e $z_i \subseteq 2^{\{1,\cdots,p\}}$ é o conjunto de classes previsto pelo classificador para o i-ésimo documento.

Assim sendo, para a Tabela 2.7, teríamos os seguintes valores de Precisão:

$f(x) = \{1, 2\}$	(Precisão = 1)
$f(x) = \{1\}$	(Precisão = 1)
$f(x) = \{2\}$	(Precisão = 1)
$f(x) = \{1, 3\}$	(Precisão = 0,5)
$f(x) = \emptyset$	(Precisão = 0)
$f(x) = \{3, 4\}$	(Precisão = 0)

Tabela 2.8: Resultados de Precisão para a Tabela 2.7

2.4.2 Recall

Recall é a probabilidade de um documento qualquer, escolhido aleatoriamente, que pertença a uma determinada categoria seja classificado como realmente pertencente a esta categoria.

Em Português, este termo é algumas vezes traduzido como “Abrangência”, “Cobertura” ou “Revocação”. Como não existe uma nomenclatura padrão, para evitar confusão, será utilizado sempre o termo na sua formulação original, em Inglês.

Algebricamente, para o caso binário ou multiclasse, Recall pode ser definido para uma determinada categoria da seguinte forma:

$$\text{Recall } = \frac{V_P}{V_P + F_N} \tag{2.4}$$

Para o caso multi-label, novamente é importante considerar que não existem apenas acerto e erro absolutos, como no caso binário, e sim uma graduação de erros, conforme ilustrado na Tabela 2.7.

Desta forma, estendemos a definição de Recall para o caso da classificação multi-label de uma determinada categoria da seguinte forma:

$$\text{Recall } = \frac{1}{n}\sum_{i=1}^{n}\frac{\mid y_i \cap z_i \mid}{\mid y_i \mid} \tag{2.5}$$

Onde n é o número de documentos classificados, $y_i \subseteq 2^{\{1,\cdots,p\}}$ é o conjunto de classes atribuído ao i-ésimo documento e $z_i \subseteq 2^{\{1,\cdots,p\}}$ é o conjunto de classes previsto pelo classificador para o i-ésimo documento.

Assim sendo, para a Tabela 2.7, teríamos os seguintes valores de Recall:

$f(x) = \{1, 2\}$	(Recall = 1)
$f(x) = \{1\}$	(Recall = 0,5)
$f(x) = \{2\}$	(Recall = 0,5)
$f(x) = \{1, 3\}$	(Recall = 0,5)
$f(x) = \emptyset$	(Recall = 0)
$f(x) = \{3, 4\}$	(Recall = 0)

Tabela 2.9: Resultados de Recall para a Tabela 2.7

2.4.3 F-Measure

Apesar das medidas de Precisão e Recall descreverem com exatidão o desempenho de um classificador, freqüentemente se necessita de um único número para melhor poder comparar dois classificadores quaisquer. Assim sendo, Van Rijsbergen [Rij79] criou a F-Measure, que é uma média harmônica ponderada dos valores de Precisão e Recall, definida da seguinte forma:

$$F_\beta = \frac{(1 + \beta^2) \cdot Precisão \cdot Recall}{\beta^2 \cdot Precisão + Recall} \tag{2.6}$$

Em (2.6), o valor de β corresponde ao maior peso que se deseja atribuir a Precisão ou ao Recall. Para $\beta > 1$, Recall tem uma importância maior, enquanto que para $0 < \beta < 1$, a Precisão recebe maior peso.

Um caso particular e o mais comum da F-Measure é quando se resolve atribuir o mesmo peso para Precisão e Recall, passando a chamá-la de F_1. Sua definição é a seguinte:

$$F_1 = \frac{2 \cdot Precisão \cdot Recall}{Precisão + Recall} \tag{2.7}$$

Outros valores também encontrados com alguma freqüência para F-Measure são a F_2, que atribui o dobro de peso ao Recall em relação à Precisão e a $F_{0,5}$, que atribui duas vezes mais peso à Precisão em relação ao Recall.

3. Classificação de textos e páginas Web

Neste capítulo será abordado o estado da arte da classificação de textos e de páginas Web, que são subproblemas da classificação geral de dados, discutida no capítulo anterior. Serão mostradas as particularidades da classificação textual e como cada uma delas pode ser abordada.

O capítulo começa com a descrição da maneira pela qual os documentos textuais podem ser representados e tratados pelos algoritmos de classificação (Seção 3.1). A seguir, a Seção 3.2 descreve algumas propriedades importantes da classificação de textos, que têm que ser observadas por qualquer algoritmo de classificação. A seguir, a Seção 3.3 mostra algumas técnicas de pré-processamento que podem ser aplicadas a textos em geral, com o objetivo de aumentar a acurácia dos classificadores, enquanto a Seção 3.4 apresenta métodos de seleção de atributos, que normalmente são utilizados para reduzir as dimensões dos documentos e prevenir a ocorrência de *overfitting*. A Seção 3.5 descreve a ponderação de termos, que pode ser utilizada para aumentar a importância de determinadas palavras ou termos oriundos de partes específicas de um documento (de um resumo, no caso de documentos que o contenham, ou de palavras-chave, no caso de páginas Web, por exemplo). Por último, a seção 3.6 descreve as particularidades e o estado da arte na classificação de páginas Web.

3.1 Representação de documentos textuais

Antes que qualquer documento possa ser classificado, ele deve ser transformado em alguma representação que possa ser utilizada pelos algoritmos classificadores. Existem várias maneiras de se fazer esta representação e a mais simples e substancialmente mais utilizada delas é conhecida como *Vector Space Model* (VSM) ou *bag-of-words*, que será descrita nesta seção.

Apesar de outras representações mais sofisticadas que agreguem maior conteúdo semântico do documento original serem possíveis, nenhuma dela apresentou até o momento avanços significativos e consistentes. Além disso, apesar de poderem capturar um maior significado do documento, sua complexidade adicional degrada a qualidade dos modelos estatísticos baseados nelas. Desta forma, a representação *bag-of-words* apresenta um bom compromisso entre expressividade e complexidade dos modelos [Joa02].

A representação *bag-of-words* pode ser vista como uma sacola onde elementos repetidos normalmente são permitidos (exceto no caso da representação binária, discutida na Seção 3.1.1). Desta forma, um documento é representado como uma coleção das palavras que ele

contém, ignorando-se a ordem na qual estas palavras aparecem e os sinais de pontuação existentes.

É evidente que esta representação se traduz na perda de informação léxica e/ou gramatical, bem como na impossibilidade de se desambiguar palavras homônimas ou usadas em sentido figurado. Ainda assim, assume-se que o número de tais palavras seja pequeno o suficiente para não gerar ruídos que possam impactar de forma significativa o algoritmo de classificação.

Para formalizar a representação *bag-of-words* , são necessárias as seguintes definições:

Definição 3.1. Uma *palavra* ou *termo* consiste em uma seqüência contígua de caracteres de um determinado alfabeto. Espaços, sinais de pontuação e demais caracteres que não fazem parte do alfabeto são considerados como separadores das palavras.

Definição 3.2. Um *dicionário* consiste em todas as palavras válidas e que poderão ser utilizadas para a classificação de uma determinada base de dados de documentos.

O dicionário pode ser pré-definido, porém mais comumente é gerado a partir das palavras presentes nos documentos usados para o treinamento do algoritmo. Sendo assim, se determinada palavra não estiver presente em nenhum documento usado no treinamento ela será ignorada no momento da classificação.

Definição 3.3. Uma *representação vetorial de um documento* consiste em um vetor de dimensões iguais ao número de palavras presentes no dicionário, onde cada possível palavra é representada em uma dimensão diferente. A ausência de uma palavra no documento sendo representado implica em que a dimensão correspondente a ela receba o valor 0, enquanto a presença de uma ou mais instâncias de uma palavra faz com que a dimensão correspondente receba um valor positivo.

Definição 3.4. A *freqüência de documentos*, ou *DF*, para uma determinada palavra representa a fração de documentos nos quais a palavra aparece. Ela é dada pela fórmula $D_f = \frac{N_p}{N_t}$, onde N_p é o número de documentos nos quais a palavra aparece e N_t é o número total de documentos.

Com estes conceitos definidos, serão mostradas agora as diferentes sub-formas de representação que podem ser usadas dentro da representação *bag-of-words* .

3.1.1 Binária

Neste tipo de representação, apenas a presença ou ausência de uma determinada palavra é determinante para a classificação de um documento. Desta forma, os valores possíveis para cada dimensão do vetor que representa um documento são 0, caso a palavra correspondente não esteja presente, e 1, caso existam no documento uma ou mais palavras correspondentes à dimensão.

Formalmente, o valor binário de um termo t_i em um documento d_j é o seguinte:

$$Bin(t_i) = \begin{cases} 0, & \text{se } t_i \text{ não ocorre em } d_j \\ 1, & \text{caso contrário} \end{cases} \tag{3.1}$$

3.1.2 Freqüência dos termos

Nesta representação, cada dimensão do vetor de representação de um documento contém o número de vezes que a palavra correspondente apareceu em tal documento. Esta é a maneira de representação mais comumente utilizada para a classificação de sites Web, porém a representação TFIDF (Seção 3.1.3) é mais comumente utilizada para a classificação de documentos textuais puros.

Formalmente, o valor TF de um termo t_i em um documento d_j é o seguinte:

$$TF(t_i) = N(t_i, d_j) \tag{3.2}$$

Onde $N(t_i, d_j)$ representa o número de vezes que t_i ocorreu em d_j.

3.1.3 Freqüência dos termos - inversa dos documentos

Esta á a representação mais comum para a classificação de documentos textuais. Ela leva em conta não apenas a freqüência de cada termo em um documento, porém também o número de documentos em que cada um destes termos aparecem.

A idéia intuitiva por trás desta representação é a seguinte [Seb99]:

- Quanto mais freqüente um termo aparecer em um documento, mais representativo de seu conteúdo ele é.
- Em quantos mais documentos um termo aparecer, menos descriminante ele é.

A fórmula mais usual para o cálculo de TFIDF de um termo t_i para um documento d_j é a seguinte:

$$TFIDF(t_i, d_j) = N(t_i, d_j) \cdot \log(\frac{N_t}{N_{t_i}}) \tag{3.3}$$

Onde $N(t_i, d_j)$ representa o número de vezes que t_i ocorreu em d_j, N_t é o número total de documentos e N_{t_i} é o número de documentos nos quais t_i ocorreu ao menos uma vez.

Outra fórmula, chamada de coleção de freqüência inversa probabilística, que também pode ser usada para o cálculo de TFIDF é a seguinte [Joa02]:

$$TFIDF(t_i, d_j) = N(t_i, d_j) \cdot \log(\frac{N_t - N_{t_i}}{N_{t_i}}) \tag{3.4}$$

Em (3.4), novamente termos que aparecem em muitos documentos recebem pesos menores.

3.2 Propriedades da Classificação de Textos

Algumas propriedades da classificação de documentos textuais serão mostradas nesta seção. Elas são comuns a todos os documentos de texto, em maior ou menor escala, dependendo do tipo de documento analisado.

3.2.1 Alta Dimensionalidade

A representação de documentos textuais sob a forma de vetores, conforme mostrado na Seção 3.1, faz com a dimensão destes vetores atinja facilmente o valor de dezenas de milhares. Caso se esteja trabalhando com documentos longos e escritos em diferentes línguas, o tamanho da dimensão dos vetores pode até mesmo ultrapassar a centena de milhar. É possível se reduzir um pouco esta dimensão através de técnicas de pré-processamento (Seção 3.3) ou seleção de atributos (Seção 3.4), porém ainda assim seu valor será bastante elevado.

Desta forma, ao se escolher um algoritmo de classificação para textos, deve-se assegurar que ele consiga tratar vetores com alta dimensionalidade ou então fazer um trabalho agressivo de redução de atributos, de modo a impedir a ocorrência de *overfitting*ou estouro da capacidade de memória.

3.2.2 Vetores Esparsos

Apesar dos vetores de representação de documentos serem de dimensões muito grandes, a ampla maioria das dimensões possui valor 0, ou seja, correspondem a palavras que não estão presentes no documento. Isso ocorre em virtude do número de palavras distintas em um único documento ser em geral medido na ordem de centenas, ou seja, entre duas e três ordens de grandeza menor que a dimensão dos vetores. Joachims [Joa02] mediu o número médio de palavras únicas por documento de três bases de dados textuais conhecidas e obteve os seguintes dados:

- Ohsumed [HBLH94]: Média de 100 palavras distintas por documento.
- Reuters [Lew97]: Média de 74 palavras distintas por documento.
- Webkb [DLR00]: Média de 130 palavras distintas por documento.

Apesar destas bases serem um pouco antigas, nada indica que o número de palavras únicas por documento de bases mais atuais seja substancialmente diferente. Sendo assim, é interessante que o algoritmo de classificação utilizado possa usar esta alta esparcidade em seu favor, de modo a melhorar sua velocidade e diminuir a necessidade de uso de memória.

3.2.3 Complexidade Lingüística

As línguas naturais são bastantes ricas em termos de sinônimos, figuras de linguagens e ambigüidade. É, portanto, perfeitamente possível se selecionar diferentes artigos, páginas Web, mensagens e outros tipos de documentos textuais que tratem de um mesmo tema e ainda assim possuam poucos termos comuns, exceto por aqueles termos extremamente freqüentes e que são considerados *stopwords* (ver Seção 3.3.3), que não servem como discriminantes de categorias. Isso também pode ocorrer com documentos que sejam a tradução um do outro: ambos tratam exatamente do mesmo tema porém não têm (praticamente) nenhuma palavra em comum.

Uma situação oposta, mas também bastante comum é a chamada de ambigüidade léxica, i.e., uma mesma palavra possuir diferentes significados, dependendo do contexto ou

da forma com que é empregada. Isso pode ocorrer também entre línguas: uma palavra ser escrita de forma igual em dois idiomas quaisquer, porém tendo significados completamente distintos.

Assim sendo, é importante que um bom algoritmo de classificação de textos consiga trabalhar com esta complexidade diretamente sem a exigência que algum tipo de seleção de atributos (ver Seção 3.4) seja realizado a priori, já que, em virtude do exposto acima, isso pode diminuir bastante sua acurácia.

3.2.4 Alta Redundância

As linguagens naturais são extremamente redundantes. É perfeitamente possível que um ser humano compreenda uma frase ou um parágrafo mesmo que determinadas palavras lhes sejam removidas ou que não sejam entendidas (no caso de um idioma estrangeiro ou em uma ligação telefônica com ruído, por exemplo). Desta forma, é evidente que os vetores de representação de documentos possuirão várias dimensões com distribuições idênticas ou muito semelhantes.

É fundamental, portanto, que um bom algoritmo de classificação de textos consiga trabalhar com esta redundância sem a necessidade de fortes suposições a respeito da independência entre termos (que é o principal problema do Naïve Bayes e de outros algoritmos probabilísticos).

3.3 Pré-processamento

Nesta seção serão mostradas técnicas simples que podem ser aplicadas aos documentos textuais antes deles serem repassados ao algoritmo de classificação, com o objetivo de melhorar sua eficiência de processamento ou produzir ganho em sua acurácia.

3.3.1 Normalização

Um importante aspecto para melhorar a representação VSM é assegurar que documentos contendo palavras semanticamente equivalentes sejam mapeados para vetores similares [STC04]. Na representação de documentos utilizando Freqüência dos termos (Seção 3.1.2) ou Freqüência dos termos - inversa dos documentos (Seção 3.1.3), é natural que documentos grandes apresentem maiores freqüências de palavras e, portanto, vetores com normas maiores. Isso pode induzir a erros de classificação.

No caso da classificação textual, como o tamanho de um documento não é relevante para sua classificação em uma ou outra categoria, podemos anular seu efeito normalizando os vetores que representam cada documento. Isso pode ser feito dentro do próprio kernel de uma SVM não linear (ver Seção 4.3) ou, mais comumente, como uma transformação inicial que pode ser aplicada a qualquer algoritmo.

A normalização de um vetor é expressa pela seguinte equação:

$$\vec{x}_{normal} = \frac{\vec{x}}{\|\vec{x}\|} = \frac{\vec{x}}{\sqrt{\vec{x} \cdot \vec{x}}} \tag{3.5}$$

3.3.2 *Stemming* / Lematização

Para diminuir a dimensão dos documentos textuais a serem classificados, é comum o uso das técnicas de *stemming* ou lematização. Apesar de seus objetivos e resultados serem semelhantes, estas duas técnicas não são a mesma coisa:

- *Stemming* consiste na redução de palavras para sua raiz ou forma básica, através da remoção de sufixos. Esta técnica é normalmente usada para o idioma Inglês através do algoritmo de Porter [Por80], considerado como o padrão de-facto. Com este algoritmo, as palavras "connected", "connecting", "connection" e "connections" seriam reduzidas para a forma normal "connect". No *stemming* , não é necessário que a palavra reduzida exista efetivamente. Por exemplo, no algoritmo de Porter, a palavra "replacement" seria reduzida a "replac", que não existe na língua inglesa.

  Para o Português não existe nenhum algoritmo de *stemming* padrão e, por ser uma língua morfologicamente mais complexa que o Inglês, muitas vezes se opta pelo uso de algoritmos de lematização. Apesar disso, pelo menos um algoritmo é conhecido: o proposto por Orengo e Huyck [OH01].

- Lematização é a transformação de uma palavra em sua forma normal. Sua principal diferença em relação ao *stemming* é que na lematização é necessário que a forma normal resultante exista na idioma ao qual correspondia a palavra original. No caso da língua portuguesa, isso significa transformar verbos para o infinitivo e substantivos/adjetivos para sua forma masculina singular, i.e., o lema corresponde à forma da palavra que seria buscada em um dicionário.

  Também não existe nenhum algoritmo padrão para realizar lematização em Português, mas dois dos algoritmos propostos são [LMR94, BS00].

Não existe um consenso entre os estudos sobre as vantagens de se fazer ou não *stemming* ou lematização. Freqüentemente um mesmo estudo comparativo feito em bases textuais diversas, produz resultados opostos: algumas bases apresentam melhores resultados com a realização de *stemming*/lematização enquanto outras não.

Em relação ao idioma Português, praticamente inexistem estudos comparativos sobre o ganho ou perda de acurácia da classificação de documentos textuais com o uso ou não de *stemming*/lematização. Um dos poucos estudos disponíveis, conduzido por Gonçalves et al. [GQ04], apontou que, para a base analisada no idioma Português, os melhores resultados ocorreram sem o uso de lematização. Entretanto, este estudo não é conclusivo por ter sido feito apenas com uma única base textual em Português, base esta que consistia de documentos judiciais, ou seja, possuía uma homogeneidade não encontrada na maioria das bases de dados textuais.

3.3.3 Remoção de *Stopwords*

A remoção de *stopwords* consiste em ignorar, para fins de treinamento e classificação, palavras que aparecem em todos ou em um número muito alto de documentos. A teoria por trás desta remoção é que se uma palavra aparece em praticamente todos os documentos ela não serve como discriminadora de categorias e, portanto, pode ser removida sem perda de informação relevante para a classificação.

Ela pode ser vista como um caso especial de Ponderação de Termos, onde se atribui peso 0 para palavras muito freqüentes. Geralmente, entretanto, por fins de eficiência, sua remoção é feita consultando-se uma lista fixa, pré-definida, de palavras comuns como artigos, preposições e conjunções.

3.3.4 Remoção de palavras raras

A remoção de palavras raras, juntamente com a remoção de *stopwords* , objetivam reduzir as dimensões dos vetores de representação dos documentos e, por conseqüência, reduzir o ruído e aumentar a performance do classificador.

No caso da remoção de palavras raras, são removidos os termos que não apresentam uma freqüência mínima, D_f de aparição nos documentos. É possível também se determinar um número absoluto mínimo abaixo do qual os termos não são considerados. Em geral, neste caso, usa-se números pequenos como 1 a 3, por exemplo.

Apesar de não haver um consenso entre os estudos em relação aos ganhos com esta remoção, como existe no caso da remoção de *stopwords* , a grande maioria dos autores considera que esta prática produz melhores resultados, já que termos com uma freqüência muito baixa introduzem ruído e não contribuem para discriminar documentos entre diferentes classes. Um dos poucos estudos que visam contradizer este fato foi o apresentado por Schönhofen et Benczúr [SB06], porém seu trabalho se focou no uso de n-gramas, o que não necessariamente se aplica ao caso de *bag-of-words* .

Um dos primeiros a fazerem um estudo empírico sobre as freqüências que são úteis como discriminantes entre categorias foram Salton et al. [SWY75], que concluíram que os termos que possuem $\frac{N_t}{100} \leq D_f \leq \frac{N_t}{10}$, onde N_t é o número total de documentos, são os melhores para discriminar categorias.

3.4 Seleção de Atributos

Um dos problemas enfrentados por boa parte dos algoritmos usados na classificação de textos está no tamanho dos vetores de representação dos documentos: eles freqüentemente podem ter dimensões medidas em dezenas ou centenas de milhares. Poucos algoritmos conseguem lidar com vetores desta dimensionalidade diretamente (SVM é um destes algoritmos).

A idéia por trás da seleção de atributos está na redução da dimensionalidade dos vetores através de técnicas de eliminação ou combinação de termos entre si.

Duas das técnicas mais utilizadas "Remoção de *Stopwords* " e "Remoção de palavras raras" já foram mostradas como técnicas de pré-processamento, nas seções 3.3.3 e 3.3.4, respectivamente.

Outras possíveis opções, também bastante utilizadas, consistem em remover termos de acordo com técnicas conhecidas de IR: *Information Gain*, *Mutual Information* e Estatística χ^2.

Yang e Pedersen [YP97] fizeram um estudo empírico comparando como estas três técnicas de seleção de atributos estariam relacionadas e quais produziriam melhores resultados. Suas conclusões foram que a freqüência de documentos D_f (definição 3.4) está intimamente relacionada com *Information Gain* e Estatística χ^2, podendo ser utilizada

de forma confiável para substituí-las. O uso de *Mutual Information* produziu resultados bastante inferiores às demais técnicas.

Forman [For03] fez um trabalho empírico bastante completo sobre a seleção de atributos, utilizando doze métricas diferentes para a classificação de textos com *Support Vector Machines*. Sua conclusão foi que com apenas uma métrica, *Separação Bi-Normal*, se obteve performance superior ao do uso de todas os atributos. O problema é que neste estudo, segundo admitiu o próprio autor, ele não variou o parâmetro C de capacidade da SVM (ver Seção 4.2). Desta forma, seria teoricamente possível obter uma acurácia ainda maior usando todas os atributos se este parâmetro fosse otimizado e este problema ainda permanece em aberto.

Outra possibilidade bastante distinta para a redução do número de atributos foi proposta por Brank et al. [BGMFM02]. Sua idéia é treinar uma SVM linear utilizando todas os atributos originais e depois transformar em 0 todos os α_i do vetor de coeficientes $\vec{\alpha}$ (que gera o vetor normal $\vec{w}$ - ver equação 4.11) todos aqueles com um valor menor que um limite pré-determinado. Desta forma, o vetor resultante seria bastante mais esparso que o vetor original. Uma nova SVM (desta vez não linear) seria então treinada com os vetores esparsos obtidos. Vetores esparsos necessitam menor espaço de memória para serem armazenados e aumentam a velocidade de classificação de SVM (lineares ou não), já que o produto escalar de quaisquer dois vetores esparso é calculado mais rapidamente que dois vetores com dimensões iguais porém menos esparsos.

Embora vários outros trabalhos de seleção de atributos existam, eles não serão abordados neste trabalho em virtude de não serem realmente necessários para a classificação de textos com o uso de SVM, conforme demonstrado teoricamente por Joachims [Joa02] e empiricamente por Forman [For03].

3.5 Ponderação de Termos

Ponderação de Termos, também conhecido como Ponderação de Atributos, consiste no aumento ou diminuição do peso de determinadas palavras no vetor de representação de documentos, com o objetivo de se obter uma melhor acurácia do classificador. As técnicas usadas na classificação de documentos têm sua origem em IR, a partir de adaptações das técnicas usadas na procura textual.

As técnicas de "Remoção de *Stopwords* " e "Remoção de palavras raras" (seções 3.3.3 e 3.3.4) também podem ser vistas pela perspectiva de Ponderação de Termos: elas seriam um caso especial desta técnica, que lhes atribuiria peso 0. As diferentes formas de representação de documentos textuais, em especial a TF (Seção 3.1.2) e TFIDF (Seção 3.1.3) também são consideradas como técnicas de Ponderação de Termos.

Além destas técnicas tradicionalmente utilizadas, outras abordagens são possíveis. Debole e Sebastiani [DS03] mostraram que técnicas normalmente utilizadas na seleção de atributos como a estatística χ^2, *Information Gain* e *Gain Ratio*, também são úteis quando usadas para aumentar ou diminuir o peso de termos. Eles chamaram esta técnica proposta por eles de *Supervised Term Weighting (STW)* ou, em Português, Ponderação de Termos Supervisionada.

Taylor e Cristianini [STC04] sugeriram o uso de uma matriz de proximidade para capturar alguma informação semântica do documento e modificar o peso de termos que sejam sinônimos ou estejam localizados semanticamente próximos. Para a construção desta ma-

triz, eles sugeriram o uso de informação externa ao algoritmo, como a fornecida por redes semânticas, citando especificamente a Wordnet [Fel98] como uma destas possíveis redes. Infelizmente tais redes não estão disponíveis para a maioria das línguas, inviabilizando seu uso de forma mais abrangente.

Outra forma de Ponderação de Termos que pode ser usada de maneira bem intuitiva aparece na classificação de documentos textuais semi-estruturados, como é o caso de páginas Web. Com esta abordagem, termos originários de partes específicas do documento, e.g. seu título ou ementa, podem automaticamente receber um peso maior, já que se imagina que eles possam caracterizar melhor seu conteúdo do que termos obtidos de partes mais genéricas do documento.

3.6 Classificação de páginas Web

A classificação de páginas Web é um problema que surgiu diante do explosivo crescimento que a Internet apresentou desde sua transformação em rede comercial. Em nenhum momento anterior na história se teve acesso a tão grande quantidade de dados, com tanta facilidade e baixo custo como atualmente. Toda esta facilidade vem com um preço: é necessário alguma forma de classificar as páginas da Internet, de modo a facilitar o acesso à informação, bem como controlar seu acesso, para evitar que informações inadequadas sejam obtidas por qualquer pessoa.

É importante destacar que controlar o acesso não significa censurar informação, mas simplesmente evitar que ela seja visualizada por todos em qualquer lugar. É extremamente importante, por exemplo, para pais e mães de crianças e adolescentes conseguir impedir que seus filhos tenham acesso à sites de conteúdo pornográfico, que incentivem a violência ou, promovam o racismo, dentre outros conteúdos considerados danosos. Empresas também freqüentemente estão preocupadas com a perda de produtividade de seus funcionários causada pelo uso irresponsável da Internet. Estimativas recentes apontam perdas anuais de 6,5 bilhões de libras por ano, apenas no Reino Unido, pelo uso de sites de relacionamento por funcionários durante seu horário de trabalho [Cro08]. As perdas totais anuais estimadas pelo mau uso da Internet, novamente no Reino Unido, chegam a 10,6 bilhões de libras por ano [cbi08].

3.6.1 Particularidades

A classificação de páginas Web se difere da classificação de textos tradicional em diversos aspectos e pode ser vista como um problema mais complexo:

- Páginas Web são documentos semi-estruturados e escritos através da linguagem HTML [BLC95]. Apesar de outros textos a serem classificados também poderem apresentar algum tipo de estrutura, tipicamente ela é removida antes da classificação o que não ocorre normalmente com páginas Web.
- Documentos Web apresentam uma estrutura de *hyperlinks*, que permite que páginas sejam conectadas a outras páginas, formando uma estrutura em forma de grafo. Apesar desta característica também ocorrer em outros tipos de documentos textuais, ela é essencial no caso da Web e é o que a caracteriza.

- Freqüentemente páginas Web são divididas em *frames* e, nestes casos, o *frame* principal normalmente contém apenas instruções para a carga dos demais. Isso faz com que muitas vezes seja necessário analisar mais de um documento durante o processo de classificação de uma página.
- Ao contrário de documentos textuais cujo conteúdo normalmente possui informação suficiente para sua correta classificação, páginas Web em muitos casos não fazem menção explícita ao seu conteúdo e nem fornecem informações textuais suficientes para que sejam corretamente classificadas. De acordo com o apresentado por [GTL+02], o site da Microsoft (`http://www.microsoft.com`), por exemplo, não faz nenhuma menção ao fato de que eles desenvolvem sistemas operacionais.
- A forma com que a Web está estruturada encoraja a criação de documentos fragmentados, de autores diversos e cujos tópicos somente podem ser determinados em contextos mais amplos. Isso explica o fato do desempenho de classificadores quando utilizados para páginas Web seja significantemente menor do que quando aplicados para documentos textuais puros, conforme observado por Chakrabarti et al. [CDI98], que comentam sobre um teste que fizeram com um mesmo classificador que apresentava acurácia de 87% para a base da Reuters e que apresentou apenas 32% quando classificando exemplos do Yahoo!
- Não muito raramente, uma mesma página Web apresenta textos (parágrafos, mensagens, citações) escritos em línguas diferentes. Isso faz com que algoritmos de *stemming* ou lematização (Seção 3.3.2) não possam ser aplicados diretamente ou, se forem aplicados sem uma análise apurada para identificar o idioma de cada pedaço do texto, produzam resultados ruins.

3.6.2 Estrutura de uma página Web

A estrutura de uma página Web consiste geralmente de um cabeçalho e um corpo. O cabeçalho contém o título da página e elementos opcionais. O corpo contém o texto da página, dividido em parágrafos, listas e outros elementos [BLC95]:

- O cabeçalho de um documento HTML é uma coleção de informação não ordenada sobre este documento. De particular interesse para a classificação destes documentos estão o título da página e os meta-dados.
- O título de uma página é obrigatório [BLC95], porém nem sempre esta especificação é seguida a risca. Tipicamente, quando disponível, ele é mostrado na parte superior de um navegador quando uma página está sendo mostrada. Ele é de particular importância na classificação de sites já que, como observado por [XHL06], imagina-se que uma palavra contida no título de um documento seja mais representativa de seu conteúdo de que uma palavra em seu corpo. Entretanto, cerca de 89% dos títulos de páginas Web possuem apenas entre uma e dez palavras e muitas vezes apresentam apenas um nome ou termos como “home page” [Pie00].
- Os meta-dados, contidos no cabeçalho de páginas Web, são contêiners extensíveis usados na indicação de determinadas características destas páginas. De grande

interesse para sua classificação estão as "keywords", que apresentam as palavras-chave da página (comumente utilizadas para ajudar a sites de busca a trazer a página como resultado de uma query) e "description", que apresenta uma descrição geral do conteúdo desta página. A informação contida nestes campos, entretanto, é inconsistente, não-homogênea e muitas vezes inexistente [Pie00], impedindo que classificações sejam feitas apenas com base nestes dados.

- O corpo da página contém o texto do documento, incluindo cabeçalhos e parágrafos, bem como outros elementos não textuais como imagens, *applets*, entre outros. Todo o texto está delimitado por tags HTML que devem ser removidas antes que o documento possa ser classificado.
- Um item importante para a classificação contido no corpo das páginas Web são os cabeçalhos ou títulos que consistem no texto que aparece entre tags do tipo <Hn> e </Hn> [XHL06].

3.6.3 Abordagens de classificação de páginas Web

Diversas abordagens, das mais distintas, já foram analisadas para tentar resolver, em todo ou em parte, o problema da classificação de páginas Web.

Asirvatham e Ravi [KR01] propuseram uma abordagem onde classificavam as páginas em três categorias genéricas, sem levar em conta o seu conteúdo e baseando-se exclusivamente em sua estrutura. Apesar do grau de acerto ter sido baixo comparado com métodos que usam informações textuais, ele serviu para mostrar que a classificação de páginas Web deve ser encarada de forma diferente de uma simples classificação de textos: existe informação adicional sob a forma de links, imagens ou metadados que pode ser utilizada para melhorar a performance do classificador.

Glover et al. [GTL+02] analisaram a possibilidade de se classificar páginas a partir do texto citado em outras páginas que as referenciam, ao invés de seu próprio conteúdo. Eles concluiram que o melhor resultado de classificação é atingido quando utilizada uma combinação do texto da citação de uma página com seu próprio conteúdo. Apesar de apresentar bons resultados, este método é de pouca utilidade prática, já que para que ele funcione é necessário se obter todas ou, pelo menos, uma boa parte das páginas que referenciam uma determinada página a ser classificada e isso não é viável, exceto para o caso de sites de busca.

Qi e Davison [QD06] estudaram a melhora dos classificadores textuais através da utilização da informação da classe dos sites que estivessem próximos à página sendo classificada. Como próximos, estariam os sites pai (que referenciam a página), irmãos (que também são referenciados por uma página pai), filhos (as páginas referenciadas pela que está sendo classificada) e esposa (uma página qualquer que referencia uma das páginas filho). Embora os resultados, segundo os autores, tenha contribuído para melhorar a taxa de acerto do classificador, novamente esta abordagem não é viável na prática, por necessitar de todo um grafo de referencias de páginas.

Chen et al. [CWZH06] propuseram uma abordagem híbrida, baseada na combinação de classificadores textuais e de imagens, que buscavam identificar imagens pornográficas e assim melhorar o grau de acerto do classificador. Entretanto, o resultado de seu método foi inferior ao obtido por diversos classificadores que utilizam informações apenas textuais.

Sua abordagem apresenta também o grande inconveniente de a grande maioria dos sites, com a excessão de sites pornográficos, não possuir imagens que consigam caracterizar seu conteúdo. Além disso, mesmo imagens pornográficas são muito difíceis de serem identificadas com precisão com a atual tecnologia, apresentando muitos falsos positivos e negativos.

Kan [Kan04] e posteriormente M. Indra Devi and K. Selvakuberan [DS07] propuseram uma forma de classificar páginas Web utilizando apenas os dados presentes na URL. Para isso, seu algoritmo realiza a decomposição da URL em diversos tokens, de tamanhos variados, tentando identificar possíveis palavras úteis para a classificação da página. Os testes realizados, entretanto, foram feitos apenas para um conjunto bastante limitado de dados de páginas acadêmicas.

Sun et al. [SLN02] estudaram formas de extração de features de páginas Web para classificação usando SVM. Foram analisados o uso apenas do texto (o que está contido no corpo de uma página Web), do texto com o título, do texto com o conteúdo das âncoras de páginas que estivessem citando a página sendo classificada e uma combinação dos três. Sua conclusão foi que, para os dados analisados, a combinação do texto, com conteúdo das âncoras ofereceu o melhor resultado, porém o uso do título junto com o texto ofereceu resultado melhor do que apenas do texto puro.

Xue et al. [XHL06] fizeram um estudo um pouco mais completo que [SLN02] e analisaram várias maneiras de representar uma página Web para fins de classificação, usando SVM, com kernels polinomial e RBF. Eles estudaram o uso do texto, do título, dos cabeçalhos das seções das páginas, dos metadados e da combinação destas features para verificar como se poderia obter os melhores resultados. A conclusão dos autores foi que, com ambos os kernels, os melhores resultados são obtidos através do uso conjunto de todas as features, sempre que os dados fossem separados (isto é, uma feature obtida de um meta dado, por exemplo, é incluída em uma dimensão diferente, mesmo se houver uma feature igual no corpo da página). Como resultado adicional, foi constatado que no caso de páginas Web, ao contrário da classificação de textos simples, melhores resultados são produzidos com uma representação que use a freqüência das features diretamente, sem usar TFIDF.

3.6.4 Tipos de classificação

Existem basicamente 4 abordagens gerais que podem ser utilizadas para a classificação de páginas Web, de acordo com Sun et at. [SLN02]:

- **Utilização apenas do texto**

  Este é o método mais simples de todos e consiste na utilização apenas do conteúdo textual das páginas, sem levar em conta nenhum outro atributo. Ele em geral serve como uma base sobre a qual os resultados de outros tipos de classificadores podem ser analisados.

- **Hipertexto**

  Nesta abordagem, são incluídos não apenas features extraídas do texto da página, mas também informações obtidas da estrutura de uma página Web, como tags HTML e seu título.

- **Páginas Vizinhas**

  Neste método, dados extraídos das páginas próximas (pai, irmãos, filhos, e esposas, como descrito em [QD06]), são utilizados na classificação de uma página. Normalmente os dados consistem de texto extraído das âncoras de citação, mas podem também incluir os parágrafos em volta de um link ou até mesmo todo o conteúdo da página.

- **Análise de Links**

  Esta abordagem consiste de analisar apenas o conteúdo do link de uma página, ou utilizar a estrutura de links para classificar uma página em virtude da proximidade dela com outras páginas já previamente classificadas.

Além destas diferentes abordagens, páginas Web podem ser classificadas de forma hierárquicas (ou seja, as categorias são subdivididas em uma árvore de categorias) ou plana, na qual não existe nenhuma categoria superior a outra.

Um dos primeiros trabalhos de classificação hierárquica de páginas Web foi desenvolvido por Dumais e Chen [DC00]. Eles utilizaram SVM para classificar páginas de forma hierárquica e plana e concluíram que existe um pequeno ganho na classificação de páginas quando feita de forma hierárquica em relação à classificação plana.

Um estudo bem mais recente conduzido por Wetzker et al. [WAB+07], entretanto, mostrou resultados opostos, mostrando que, tanto na base Reuters quanto para páginas Web, o classificador Hierárquico apresentou acurácia inferior ao plano.

De forma geral, a classificação hierárquica funciona bem para grandes coleções de documentos organizados de formas complexas, quando consegue produzir duas vantagens principais: ganho de eficiência e diminuição dos erros de classificação. Entretanto, ela apresenta o problema de não possibilitar que um mesmo documento receba mais de uma categoria [CM07].

4. Support Vector Machines

Support Vector Machine é uma técnica de aprendizagem de máquina utilizada para classificação e reconhecimento de padrões e regressão de dados. Sua formulação inicial foi descrita pela primeira vez por Vapnik et al. em [BGV92], sendo entretanto baseada em um trabalho anterior do próprio Vapnik [Vap82]. Vários estudos comparativos entre algoritmos de classificação realizados nos últimos anos apontaram freqüentemente as SVM como o algoritmo que produzia os melhores resultados para a classificação de padrões e, em especial, para a classificação de textos [CV95, DPHS98, YL99, Joa02, CCO03, ZL03].

Este capítulo resume o estado da arte das SVM, de forma construtiva: inicialmente é mostrada na Seção 4.1 sua concepção original, formulada por Vapnik, e posteriormente conhecida como *Hard-Margin SVM*. A seguir, a evolução do algoritmo que se tornou conhecido como *Soft Magin SVM* é descrita na Seção 4.2, seguida por sua formulação não-linear, na Seção 4.3. A Seção 4.4 descreve variações posteriores ao algoritmo de SVM, formuladas por diferentes autores, e a Seção 4.5 descreve a extensão que permite que o algoritmo trate diretamente problemas de classificação multiclasse. Por fim, a Seção 4.6 apresenta os algoritmos conhecidos atualmente para realizar o treinamento das SVM.

4.1 *Hard-Margin* SVM

Na sua concepção mais simples, originalmente descrita em [BGV92], uma *Hard-Margin SVM* consiste em um algoritmo que objetiva encontrar um hiperplano que consiga separar com a maior margem possível e de forma linear vetores em um espaço vetorial n-dimensional. A partir da definição deste hiperplano, que é gerado a partir dos dados do treinamento, obtêm-se uma função de decisão que pode ser usada para a classificação de dados pertencentes a duas categorias diferentes: as que se localizarem em um lado do hiperplano pertencem a uma classe e as que se localizarem no lado oposto pertencem a outra.

A limitação deste tipo de SVM é que ele somente funciona para os casos onde os dados sejam totalmente separáveis linearmente, i.e., não admite erros de classificação, ruídos, entre outros.

4.1.1 Formalização

Seja um conjunto de p exemplos a serem utilizados para treinar o algoritmo, mapeados sob a forma de vetores $\vec{x_i}$, $1 \leq i \leq p$, com rótulos y_i:

$$(\vec{x_1}, y_1), (\vec{x_2}, y_2), \cdots, (\vec{x_p}, y_p) \tag{4.1}$$

Onde os rótulos assumiriam os seguintes valores:

$$\begin{cases} y_k = 1, & \text{se } x_k \in \text{classe A} \\ y_k = -1, & \text{se } x_k \in \text{classe B} \end{cases}$$

Estes dados são linearmente separáveis se existir um vetor $\vec{w}$ e um escalar b tal que as inequações

$$\begin{cases} \vec{w} \cdot \vec{x_i} + b \geq 1, & \text{se } y_i = 1 \\ \vec{w} \cdot \vec{x_i} + b \leq -1, & \text{se } y_i = -1 \end{cases} \tag{4.2}$$

são válidas para todos os elementos do conjunto de treinamento (4.1). Estas duas inequações (4.2) podem ser combinadas em uma única da seguinte forma:

$$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1, \quad i = 1, \cdots, p \tag{4.3}$$

Existe um único hiperplano que separa os dados do conjunto de treinamento (4.1) com a margem máxima. Este hiperplano determina a direção do vetor $\vec{w}$ onde a distância das projeções dos vetores dos dados do conjunto de treinamento das duas classes diferentes é máxima. Esta distância é expressada através da seguinte fórmula:

$$\delta = \underset{\{x:y=1\}}{min} \frac{\vec{x} \cdot \vec{w}}{\|\vec{w}\|} - \underset{\{x:y=-1\}}{max} \frac{\vec{x} \cdot \vec{w}}{\|\vec{w}\|} \tag{4.4}$$

O hiperplano ótimo é o que maximiza a distância (4.4). Juntando (4.3) e (4.4), temos que:

$$\delta = \frac{2}{\|\vec{w}\|} = \frac{2}{\sqrt{\vec{w} \cdot \vec{w}}} \tag{4.5}$$

Isso significa que o hiperplano que maximiza a distância δ é único e que podemos maximizar a distância de separação entre as classes minimizando $\|\vec{w}\|^2$, sujeito às restrições (4.3).

Os vetores $\vec{x_i}$ para os quais $\vec{w} \cdot \vec{x_i} + b = 1$ se localizam no hiperplano H_1. Os que atendem $\vec{w} \cdot \vec{x_i} + b = -1$ se localizam no hiperplano H_2. Eles são chamados de vetores de suporte e são os únicos cuja remoção do conjunto de treinamento alteraria a solução gerada. Os hiperplanos H_1 e H_2 são paralelos e nenhum ponto do conjunto de dados de treinamento se localiza entre eles. A Figura 4.1 ilustra esta definição. Nela, os vetores de suporte estão sobre as linhas pontilhadas e definem os hiperplanos H_1 e H_2.

O problema de minimizar $\|\vec{w}\|^2$, sujeito às restrições (4.3) pode ser reescrito utilizando os multiplicadores de Lagrange. Existem duas razões básicas de se fazer isso [Bur98]:

1. As restrições (4.3) são substituídas pelas restrições dos próprios multiplicadores de Lagrange, que são bastante mais simples de se trabalhar.

2. Com a reformulação do problema, os dados do treinamento vão aparecer apenas sob a forma de produtos internos entre vetores. Isso é de fundamental importância na medida em que possibilita que se generalize o procedimento para casos não lineares, como descrito na Seção 4.3.

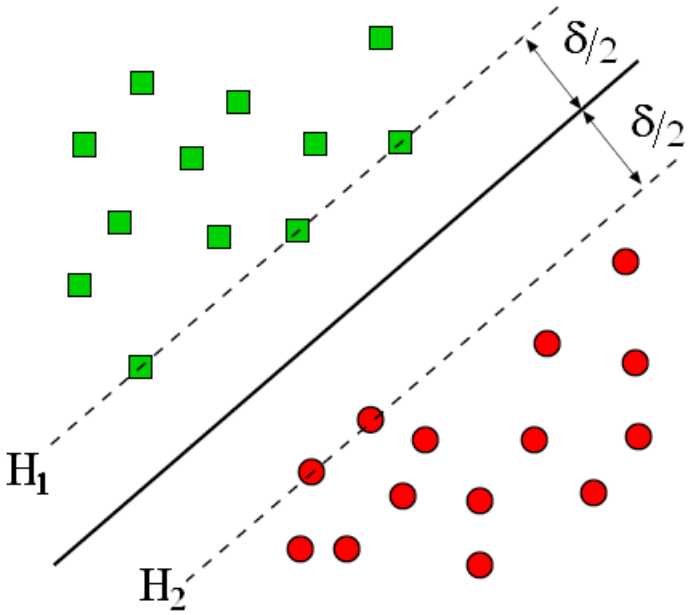


Figura 4.1: Separação linear pelo hiperplano ótimo

A reformulação do problema consiste na introdução de multiplicadores de Lagrange positivos $\alpha_i,\ i = 1, \cdots, p$, um para cada uma das restrições das inequações (4.3). Isso produz o seguinte problema primal de otimização:

$$\text{Minimizar: } L_P = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{p} \alpha_i y_i (\vec{x_i} \cdot \vec{w} + b) + \sum_{i=1}^{p} \alpha_i \tag{4.6}$$

De acordo com a restrição que:

$$\alpha_i \geq 0 \tag{4.7}$$

Este é um problema de programação quadrática convexo. Isso significa que o problema dual equivalente pode ser resolvido e a solução obtida será a mesma. Este caso particular de formulação do problema dual é chamado de Wolfe Dual [Fle87]. A reformulação do problema primal em dual, produz o seguinte problema de otimização [Bur98, Joa02]:

$$\text{Maximizar: } L_D = \sum_{i=1}^{p} \alpha_i - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j (\vec{x_i} \cdot \vec{x_j}) \tag{4.8}$$

De acordo com as restrições que:

$$\sum_{i=1}^{p} \alpha_i y_1 = 0 \tag{4.9}$$

$$\alpha_i \geq 0 \tag{4.10}$$

A solução deste problema é um vetor de coeficientes $\vec{\alpha}^T = (\alpha_1, \cdots, \alpha_p)^T$ para os quais (4.8) apresenta valor máximo. Estes coeficientes podem ser utilizados para construir o hiperplano ótimo de separação entre as classes (4.3), através da seguinte equação [CV95]:

$$\vec{w} = \sum_{i=1}^{p} \alpha_i y_i x_i \tag{4.11}$$

Em (4.11) apenas os vetores de suporte possuem coeficientes $\alpha_i \neq 0$. Para calcular o valor do escalar b de (4.3), basta selecionar arbitrariamente qualquer um destes vetores de suporte, $\vec{x_{vs}}$ e substituí-lo em (4.12).

$$b = y_{vs} - \vec{w} \cdot \vec{x_{vs}} \tag{4.12}$$

Uma vez calculados $\vec{w}$ e b, podemos utilizar a máquina obtida para a classificação de um dado exemplo $\vec{x}$, através da seguinte função de decisão:

$$f(\vec{x}) = \vec{w} \cdot \vec{x} + b \tag{4.13}$$

Se $f(\vec{x}) > 0, \vec{x} \in$ classe A, caso contrário $\vec{x} \in$ classe B.

4.2 *Soft-Margin* SVM

O algoritmo de *Hard-Margin SVM* , mostrado acima, funciona bem quando os dados são linearmente separáveis, entretanto, caso isso não ocorra, o problema não pode ser resolvido com o uso do algoritmo.

Apesar da maioria dos casos de classificação de textos ser separável linearmente, isso não ocorre com muitos outros tipos de padrões e, mesmo no caso textual, a presença de ruídos pode inviabilizar a separação linear ou produzir um hiperplano separador com margem muito pequena.

Para resolver esta limitação do algoritmo original, Cortes e Vapnik [CV95] propuseram uma alteração no algoritmo de *Hard-Margin* para que ele pudesse tratar também o caso não linearmente separável. Esta modificação ficou conhecida como *Soft-Margin SVM*.

A idéia básica é relaxar as restrições (4.2), mas apenas quando necessário, i.e., introduzir um custo adicional quando isso for feito [Bur98]. A forma de fazê-lo é introduzir variáveis positivas de "afrouxamento" (*slack variables*) $\xi_i, i = 1, \cdots, p$ nas restrições (4.2).

4.2.1 Formalização

Seja um conjunto de p exemplos a serem utilizados para treinar o algoritmo, mapeados sob a forma de vetores $\vec{x_i}$, $1 \leq i \leq p$, com rótulos y_i:

$$(\vec{x_1}, y_1), (\vec{x_2}, y_2), \cdots, (\vec{x_p}, y_p) \tag{4.14}$$

Onde os rótulos assumiriam os seguintes valores:

$$\begin{cases} y_k = 1, & \text{se } x_k \in \text{classe A} \\ y_k = -1, & \text{se } x_k \in \text{classe B} \end{cases}$$

Estes dados são linearmente separáveis, com erros, se existir um vetor $\vec{w}$ e um escalar b tal que as inequações

$$\begin{cases} \vec{w} \cdot \vec{x_i} + b \geq 1 - \xi_i, & \text{se } y_i = 1 \\ \vec{w} \cdot \vec{x_i} + b \leq -1 + \xi_i, & \text{se } y_i = -1 \\ \xi_i \geq 0, i = 1, \cdots, p \end{cases} \tag{4.15}$$

são válidas para todos os elementos do conjunto de treinamento (4.14). Novamente, estas duas inequações (4.15) podem ser combinadas em uma única da seguinte forma:

$$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 - \xi_i, \quad i = 1, \cdots, p \tag{4.16}$$

Para que um erro ocorra, é necessário que ξ_i supere o valor 1, desta forma $\sum_{i=i}^{p} \xi_i$ é um limite superior para o número de erros de treinamento. Sendo assim, uma maneira natural de atribuir um custo extra para os erros é mudar a função objetiva a ser minimizada de $\|\vec{w}\|^2$ para

$$\frac{1}{2}\|\vec{w}\|^2 + C(\sum_{i=i}^{p} \xi_i) \tag{4.17}$$

onde C é um parâmetro escolhido pelo usuário e que corresponde a um balanço entre a capacidade de aprendizagem da máquina e o número de erros de treinamento. Valores pequenos de C produzirão mais erros, enquanto valores maiores farão com que o algoritmo se comporte mais próximo às *Hard-margin SVM*. A Figura 4.2 mostra o caso de um conjunto de dados não linearmente separáveis, porém que podem ser separados por uma Soft-margin SVM. Os vetores de suporte estão sobre as linhas pontilhadas e definem os hiperplanos H_1 e H_2. O ponto com erro está marcado com uma cruz.

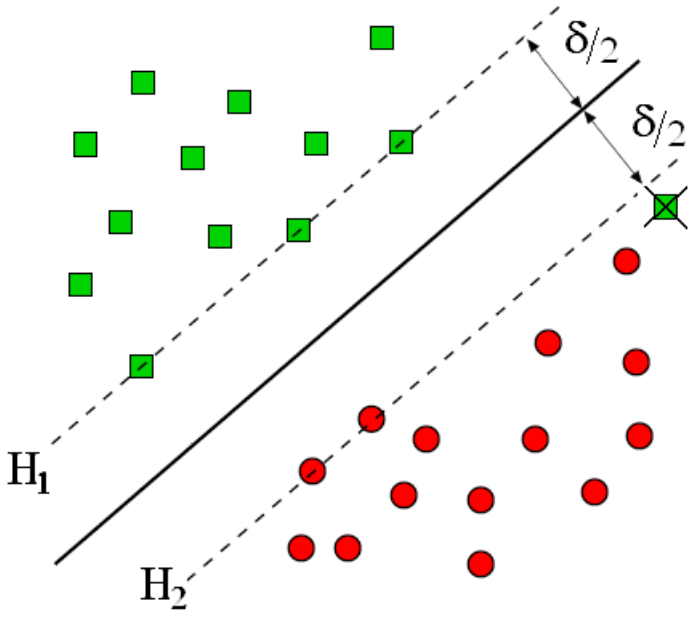


Figura 4.2: Separação linear pelo hiperplano ótimo, com erro

A reformulação de (4.17) sujeito às restrições (4.15), utilizando-se os multiplicadores de Lagrange, produz o seguinte problema primal [Bur98]:

$$\text{Minimizar: } L_P = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{p}\xi_i - \sum_{i=1}^{p}\alpha_i y_i((\vec{x_i}\cdot\vec{w}+b)-1+\xi_i) - \sum_{i=1}^{p}\mu_i\xi_i \qquad (4.18)$$

Onde μ_i são os multiplicadores de Lagrange introduzidos para garantir a positividade de ξ_i.

Novamente, é mais fácil resolver o problema Wolfe dual equivalente, que é exatamente igual ao da *Hard-Margin SVM*, exceto pela introdução de uma limitação superior nos valores dos coeficientes α_i:

$$\text{Maximizar: } L_D = \sum_{i=1}^{p}\alpha_i - \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j y_i y_j(\vec{x_i}\cdot\vec{x_j}) \qquad (4.19)$$

De acordo com as restrições que:

$$\sum_{i=1}^{p}\alpha_i y_1 = 0 \qquad (4.20)$$

$$0 \leq \alpha_i \leq C \qquad (4.21)$$

A solução deste problema é novamente um vetor de coeficientes $\vec{\alpha}^T = (\alpha_1, \cdots, \alpha_p)^T$ para os quais (4.19) apresenta valor máximo, com a única diferença que agora existe um limite superior C para cada α_i. Estes coeficientes podem ser utilizados para construir o hiperplano ótimo de separação entre as classes (4.16), através da seguinte equação:

$$\vec{w} = \sum_{i=1}^{p}\alpha_i y_i x_i \qquad (4.22)$$

Em (4.22) apenas os vetores de suporte possuem coeficientes $\alpha_i \neq 0$. Para diferenciar aqueles que possuem $0 < \alpha_i < C$ dos que possuem $\alpha_i = C$, chamam-se os primeiros de vetores de suporte não limitados, enquanto os últimos são chamados de vetores de suporte limitados (*bounded*) [Joa02].

Para calcular o valor do escalar b de (4.16), basta selecionar arbitrariamente qualquer um dos vetores de suporte não limitados, $\vec{x_{vsnl}}$ e substituí-lo em (4.23).

$$b = y_{vsnl} - \vec{w}\cdot\vec{x_{vsnl}} \qquad (4.23)$$

Exatamente da mesma forma que nas Hard-Margin SVM, uma vez calculados $\vec{w}$ e b, podemos utilizar a máquina obtida para a classificação de um dado exemplo $\vec{x}$, através da seguinte função de decisão:

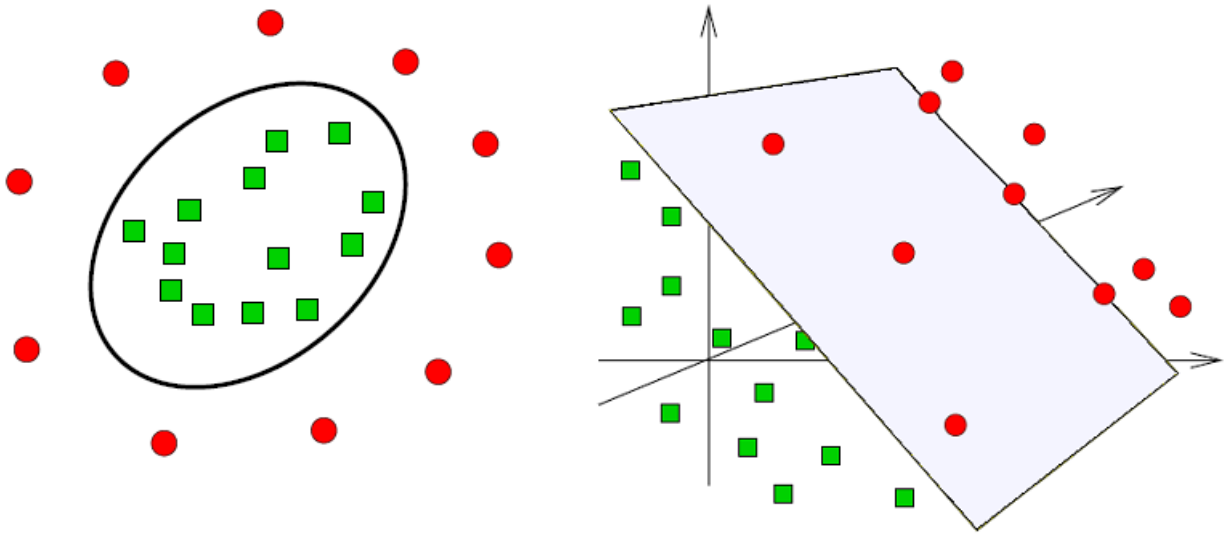
$$f(\vec{x}) = \vec{w}\cdot\vec{x} + b \qquad (4.24)$$

Se $f(\vec{x}) > 0, \vec{x} \in$ classe A, caso contrário $\vec{x} \in$ classe B.

4.3 SVM Não lineares

Até o momento, as SVM foram consideradas apenas para o caso onde os dados pudessem ser linearmente separáveis, ainda que com erros. Isso na prática, é uma grande limitação, já que muitos dados de problemas reais não podem ser linearmente separados.

Vapnik et al. [BGV92] mostraram que este problema pode ser resolvido e o algoritmo SVM generalizado para o caso não linear, de forma extremamente simples e elegante. Para entender como isso é feito, inicialmente deve-se atentar para o fato de que a única maneira como os dados do treinamento aparecem na equação (4.19) é através de produtos internos dos vetores $(\vec{x_i} \cdot \vec{x_j})$. Isso significa que se soubermos como calcular o produto interno de dois vetores em um espaço Euclidiano hiperdimensional qualquer (com dimensões possivelmente infinitas), poderemos calcular nossa função de decisão, ainda que não tenhamos idéia de como os vetores são representados neste espaço hiperdimensional. A figura 4.3 ilustra a transformação dos dados de uma dimensão baixa (a), onde eles não podem ser separados linearmente, em uma dimensão maior (b), onde a separação linear por um hiperplano é possível.



(a) Problema original, em uma dimensão baixa (b) Problema remapeado para uma dimensão maior

Figura 4.3: Transformação dos dados de uma dimensão baixa em uma dimensão maior

4.3.1 Formalização

Seja um mapeamento qualquer Φ, que mapeia os dados do treinamento de um espaço de entrada de d dimensões, para um espaço Euclidiano qualquer, chamado de espaço de *features*, definido da seguinte forma:

$$\Phi : \mathbb{R}^d \mapsto \mathcal{H} \tag{4.25}$$

A partir deste momento, todo o treinamento e execução do algoritmo como classificador vai depender dos produtos internos dos vetores no espaço $\mathcal{H}$, ou seja, através de funções da forma $\Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$. Desta forma, a equação (4.19) pode ser reescrita como:

$$\text{Maximizar: } L_D = \sum_{i=1}^{p} \alpha_i - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j (\Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})) \tag{4.26}$$

O problema passa agora a ser como calcular $\Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$, porém, conforme demonstrou Vapnik [BGV92], isso também pode ser feito de maneira bastante simples e eficiente: ao invés de se calcular explicitamente o valor de $\Phi(\vec{x_k})$, para cada vetor k, pode-se computar diretamente o valor $\Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$, através de uma função que usa os dados de entrada de forma direta. Em outras palavras, o mapeamento dos vetores de entrada pode ser contornado e o produto interno dos vetores mapeados por Φ ser calculado diretamente. Uma função que possui a capacidade de calcular diretamente o valor do produto interno de dois vetores mapeados por Φ em um espaço $\mathcal{H}$, usando os dados de entrada diretamente é chamada de *função kernel* .

Segundo [STC04], uma *função kernel* é uma função K tal que, para todo $\vec{x_i}, \vec{x_j} \in \mathbb{R}^d$, satisfaz:

$$K(\vec{x_i}, \vec{x_j}) = \Phi(\vec{x_i}) \cdot \Phi(\vec{x_j}) \tag{4.27}$$

Sendo assim, podemos reescrever a equação (4.26), para obtermos a formulação de uma SVM não linear:

$$\text{Maximizar: } L_D = \sum_{i=1}^{p} \alpha_i - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j K(\vec{x_i}, \vec{x_j}) \tag{4.28}$$

É interessante que com o uso das funções kernel, podemos calcular uma SVM cujos dados de entrada sejam mapeados para um espaço de grandes dimensões, até mesmo infinitas, sem termos que trabalhar com o mapeamento diretamente, o que seria em geral bastante complicado.

O problema agora é, uma vez resolvida (4.28) e obtidos os coeficientes α_i, como utilizar a máquina obtida para a classificação dos dados, já que precisamos do vetor $\vec{w}$ e ele também reside no espaço hiperdimensional $\mathcal{H}$. A solução é reescrever a função de decisão (4.24) para utilizar a decomposição de $\vec{w}$ nos vetores de suporte ao invés de utilizá-lo diretamente [Bur98]:

$$\begin{aligned} f(\vec{x}) &= \sum_{i=1}^{N_{vs}} \alpha_i y_i \Phi(\vec{x_i}) \cdot \Phi(\vec{x}) + b \\ &= \sum_{i=1}^{N_{vs}} \alpha_i y_i K(\vec{x_i}, \vec{x}) + b \end{aligned} \tag{4.29}$$

Onde x_i são os vetores de suporte. Desta forma, novamente evitamos computar explicitamente $\Phi(\vec{x})$ e utilizamos somente a função $K(\vec{x_i}, \vec{x})$. O problema desta solução é que ela faz com que uma SVM não linear seja em média N_{vs} vezes mais lenta que uma

linear, onde N_{vs} é o número de vetores de suporte. Burges [Bur96] propôs um algoritmo que substitui o conjunto dos vetores de suporte de (4.29) por um conjunto reduzido, não originários do treinamento, que produz uma solução aproximada para o problema em um tempo menor. Em seu artigo, o conjunto reduzido conseguiu fornecer uma função de decisão até 10 vezes mais rápida que a função original, essencialmente sem perda de precisão. Entretanto, nenhuma prova formal foi apresentada, apenas resultados empíricos, sendo este ainda um problema a ser melhor estudado.

A última questão a ser respondida é que tipo de funções podem ser utilizadas como funções kernel. Vapnik [CV95] demonstrou que qualquer função que satisfaça o teorema de Mercer pode ser usada como uma função kernel. Isso significa que para que exista um mapeamento Φ e uma expansão K, definidos como:

$$K(\vec{x_i}, \vec{x_j}) = \sum_{i=1}^{\infty} \Phi(\vec{x})_i \Phi(\vec{x})_j \tag{4.30}$$

É necessário e suficiente que, para cada função $g(\vec{x})$ tal que

$$\int g(\vec{x})^2 d\vec{x} < \infty \tag{4.31}$$

tenhamos

$$\int K(\vec{x}, \vec{y}) g(\vec{x}) g(\vec{y}) d\vec{x} d\vec{y} \geq 0 \tag{4.32}$$

Em geral é bastante complicado verificar se uma determinada função satisfaz ou não o teorema de Mercer, já que deve-se considerar toda função g que atenda (4.31). Devido a isso, normalmente usam-se como kernel funções que já se provou que o atendem. As mais freqüentemente utilizadas são as seguintes [Bur98, Joa02]:

$$\begin{aligned}
\text{Linear: } K(\vec{x}, \vec{y}) &= \vec{x} \cdot \vec{y} && (4.33)\\
\text{Polinomial: } K(\vec{x}, \vec{y}) &= (\vec{x} \cdot \vec{y} + 1)^p && (4.34)\\
\text{Radial Basis Function (RBF): } K(\vec{x}, \vec{y}) &= e^{-\gamma \|\vec{x} - \vec{y}\|^2}, \quad \gamma > 0 && (4.35)\\
\text{Sigmoid: } K(\vec{x}, \vec{y}) &= \tanh(k(\vec{x} \cdot \vec{y}) + \delta) && (4.36)
\end{aligned}$$

De particular interesse está o kernel RBF (4.35). Nele, o número de centros, os próprios centros, seus pesos (os coeficientes α_i) e o valor do escalar b são determinados automaticamente durante o treinamento da SVM, produzindo em geral excelentes resultados quando comparados a outros kernels [Bur98].

Um exemplo de uma separação de dados não linear, através do uso do kernel RBF (4.35) com $\gamma = 0,5$ e $C = 1000$ está ilustrada na Figura 4.4, que foi produzida com o uso da LIBSVM [CL01].

Outro ponto a ser destacado é que o kernel sigmoid (4.36) somente satisfaz as condições de Mercer para alguns valores dos parâmetros k e δ. Isso foi notado inicialmente de forma empírica por Vapnik [CV95].

Figura 4.4: Exemplo do uso do kernel RBF para a separação não linear de dados

4.4 Outros tipos de SVM binárias

Atualmente existem algumas variações das SVM em relação à forma com que foram concebidas inicialmente por Vapnik et al. Enquanto que a maioria das variações foi concebida para a regressão e, portanto, está fora do escopo deste documento, algumas possuem aplicação também para a classificação de dados e serão descritas nas seções a seguir.

4.4.1 SVM ponderada

Osuna et al. [OFG97c] propuseram uma extensão na formulação do problema das *Soft-Margin SVM* para tratar melhor os seguintes casos:

1. Uma proporção desigual de exemplos de dados entre classes (bases desbalanceadas).
2. A necessidade de se alterar o "peso" de uma classe em relação à outra, que ocorre freqüente no caso onde um erro de classificação de um tipo é mais custoso ou menos desejável que outro (um exemplo típico é o caso da filtragem de SPAM, onde um falso negativo é muito menos problemático que um falso positivo).

Desta forma, o problema (4.17) é reformulado da seguinte maneira:

$$\text{Minimizar: } \frac{1}{2}\|\vec{w}\|^2 + C^+(\sum_{i:y=1}\xi_i) + C^-(\sum_{i:y=-1}\xi_i) \tag{4.37}$$

Sujeito as condições que:

$$y_i(\vec{w}\cdot\vec{x_i}+b) \geq 1-\xi_i, \quad i = 1, \cdots, p \tag{4.38}$$

O problema dual equivalente à este é idêntico ao (4.19) após se substituir os limites superiores para os coeficientes α_i pelas constantes C^+ e C^-, para os exemplos positivos e negativos, respectivamente.

4.4.2 Nu-SVM

Esta modificação foi proposta por Schölkopf et al. [SSWB00] e ficou conhecida como $\nu-$SVM. Este tipo de SVM foi originalmente concebida para a regressão, mas pode também ser utilizada para a classificação de dados. Nela, ao invés do parâmetro C, usado para ajustar o balanço entre número de erros de treinamento e capacidade de aprendizagem, foi criado um novo parâmetro ν, que controla o número de vetores de suporte e erros. Segundo os próprios autores, quando usada para a classificação, se a escolha do novo parâmetro ν for feita corretamente, a performance da $\nu-$SVM é idêntica a da *Soft-Margin SVM*. Devido à isso, não abordaremos em detalhes sua formulação.

4.5 SVM Multiclasse

Originalmente, conforme concebidas por Vapnik et al., as SVM foram designadas para diferenciar apenas entre exemplos pertencentes a duas classes distintas. Desta forma, as extensões criadas para tratar o caso de multiclasses eram apenas métodos de decomposição (um-contra-todos ou um-contra-um) [GMS05].

Os primeiros a publicarem a descrição de uma SVM multiclasse que tratava o problema diretamente foram Weston e Watkins, em 1998 [WW98]. Este tipo de SVM passou a ser conhecida como M-SVM e, posteriormente, algumas variações foram apresentadas sobre a idéia original. Entretanto, apesar de sua criação remontar há vários anos, as SVM multiclasses ainda são um estudo em andamento, já que apenas recentemente elas passaram a despertar um interesse maior na comunidade científica, com o objetivo de entender melhor suas especificidades [Gue07].

De qualquer forma, apesar de conseguirem tratar diretamente o caso de rótulos multiclasse, ainda não existe nenhum modelo conhecido que consiga tratar o caso multi-label, i.e., onde cada vetor pode assumir um conjunto de rótulos e não apenas um único, que é o escopo deste trabalho. Devido a isso, será descrita aqui apenas a versão de Weston e Watkins, incluída com o intuito de ilustrar o funcionamento geral das M-SVM. Informações sobre a M-SVM de Crammer e Singer se encontram em [CS00, CS02, wHjL02, RK04, Gue07] e sobre a M-SVM de Lee et al. podem ser obtidas em [LLW04, RK04, Gue07].

4.5.1 M-SVM de Weston e Watkins

Seja um conjunto de l exemplos a serem utilizados para treinar o algoritmo, mapeados sob a forma de vetores $\vec{x_i}$, $1 \leq i \leq p$, com rótulos y_i:

$$(\vec{x_1}, y_1), (\vec{x_2}, y_2), \cdots, (\vec{x_p}, y_p) \tag{4.39}$$

Onde os rótulos $y_i \in \{1, \cdots, k\}$, as k classes distintas que o algoritmo consegue endereçar diretamente. O problema de otimização (4.17) pode então ser generalizado para o seguinte:

$$\text{Minimizar: } L_P = \frac{1}{2}\sum_{m=1}^{k}\|\vec{w_m}\|^2 + C(\sum_{i=i}^{p}\sum_{m=1}^{k}\xi_i^m) \tag{4.40}$$

De acordo com as restrições que:

$$(\vec{w_{yi}}\cdot\vec{x_i}) + b_{yi} \geq (\vec{w_m}\cdot\vec{x_i}) + b_m + 2 - \xi_i^m \tag{4.41}$$

$$\xi_i^m \geq 0, \quad i = 1, \cdots, p \quad m \in \{1, \cdots, k\} \setminus y_i \tag{4.42}$$

E a função de decisão se torna:

$$\arg\max_k[(\vec{w_i}\cdot\vec{x}) + b_i], \quad i = 1, \cdots, k \tag{4.43}$$

Novamente, da mesma forma que no caso das SVM binárias, é mais fácil resolver o problema dual equivalente [wHjL02]:

$$\text{Maximizar: } L_D = 2\sum_{i,m}\alpha_i^m + \sum_{i,j}(-\frac{1}{2}c_j^{yi}A_iA_j + \sum_m \alpha_i^m\alpha_j^{yi} - \frac{1}{2}\sum_m \alpha_i^m\alpha_j^m)K(\vec{x_i}, \vec{x_j}) \tag{4.44}$$

Onde:

$$A_i = \sum_{m=1}^{k}\alpha_i^m \tag{4.45}$$

$$c_i^n = \begin{cases} 1, & \text{se } y_i = n \\ 0, & \text{se } y_i \neq n \end{cases} \tag{4.46}$$

De acordo com as restrições que:

$$\sum_{i=1}^{p}\alpha_i^n = \sum_{i=1}^{p}c_i^nA_i, \quad n = 1, \cdots, k \tag{4.47}$$

$$0 \leq \alpha_i^m \leq C, \quad \alpha_i^{yi} = 0, \quad i = 1, \cdots, p, \quad m \in \{1, \cdots, k\} \setminus y_i \tag{4.48}$$

Com a solução deste problema, podemos construir os k hiperplanos separadores das classes, usando a seguinte equação:

$$\vec{w_n} = \sum_{i=1}^{p}(c_i^nA_i - \alpha_i^n)\Phi(\vec{x_i}), \quad n = 1, \cdots, k \tag{4.49}$$

E a função de decisão se torna

$$\arg\max_k[\sum_{i=1}^{p}(c_i^nA_i - \alpha_i^n)K(\vec{x_i}, \vec{x}) + b_n] \tag{4.50}$$

4.6 Treinamento de SVM

Para se treinar uma SVM, deve-se resolver um problema de otimização quadrática, conforme mostrado nas seções anteriores. Este problema possui grandes dimensões, já que a quantidade de exemplos a serem treinados é contada, em geral, na ordem de dezenas de milhares. Desta forma, é computacionalmente inviável (em termos de memória e tempo) resolver este problema diretamente, através de métodos conhecidos de programação quadrática.

Esta seção vai abordar os diferentes algoritmos de decomposição propostos para resolver o problema de treinamento das SVM. Antes de abordar os algoritmos, entretanto, é fundamental comentar sobre as condições de Karush-Kuhn-Tucker (KKT), que possuem um papel fundamental em todos os algoritmos de treinamento de SVM.

4.6.1 Condições de Karush-Kuhn-Tucker

As condições de Karush-Kuhn-Tucker (KKT) são condições necessárias para que a solução de um problema de programação não-linear seja ótima. Elas são satisfeitas para um problema de otimização qualquer, com qualquer tipo de restrição, desde que a interseção do conjunto de direções possíveis com o conjunto de direções de descida coincida com a interseção do conjunto de direções possíveis das restrições linearizadas com o conjunto de direções possíveis [Bur98]. No caso das SVM, esta suposição é sempre verdadeira, já que as restrições são sempre lineares. Além disso, o problema do treinamento das SVM é convexo e, para problemas convexos, as condições KKT são necessárias e suficientes para que $\vec{w}$, α e b sejam uma solução.

Desta forma, resolver o problema de treinamento de SVM é equivalente a encontrar uma solução para as condições KKT e isso é o princípio no qual se baseiam os algoritmos de decomposição, mostrados nas próximas seções.

Para o problema primal de uma soft-margim SVM (4.18), as condições KKT são as seguintes [Bur98]:

$$\frac{\partial L_P}{\partial w_\nu} = w_\nu - \sum_{i=1}^{p} \alpha_i y_i x_{i\nu} = 0 \tag{4.51}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{p} \alpha_i y_i = 0 \tag{4.52}$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \tag{4.53}$$

$$y_i(\vec{x_i} \cdot \vec{w} + b) - 1 + \xi_i \geq 0 \tag{4.54}$$

$$\xi_i,\ \alpha_i,\ \mu_i \geq 0 \tag{4.55}$$

$$\alpha_i(y_i(\vec{x_i} \cdot \vec{w} + b) - 1 + \xi_i) = 0 \tag{4.56}$$

$$\mu_i \xi_i = 0 \tag{4.57}$$

Onde ν varia de 1 à dimensão dos vetores de treinamento.

4.6.2 Algoritmo Chunking

Este foi o primeiro algoritmo de treinamento de SVM, proposto por Boser et al. [BGV92]. Ele inicia com um pequeno subconjunto arbitrário de dados e realiza o treinamento deste subconjunto. O resto dos dados é então testado no classificador resultante e uma lista de erros é construída, ordenada por quão longe eles estão no lado errado do hiperplano (i.e., o quanto as condições KKT são violadas). Feito isso, um novo subconjunto de dados é produzido, usando os N_{vs} vetores de suporte até então descobertos e os N_e maiores erros (o valor de $N_{vs} + N_e$ é determinado heuristicamente, para que o algoritmo progrida em uma velocidade adequada). Feito isso, o processo é repetido até que todos os pontos satisfaçam as condições KKT ou estejam dentro da margem de erro permitida.

Este algoritmo tem o problema de exigir que uma matriz $N_{vs} \times N_{vs}$ (com os vetores de suporte intermediários) caiba inteira em memória, o que pode não ser possível se o número de vetores de suporte for muito elevado.

4.6.3 Algoritmo de Decomposição

Este algoritmo foi proposto por Osuna et al. [OFG97a] (e mostrado de forma mais completa em [OFG97b]) com o objetivo de diminuir a necessidade do uso de memória para o treinamento de SVM com grande quantidade de pontos. Ele se baseia no fato de que, em geral, o número de vetores de suporte será bastante pequeno e portanto, boa parte dos coeficientes α_i dos vetores do treinamento serão 0.

O algoritmo funciona dividindo o conjunto de dados de treinamento em dois subconjuntos: um conjunto de trabalho B de tamanho fixo, grande o suficiente para conter todos os vetores de suporte mas pequeno o suficiente para caber em memória e outro conjunto N. Apenas o conjunto B possui pontos que atendem as condições KKT e apenas ele possui coeficientes α_i que podem variar (os pontos de N possuem coeficientes $\alpha_j = 0$). Seus passos são os seguintes:

1. Escolha arbitrariamente $|B|$ pontos do conjunto de dados
2. Treine o algoritmo com os pontos de B
3. Enquanto existir algum $j \in N$ tal que:

   - $\alpha_j = 0$ e $f(\vec{x_j})y_j < 1$
   - $\alpha_j = C$ e $f(\vec{x_j})y_j > 1$
   - $0 < \alpha_j < C$ e $f(\vec{x_j})y_j \neq 1$

   Onde $f(\vec{x})$ é a função definida em (4.29).

   Substitua $\alpha_i = 0,\ i \in B$ por α_j e resolva o novo problema obtido.

4.6.4 Algoritmo SMO

O algoritmo SMO (Sequential Minimal Optimization) foi desenvolvido por Platt em 1998 [Pla98] e posteriormente melhorado em 1999, através da inclusão da heurística "shrinking" [Pla99], originalmente desenvolvida por Joachims [Joa98].

Ele se propõe a resolver o problema de otimização (4.28) de forma analítica, através da otimização de apenas dois multiplicadores de Lagrange de cada vez. Esta abordagem foi provada como convergente por Osuna et al. [OFG97b], desde que em cada etapa seja utilizado pelo menos um multiplicador que viole as condições KKT para o problema. Estas condições são particularmente simples (e são as mesmas usadas no algoritmo de decomposição):

$$\alpha_i = 0 \quad \Leftrightarrow \quad f(\vec{x_i})y_i \geq 1 \tag{4.58}$$
$$\alpha_i = C \quad \Leftrightarrow \quad f(\vec{x_i})y_i \leq 1 \tag{4.59}$$
$$0 < \alpha_i < C \quad \Leftrightarrow \quad f(\vec{x_i})y_i = 1 \tag{4.60}$$

Onde $f(\vec{x})$ é a função definida em (4.29).

O funcionamento do SMO é baseado no algoritmo de decomposição de Osuna (Seção 4.6.3), porém ao invés de otimizar um número fixo $|B|$ de exemplos, ele os otimiza sempre dois a dois. Além disso, o algoritmo possui duas heurísticas de forma a escolher os multiplicadores a serem otimizados de modo a tentar dar o maior passo possível na direção da otimização, diminuindo o tempo para a resolução do problema completo. Outra grande vantagem do SMO é que, por resolver de forma analítica cada subproblema, ele não necessita de nenhuma biblioteca de programação quadrática, podendo ser facilmente implementado em qualquer linguagem de programação moderna.

O SMO funciona primeiro computando as restrições dos dois multiplicadores escolhidos e selecionando o valor mínimo para estas restrições. Como são apenas dois multiplicadores, elas podem ser mostradas em duas dimensões, como ilustrado na Figura 4.5, retirada de [Pla99]. Os multiplicadores de Lagrange α_1 e α_2 devem atender as restrições do problema completo. A restrição imposta pela inequação (4.21) faz com que eles fiquem dentro do quadrado, enquanto que a equação linear (4.20) faz com que eles se situem em uma linha diagonal.

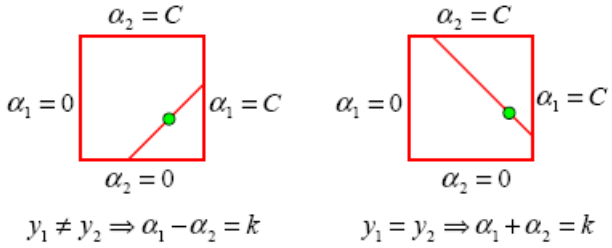


Figura 4.5: Restrições dos multiplicadores de Lagrange em duas dimensões

Em condições normais a função objetiva é positiva definida e existe um mínimo na direção da equação linear da restrição, definido por:

$$\alpha_2^{novo} = \alpha_2 + \frac{y_2(E_1 - E_2)}{K(\vec{x_1}, \vec{x_1}) + K(\vec{x_2}, \vec{x_2}) - 2K(\vec{x_1}, \vec{x_2})} \tag{4.61}$$

Onde $E_i = f(\vec{x}) - y_i$.

Uma vez definido o valor de α_2, o valor de α_1, é computado da seguinte forma:

$$\alpha_1^{novo} = \alpha_1 + y_1 y_2(\alpha_2 - \alpha_2^{novo,escolhido}) \tag{4.62}$$

O valor do escalar b é computado ao final de cada etapa, de modo que as condições KKT sejam satisfeitas por ambos os exemplos otimizados.

Heurísticas

Existem duas heurísticas no algoritmo SMO original, mais a heurística "shrinking", incorporada em sua revisão em [Pla99].

Para a escolha de α_1, o algoritmo se concentra nos exemplos que têm mais possibilidade de violar as condições KKT, i.e., o conjunto de vetores de suporte não limitados (os que possuem $0 < \alpha_i < C$). Isso se dá pois à medida em que o algoritmo prossegue, os vetores que estão limitados tendem a continuar limitados enquanto os que não estão tendem a se mover, à medida em que novos exemplos são otimizados.

Uma vez que α_1 é escolhido, o SMO escolhe o α_2 de modo a maximizar o salto realizado durante a otimização conjunta (4.61): $|E_1 - E_2|$. O algoritmo mantém um cache do valor E para cada vetor de suporte não limitado do conjunto de treinamento e então escolhe o erro para maximizar o passo: se E_1 é positivo, será escolhido o exemplo com o menor erro E_2 e vice-versa.

5. *Minimum Description Length* e Código de Huffman

Neste capítulo será mostrado o princípio da Minimum Description Length (MDL) e como ele pode ser utilizado em combinação com um código universal para realizar classificação de dados em geral. Será analisado também um destes códigos universais, a Codificação Adaptativa de Huffman, que em combinação com o princípio MDL produz o algoritmo de classificação CAH+MDL, alvo desta pesquisa.

5.1 O Princípio *Minimum Description Length*

O Princípio da Descrição de Tamanho Mínimo ou, originalmente, *Minimum Description Length* (MDL) foi concebido inicialmente por Rissanen [Ris78] para ser usado na seleção de modelos e posteriormente generalizado para a aprendizagem por máquina. Ele é baseado no entendimento de que qualquer regularidade em quaisquer dados pode ser utilizada para comprimir tais dados, i.e., descrevê-los utilizando menos símbolos que os necessários para descrevê-los de forma literal [Grü05]. Quanto maior for a regularidade, mais os dados podem ser comprimidos.

Traçando uma analogia com a teoria de aprendizagem, pode-se dizer que quanto mais se consegue comprimir determinados dados, mais regularidades foram encontradas e, portanto, mais se conhece sobre estes dados. Desta forma, pode-se estender o princípio da MDL para a classificação de dados.

5.1.1 MDL de Duas Partes

Tradicionalmente, o princípio MDL é visto como composto de duas partes independentes: modelos e dados. Assim sendo, para determinados dados, o modelo cuja soma de seu tamanho com o tamanho dos dados comprimidos seja a menor é selecionado. Este critério de seleção de modelo faz naturalmente o balanço entre sua complexidade e o grau sob o qual os dados se encaixam nele, apresentando uma idéia similar ao princípio de Minimização de Risco (*Risk Minimization*) proposto por Vapnik [Vap82], que é a teoria na qual o funcionamento das *Support Vector Machines* está baseado. Desta forma, a classificação de dados através do MDL está automaticamente imune, ou pelo menos extremamente pouco vulnerável, a *overfitting*, igualmente ao caso das SVMs.

O problema do MDL de duas partes é que ele não especifica como o modelo deve ser codificado. Este problema, entretanto, foi solucionado por Rissanen [Ris96] com a criação do MDL refinado, que será descrito na próxima seção.

5.1.2 MDL Refinado

Na evolução do MDL, Rissanen [Ris96] propôs o uso de códigos universais para medir o comprimento da descrição de cada modelo, que era o que faltava no formulação do princípio original.

Um código universal possui a propriedade de comprimir os dados praticamente tão bem quanto o melhor modelo para determinada classe. Rissanen demonstrou que a diferença de compressão entre o melhor modelo e um código universal qualquer cresce de forma inferior à linear em relação ao comprimento da seqüência a ser comprimida.

Desta forma, qualquer código pode ser utilizado, desde que seu tamanho seja fixo. Em particular, códigos adaptativos são também considerados universais e, portanto, podem ser utilizados. A vantagem dos códigos adaptativos é que eles são criados à medida em que os dados são analisados, não sendo necessário conhecimento prévio.

5.1.3 MDL para Classificação de Dados

Os primeiros a experimentarem com algoritmos de compressão para a classificação de dados foram Frank et al. [FCW00], que utilizaram o método de compressão PPM para a criação de modelos e o teorema de Bayes para a atribuição de probabilidades. Eles obtiveram resultados superiores aos obtidos pelo algoritmo Naïve Bayes, porém bastante inferiores aos obtidos por uma *Support Vector Machine* linear.

A utilização da Codificação Adaptativa de Huffman para a filtragem de SPAM foi apresentada pela primeira vez por Zhou et al. [ZMN05] para a filtragem de SPAM, com excelentes resultados quando comparados a outros métodos de filtragem.

O uso do princípio MDL diretamente para a classificação de dados, entretanto, foi proposto ainda mais recentemente, por Bratko et al. [BFC$^+$06], também para a filtragem de SPAM. Eles utilizaram os algoritmos PPM e DMC junto com o princípio MDL, obtendo bons resultados em comparação com outros métodos e produtos de filtragem de SPAM, segundo avaliação feita pelos autores.

Braga e Ladeira [BL07] posteriormente estudaram o uso do MDL junto com a codificação Adaptativa de Huffman (ver Seção 5.2), igualmente para a filtragem de SPAM, e ele também obteve bons resultados quando comparados a outros métodos.

5.2 Codificação Adaptativa de Huffman

A Codificação Adaptativa de Huffman ou Código de Huffman é um código de tamanho variável criado por D. A. Huffman [Huf52], originalmente concebido para a compressão de dados na transmissão de mensagens. Na sua formulação original, ele transformava códigos de representação de caracteres de tamanho fixo (ASCII ou EBCDIC, que usavam 7 ou 8 bits para a representação de cada caractere) em códigos de tamanho variável, onde caracteres que aparecessem com mais freqüência em uma dada mensagem receberiam representações com menos bits que caracteres menos freqüentes, diminuindo, portanto, o tamanho da mensagem a ser transmitida.

O algoritmo original efetuava duas passagens sobre os dados, uma para contar a freqüência de cada caractere e outro para gerar uma árvore binária onde todos os caracteres utilizados estariam representados de acordo com sua freqüência: os mais utilizados

estariam próximos à raiz enquanto os pouco utilizados se encontrariam mais distantes. Nesta árvore apenas as folhas conteriam caracteres, enquanto os nós internos seriam rotulados simplesmente com a soma da freqüência de suas sub-árvores. O código a ser utilizado para cada caractere seria o caminho percorrido desde a raiz até a folha onde ele estaria representado, adicionando-se um 0 todas as vezes que se tomasse o caminho da sub-árvore da esquerda e 1 todas as vezes que se tomasse o da direita.

Além do maior gasto de processamento em virtude da realização de duas leituras dos dados, o algoritmo original possuía também o inconveniente de exigir que a árvore gerada fosse enviada do transmissor para o receptor, no caso de transmissão de dados, o que também o fazia perder tempo.

Alguns algoritmos foram propostos posteriormente para criação da árvore em um único passo. Destes, o mais utilizado e o único que garante a construção de uma árvore de tamanho mínimo é o proposto por Vitter [Vit87]. Ele também possui a virtude de não ser necessária a transmissão da árvore, já que ambos, o emissor e o receptor, iniciam com uma árvore vazia e a vão modificando de forma idêntica, de modo a manterem ambas sempre sincronizadas.

Um exemplo de uma árvore de Huffman montada para o alfabeto $\Sigma = [A..G]$, cujas freqüências individuais dos caracteres são: A = 22, B = 4, C = 3, D = 7, E = 6, F = 10, G = 5 está mostrada na Figura 5.1. Nesta árvore, a palavra "AADFAGE", seria codificada com a seguinte seqüência de bits: 0 0 100 110 0 1110 1111. Desta forma, a palavra original que necessitava de 56 bits para ser transmitida passa a necessitar de somente 17 bits, uma compressão de 70%.

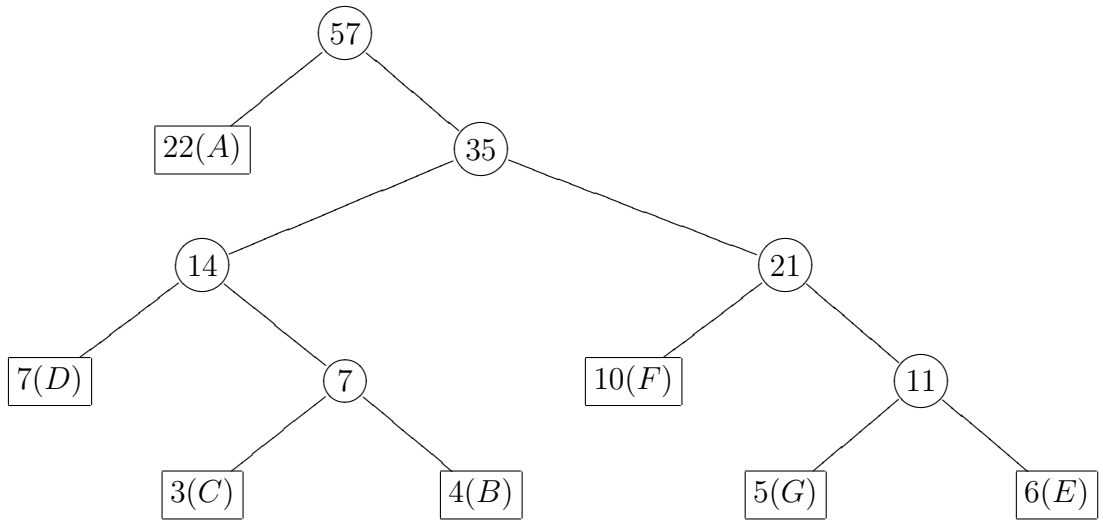


Figura 5.1: Exemplo de árvore de Huffman para o alfabeto $\Sigma = [A..G]$

5.3 Algoritmo CAH+MDL

O Algoritmo CAH+MDL é constituído pela combinação direta do princípio MDL com a Codificação Adaptativa de Huffman, exceto pelo fato de que esta última é estendida para

trabalhar com palavras ao invés de caracteres. Ele possui os seguintes pontos fortes:

- O mesmo algoritmo pode ser utilizado para realizar classificações binárias, multi-classe e multi-label, sem nenhuma complexidade adicional;
- Ele suporta treinamento incremental, desta forma pode ser utilizado em tarefas que requerem classificação e atualização da base de conhecimento em tempo real;
- O algoritmo consegue tratar diretamente problemas multi-label, com tempo de treinamento linear em relação ao número de categorias, independentemente de suas possíveis combinações. Isso não ocorre com algoritmos que necessitam decompor o problema de classificação multi-label em problemas menores binários ou multiclasse, conforme descrito na Seção 2.2;
- Ele possui implementação simples.

5.3.1 Treinamento

O algoritmo CAH+MDL pode funcionar de duas formas: sob a hipótese de mundo fechado, i.e., cada documento deve pertencer a pelo menos uma categoria, e com a hipótese de mundo aberto, onde esta restrição não existe. Neste último caso, é possível que um documento não pertença a nenhuma das categorias conhecidas pelo classificador e, assim sendo, ele é marcado como indefinido. O treinamento sob cada uma das duas hipóteses está explicado nas duas subseções a seguir.

Hipótese de Mundo Fechado

Na fase de treinamento, inicialmente todos os documentos são tokenizados e transformados em representação vetorial (VSM ou *Vector Space Model*), utilizando-se a frequência absoluta dos termos, conforme descrito na Seção 3.1.2. Neste modelo, cada dimensão dos vetores resultantes representa uma palavra ou termo distinto e seu valor é a contagem de suas ocorrências no documento. Entretanto, para impossibilitar que um único documento possa atribuir muita importância a um termo, i.e., um termo aparecendo um grande número de vezes em um único documento e, portanto, fazendo com que o algoritmo o considere muito importante, o número máximo de repetições de um termo por documento pode ser configurado como um parâmetro do algoritmo.

Após todos os documentos serem convertidos na sua representação vetorial, são construídas k árvores de Huffman estendidas, uma para cada uma das categorias possíveis. Estas árvores são extensões da formulação original na medida em que, ao invés de caracteres, cada uma das folhas é rotulada com uma palavra. As freqüências utilizadas nas árvores de Huffman tradicionais são usadas de forma idêntica aqui, com a distinção que elas representam as freqüências de aparição de cada uma das palavras presentes no conjunto de documentos de treinamento de cada classe, i.e., cada palavra é contada como se todos os documentos usados para treinar uma classe fossem concatenados para formar um único, grande, documento.

As árvores de Huffman estendidas são construídas através de uma versão modificada do algoritmo de Vitter [Vit87], que permite seu treinamento on-line e incremental, fazendo com que o algoritmo seja bastante útil para aplicações onde o conjunto de dados de

treinamento sofre constantes mudanças, que é o caso, dentre outros, da filtragem de SPAM e de sites Web. Esta versão modificada do algoritmo de Vitter inclui a possibilidade de remoção de documentos das árvores (o original permite apenas sua inclusão), sendo que ela é importante por duas razões: se o algoritmo foi treinado manualmente e houve um erro de treinamento (por um usuário clicar no botão errado, por exemplo), ele pode ser desfeito. Além disso, para calcular as distribuições de probabilidade (que serão explicadas mais a frente, nesta seção), é necessário se remover o documento sendo analisado de todas as árvores antes de calcular os dados gerados por ele.

Após as árvores com as frequências serem criadas, os documentos do treinamento são utilizados uma vez mais pelo algoritmo, de forma a calcular o número de bits necessário para "comprimir", i.e., codificar, cada documento por cada árvore. Para realizar este procedimento usa-se o método de Validação Cruzada Deixando Um de Fora (no original, *Leave-one-out Cross-Validation*) onde cada um dos documentos é removido de todas as árvores às quais ele pertence antes do cálculo ser feito e incluído novamente após ele ser concluído.

O procedimento completo implica em que todos os termos presentes em um documento são codificados por cada uma das árvores e os números de bits necessários para suas codificações são multiplicados pelo seus números de ocorrências. Para exemplificar, se uma determinada palavra aparece quatro vezes em um documento e são necessários 17 bits para codificá-la por uma das árvores, o número total de bits necessário para codificar as quatro instâncias por esta árvore será 68. Desta forma, para se obter o número total de bits necessário para codificar um documento por cada uma das árvores, basta somar o número de bits de codificação de cada dimensão do vetor que representa o documento, i.e, as palavras individuais vezes seu número de ocorrências no documento.

Para cada documento, após ele ter sido removido de todas as árvores correspondentes às categorias às quais ele pertence e o número de bits necessário para codificar todas as suas palavras por cada uma das árvores de Huffman ter sido calculado, o algoritmo seleciona o menor e o segundo menor número de bits e divide o primeiro pelo segundo para produzir um número R, $0 < R \leq 1$. Esta razão R indica o quão próxima uma da outra foi a classificação do documento pelas duas árvores mais adequadas: um valor próximo de 1 significa que o documento tem uma probabilidade maior de pertencer às categorias correspondentes às duas árvores, enquanto que um valor próximo de 0, indica que é bastante improvável que de fato o documento pertença a ambas.

Para ilustrar este procedimento, vamos supor que exista um classificador capaz de identificar 5 categorias (o que significa que ele possui 5 árvores de Huffman estendidas) e que exista um documento representado no formato VSM por um vetor $\vec{x}_i$. Cada dimensão deste vetor consiste no número de ocorrências de uma palavra diferente no documento. Este vetor é então codificado por cada uma das 5 árvores e elas produzem, muito provavelmente, números diferentes de bits. Suponhamos, para exemplificar, que os números de bits produzidos pelas árvores sejam $T_1 = 1714$, $T_2 = 3858$, $T_3 = 4649$, $T_4 = 3892$ e $T_5 = 2948$. O algoritmo então selecionaria o menor valor (1714) e o dividiria pelo segundo menor (2948), produzindo $R = 0,581$.

Após a razão R ser calculada para um documento, ela é usada da seguinte maneira:

- Se o documento não pertencer à categoria cuja árvore produziu o menor número de bits, ela é simplesmente descartada;

- Se o documento pertencer apenas à categoria que corresponde à árvore que produziu o menor número de bits, ela é adicionada a um conjunto rotulado com esta categoria;
- Se o documento pertencer a duas categorias (desde que uma delas seja a categoria cuja árvore produziu o menor número de bits) ela é adicionada a um conjunto rotulado com os nomes de ambas as categorias (que é um conjunto distinto dos conjuntos com os nomes das duas categorias separadas).

É importante destacar que, apesar da descrição do algoritmo estar sendo feita em termos de uma ou duas categorias, o procedimento acima pode ser generalizado para qualquer número delas. Por exemplo, se um documento pertencesse a três categorias, duas razões R_1 e R_2 seriam calculadas. R_1 seria o resultado da divisão do menor número de bits pelo segundo menor e R_2 seria obtido a partir da divisão do segundo menor número pelo terceiro menor. R_1 seria então adicionado ao conjunto rotulado com as duas primeiras categorias (as duas cujas árvores produziram os dois menores valores do número de bits necessário à codificação do documento) e R_2 seria incluído em um conjunto rotulado com as três.

Após todas as razões serem calculadas e adicionadas aos conjuntos correspondentes, o algoritmo terá uma lista completa de conjuntos: um para cada categoria individual e um para cada combinação distinta de categorias que existe nos documentos de treinamento. Ele então ajusta cada um destes conjuntos a uma distribuição de probabilidade do sistema Johnson (Johnson S_U, Johnson S_B e Johnson S_L, que é equivalente à distribuição Log-Normal). Os parâmetros destas distribuições são calculados a partir de uma versão ligeiramente modificada do algoritmo de adequação de percentuais desenvolvido por Slifker et al. [SS80]. A única diferença é que diversos valores do parâmetro "z" (conforme descrito no algoritmo) são testados e o melhor valor é selecionado através de testes Chi-square. A vantagem do sistema Johnson é que ele permite aproximar de forma bastante precisa através de uma de suas formas um grande número de distribuições contínuas padrão (Gama, Normal, Log-Normal, Beta, Exponential, entre outras) [SH68], permitindo, portanto, uma implementação mais simples do sistema de ajuste de distribuição de probabilidades do algoritmo.

Um último aspecto a ser destacado no treinamento é que, uma vez que as árvores e os conjuntos com as razões R forem obtidos, qualquer treinamento subsequente será feito em tempo real, já que apenas as árvores correspondentes às categorias às quais um novo documento a ser treinado pertencer serão modificadas e apenas uma nova razão R terá que ser calculada e inserida em um conjunto. A inclusão do novo documento nas árvores é feita em tempo linear pelo algoritmo de Vitter e o cálculo das novas distribuições de probabilidades é feito em tempo constante pelo algoritmo de Slifker et al. Isso faz com que, mesmo que eventualmente o tempo de treinamento inicial do algoritmo possa ser maior em comparação com outros, ele seja ideal para aplicações cujos dados de treinamento serão atualizados com o passar do tempo.

Hipótese de Mundo Aberto

O treinamento do algoritmo CAH+MDL para a situação de mundo aberto, i.e, onde não existe a obrigação de se atribuir pelo menos uma categoria para cada documento, é feito da mesma maneira que na hipótese de mundo fechado, porém com a seguinte adição: para

cada documento sendo treinado, além da razão R, é calculado também o número médio de bits por termo ou palavra, M, para cada uma das categorias às quais ele pertença. Este cálculo é feito simplesmente dividindo-se o número de bits obtidos por cada árvore de Huffman estendida pelo número de termos presentes no documento. Para cada categoria à qual o documento pertença, o número médio de bits por palavra correspondente, M, é então inserido em um conjunto rotulado com M_i, $1 \leq i \leq q$, onde q é o número de categorias conhecidas pelo classificador e $i \in \{1, 2, \cdots, q\}$ é o número correspondente à categoria sendo analisada.

Voltando ao exemplo anterior de um classificador capaz de identificar 5 categorias, suponhamos que os números de bits produzidos pelas árvores estendidas na classificação de um documento que pertença às categorias 1 e 5 sejam $T_1 = 1714$, $T_2 = 3858$, $T_3 = 4649$, $T_4 = 3892$ e $T_5 = 2948$ e que o número de palavras contidas no documento seja 143 (neste caso não faz diferença se as palavras são distintas ou se há repetições de uma ou várias palavras). Depois de calcular o valor de R, que seria 0,581 como já mostrado na subseção acima, o algoritmo então calcularia o valor de M, da forma descrita no parágrafo anterior, para as categorias 1 e 5 (que são as categorias às quais o documento pertence). Ele obteria $M_1 = 11,98$ e $M_5 = 20,61$. Os dois valores M_1 e M_5 seriam então incluídos nos conjuntos rotulados com seus nomes.

Após os valores médios de bits por palavra, M, serem obtidos para todos os documentos do conjunto de treinamento, o classificador então ajustaria cada um dos conjuntos M_i, $1 \leq i \leq q$, onde q é o número de categorias conhecidas pelo classificador, a uma distribuição de probabilidade do sistema Johnson, através de uma versão modificada do algoritmo de Slifker et al, da mesma forma como descrito na subseção anterior. Novamente, a escolha apenas da família Johnson é devido a sua versatilidade e para simplificar a implementação do módulo de ajuste de distribuições do algoritmo.

5.3.2 Classificação

A classificação de documentos pelo algoritmo CAH+MDL pode funcionar de duas maneiras, dependendo do método escolhido para realizar seu treinamento: sob a hipótese de mundo fechado, i.e., cada documento deve pertencer a pelo menos uma categoria, e com a hipótese de mundo aberto, onde esta restrição não existe. Os procedimentos necessários para a classificação de um documento sob cada uma das duas hipóteses está explicado nas duas subseções a seguir.

Hipótese de Mundo Fechado

Para classificar um documento, o algoritmo inicialmente calcula os números de bits necessários para codificá-lo em cada uma das árvores estendidas de Huffman. Para obter estes números, ele pesquisa cada termo do documento sendo classificado em todas as árvores. Se ele é encontrado, o número de bits que seria usado para sua codificação (o caminho do nó raiz da árvore até a folha que o contém) é adicionado no total da árvore (neste caso, é importante levar em consideração que não é necessário se preocupar com o código designado a cada termo, apenas com o número de bits que ele possui). Caso um termo ou palavra não esteja presente em uma árvore qualquer, o maior código atribuído a um termo por todas as árvores, mais uma penalidade configurável como parâmetro do algoritmo é adicionado ao total da árvore. O objetivo de se atribuir o maior código possível entre

todas as árvores mais uma penalidade fixa é para igualar as condições entre árvores de diferentes tamanhos. Se a penalidade fosse dependente de cada árvore, árvores menores (que conhecem menos termos porém atribuem códigos menores para os termos conhecidos) teriam vantagem sobre árvores maiores em relação a termos desconhecidos, já que a penalidade que elas atribuiriam a estes termos seria menor.

Quando todos os termos são verificados, independentemente do problema de classificação, i.e., binário, multiclasse ou multi-label, a categoria cuja árvore produziu o menor número de bits é atribuída ao documento.

Caso se esteja trabalhando com classificação multi-label, após a atribuição desta primeira categoria, ainda é necessário que o classificador verifique se categorias adicionais também devem ser atribuídas ao documento. Para tanto, ele calcula a razão R dividindo o menor número de bits pelo segundo menor número, conforme descrito na Seção 5.3.1. Neste caso, para decidir se uma segunda categoria será ou não atribuída, ele procede da seguinte maneira:

- Se não existe nenhuma distribuição de probabilidade rotulada conjuntamente com o nome da categoria que já foi atribuída ao documento (a que corresponde à árvore que conseguiu compactá-lo mais) e com a categoria cuja árvore produziu o segundo menor número de bits, então o documento é rotulado apenas com a primeira categoria (isso significa que não é possível que um documento receba duas categorias a não ser que exista ao menos um documento no conjunto de dados de treinamento rotulado com ambas categorias cujas árvores produziram os menores números de bits para sua codificação).
- Se existe uma distribuição de probabilidade com ambas as categorias cujas árvores obtiveram o menor número de bits, o algoritmo calcula a função de densidade de probabilidade $f(x)$ das duas distribuições de probabilidade para o valor de $x = R$: a distribuição com apenas a primeira categoria e a rotulada com ambas. A segunda categoria é atribuída ao documento apenas se a função de densidade de probabilidade para a distribuição rotulada com ambas as categorias resultar em um valor maior do que o correspondente apenas à primeira categoria. A Figura 5.2 mostra um exemplo real de um gráfico das funções de densidade de probabilidade, em um classificador capaz de reconhecer 7 categorias, para apenas a categoria 7 (a curva à esquerda) e para as categorias 5 e 7 (a curva à direita). Neste exemplo, supondo que que a árvore correspondente à categoria 7 foi a que produziu o menor número de bits para a codificação do documento e que a árvore correspondente à categoria 5 foi a que resultou no segundo menor número, então se a razão R para um determinado documento for menor que 0,81 apenas a categoria 7 será atribuída a ela, caso contrário, ele será rotulado com as categorias 5 e 7.

De maneira similar à mostrada na Seção 5.3.1, o procedimento acima descrito pode ser generalizado para qualquer número de categorias. Neste caso, primeiro o classificador decide se uma segunda categoria deve ser atribuída a um determinado documento, baseando-se na razão R_1, que é o resultado da divisão do menor número de bits necessários para codificá-lo por uma das árvores pelo segundo menor número de bits. Se uma segunda categoria é atribuída, então o classificador decide se uma terceira deve ou não ser também atribuída, desta vez baseando-se na segunda razão R_2, obtida a partir da

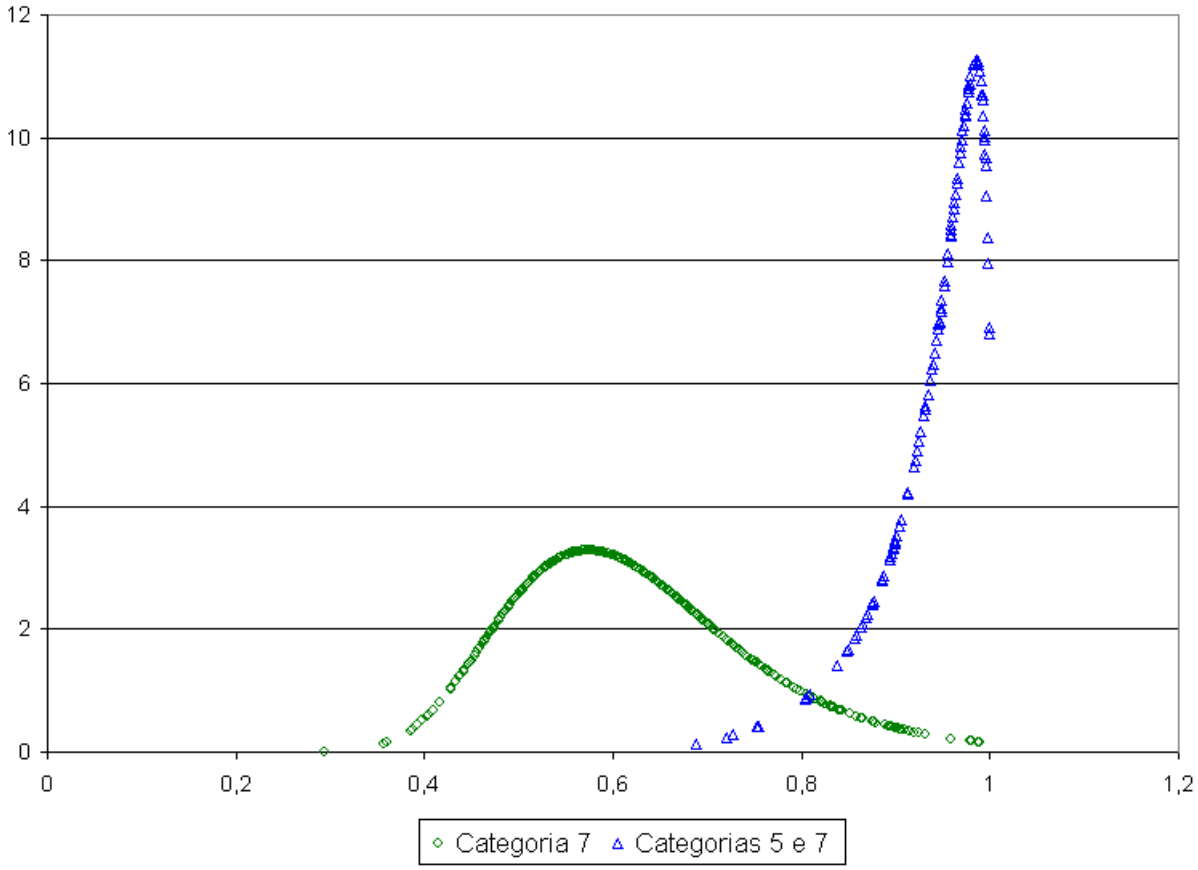


Figura 5.2: Funções de densidade de probabilidade para as categorias 7 e 5/7

divisão do segundo menor número de bits necessários para a codificação do documento pelo terceiro menor número e assim sucessivamente.

Hipótese de Mundo Aberto

A classificação de documentos pelo algoritmo CAH+MDL para a situação de mundo aberto, i.e, onde não existe a obrigação de se atribuir pelo menos uma categoria para cada documento, só faz sentido para problemas de classificação multi-label. Isso se dá já que no caso de problemas binários ou multiclasse uma categoria é sempre atribuída a todos os documentos. Neste caso, atribui-se a cada documento a categoria cuja árvore de Huffman estendida codificá-lo com o menor número de bits, como mostrado na subseção anterior.

Para os problemas de classificação multi-label na hipótese de mundo aberto, o classificador se comporta exatamente da mesma forma que foi descrita na subseção anterior, com uma única diferença: ao invés de automaticamente atribuir ao documento a categoria cuja árvore obteve a melhor codificação (menor número de bits), o classificador avalia se o resultado de sua codificação por esta árvore foi "suficientemente bom" (leia mais abaixo para obter a explicação do que se entende por esta expressão) para que a categoria correspondente a ela possa ser a ele atribuída.

Para decidir se deve-se atribuir ou não a primeira categoria (ou seja, se o documento será rotulado com esta categoria ou marcado como "indefinido", i.e., nenhuma categoria atribuída a ele) não se pode simplesmente treinar o algoritmo para reconhecer uma "categoria indefinida", pelas seguintes razões:

- Os documentos indefinidos, i.e, que não pertencem a nenhuma categoria, são bastante diferentes entre si. Devido a isso, se torna inviável a abordagem de se obter uma amostra deles para treinar o classificador e tratá-los como pertencentes uma categoria adicional, "indefinida".
- Ainda que fosse possível treinar o classificador com documentos desta "categoria indefinida", a quantidade de exemplos de cada tipo seria pequena (devido a grande quantidade de tipos) e a distribuição de termos ficaria muito espalhada, já que muitos tipos de documentos "indefinidos" podem não ter nada em comum uns com os outros. Isso faria com que a árvore da categoria "indefinida" tivesse códigos em geral maiores que as árvores das demais categorias e, como consequencia, apresentasse normalmente resultados inferiores de codificação (maiores números de bits), o que na prática faria com que muito poucos ou até menos nenhum documento fosse marcado como indefinido.
- Novamente, mesmo que fosse possível treinar o algoritmo em uma categoria indefinida, ela teria que ter comportamento diferente das demais: ela nunca poderia ser atribuída como categoria adicional (segunda, terceira, etc) e quando fosse atribuída como primeira, categorias adicionais não poderiam ser incluídas. Isso traria uma complexidade adicional ao algoritmo.

Devido aos problemas acima apontados, escolheu-se uma abordagem diferente para tratar a possibilidade de não se atribuir nenhuma categoria a determinado documento: após obter a árvore que consegue codificar o documento com o menor número de bits e determinar este número, o classificador calcula então o número médio de bits por termo ou palavra, M, que é obtido simplesmente pela divisão do número total de bits da melhor codificação do documento pelo número de termos nele presentes. Esta média M é então então comparada com o valor da função inversa de distribuição de probabilidade $F^{-1}(x)$ correspondente à M_y, onde $0 < x < 1$ corresponde a um intervalo de confiança arbitrário, definido como um parâmetro do algoritmo, e y ao número da categoria correspondente à arvore que obteve a melhor codificação do documento. Caso M seja maior que o valor de $F^{-1}(x)$, o documento é marcado como indefinido, caso contrário a primeira categoria é atribuída e o restante do algoritmo segue exatamente como o descrito na hipótese de mundo fechado.

A intuição por trás desta abordagem é a seguinte: independentemente do documento a ser classificado, sempre haverá uma árvore que produzirá o menor número de bits para sua codificação (se duas ou mais árvores produzirem exatamente o mesmo número de bits, situação possível mas altamente improvável, pode-se realizar o teste acima para uma delas e, caso a categoria correspondente não seja atribuída ao documento, repete-se para a(s) outra(s), até que uma seja atribuída ou todas sejam rejeitadas). Entretanto, apesar de haver uma melhor árvore, ela pode não ter conseguido codificar o documento de forma suficientemente boa para que a categoria correspondente lhe seja atribuída. Isso ocorre

na medida em que quanto mais um documento difere dos exemplos conhecidos pelo classificador, a tendência é de que o número de bits necessário para codificá-lo seja maior, já que muitos de seus termos não estarão presentes nas árvores ou, caso estejam, produzirão códigos maiores uma vez que sua frequencia de aparição será menor que dos termos característicos das classes. Isso produzirá um maior número médio de bits por termo que, mais provavelmente, estará fora do intervalo de confiança utilizado como parâmetro do algoritmo, trazendo como consequencia a marcação do documento como indefinido.

À medida em que se escolhe menores intervalos de confiança, mais documentos são marcados como indefinidos e menor é o Recall do classificador. O intervalo ideal vai depender da aplicação que o esteja utilizado: para aplicações onde busca-se selecionar ou classificar apenas uma amostra dos documentos com grande grau de acurácia (o que ocorre tipicamente em mineração de dados), pode-se utilizar um intervalo de confiança pequeno, fazendo com que o algoritmo tenha uma alta Precisão tanto na classificação das categorias conhecidas quanto na identificação de documentos indefinidos, porém apresentando um baixo Recall. Por outro lado, para aplicações onde se deseja apenas eliminar documentos que sejam substancialmente diferentes dos conhecidos, pode-se usar um intervalo de confiança maior, o que fará com que o Recall do classificador seja alto, porém baixando a Precisão na identificação de documentos indefinidos. A situação limite ocorre quando o intervalo de confiança é muito próximo a 1, fazendo com que nenhum documento seja marcado como indefinido e que o classificador se comporte como na hipótese de mundo fechado.

6. Problema e Metodologia

Neste capítulo, será descrito o problema que esta pesquisa está interessada em resolver, na Seção 6.1, seguida pela metodologia adotada para resolvê-lo, mostrada na Seção 6.2, e pelo domínio, descrito na Seção 6.2.1. Por fim, a Seção 6.2.2 mostra como os resultados serão avaliados e comparados.

6.1 Problema a ser resolvido

A grande quantidade de sites existentes na Internet dificulta a localização daqueles que são mais relevantes do ponto de vista de um usuário, implicando maior esforço cognitivo para a discriminação entre eles.

Nesta pesquisa, é feita a proposição de um novo algoritmo para classificação automática de páginas Web em categorias multi-label e em tempo real, levando em conta a possibilidade de cada site ser classificado como pertencente a um subconjunto de categorias específicas ou a nenhuma delas. O uso deste algoritmo pode contribuir para reduzir o esforço cognitivo dos usuários.

O algoritmo aqui proposto consiste em duas extensões ao algoritmo conhecido por CAH+MDL: a primeira para permitir a classificação de documentos textuais em categorias multi-label e a segunda para permitir que ele trabalhe sob a hipótese de mundo fechado, onde cada documento deve receber uma categoria, e mundo aberto, onde podem existir documentos que não recebem nenhuma categoria.

O algoritmo CAH+MDL, que originalmente permitia a classificação de documentos textuais em problemas de classificação multi-classe, foi estudado anteriormente por Braga [BL07] para a filtragem de SPAM, tendo obtido bons resultados quando comparados a outros métodos. Desta forma, as extensões propostas visam torná-lo mais completo e possibilitar seu uso em problemas mais complexos.

6.2 Metodologia a ser utilizada

Esta pesquisa será dividida em várias etapas, como descrito a seguir:

1. Geração de uma base de páginas Web manualmente classificadas em categorias multi-label, nos idiomas inglês e português.

   Isto é necessário já que não foi encontrada pelo autor nenhuma base de páginas Web multi-label que esteja disponível nos idiomas inglês ou português. Sendo assim, a etapa inicial consiste em se obter e classificar manualmente um número significativo

de sites, que posteriormente possam ser utilizados para treinar e analisar o algoritmo proposto.

2. Extração de atributos das páginas classificadas.

   Esta etapa consiste em acessar cada página Web e se obter e contar as palavras pertencentes a ela. Para cada site, termos obtidos no corpo da página bem como no título e nos metadados HTML serão contados e gravados em arquivos. Os termos provenientes de partes outras que do corpo da página, serão marcados com sua origem, de forma a possibilitar o estudo de como sua ponderação afeta a acurácia da classificação.

3. Implementação de um classificador CAH+MDL inicial.

   Consiste na implementação em linguagem Java de um classificador CAH+MDL básico, i.e., sem as extensões propostas nesta pesquisa, que fará o processo de classificação inicial de sites Web em categorias multi-classe, da mesma forma com que foi proposta por Braga et al. [BL07]. Os resultados deste classificador serão então analisados para possibilitar a proposição das extensões.

4. Parametrização do classificador CAH+MDL inicial.

   Nesta etapa, o classificador CAH+MDL desenvolvido na etapa anterior será parametrizado de modo a se obter a maior precisão possível para a categoria a ser atribuída a cada site por ele. Isso é fundamental já que a categoria inicial é utilizada como base na atribuição das categorias adicionais.

   Os parâmetros a serem estudados são os seguintes:

   - Número máximo de repetições de uma palavra por página Web
   - Penalidade que será atribuída a termos não presentes nas árvores de Huffman estendidas
   - Ponderação de termos provenientes do título da página e de seus metadados

5. Extensão do algoritmo CAH+MDL para classificação multi-label

   Aqui é proposta a extensão que permite que o algoritmo CAH+MDL atribua mais de uma categoria por página Web. Para tal, ele verifica se a probabilidade do site ser classificado em mais de uma categoria é maior do que em apenas uma, considerando apenas as combinações multi-label existentes na base de dados usada como treinamento.

6. Extensão do algoritmo CAH+MDL para a hipótese de mundo aberto

   Esta extensão permite que o algoritmo CAH+MDL não seja obrigado a sempre atribuir uma categoria para todos os sites sendo classificados, podendo marcar determinadas páginas como sendo "indefinidas". Para isso, será verificado se a menor nota gerada por uma das árvores de Huffman estendidas está contida em um grau de significância arbitrário do intervalo de confiança da classe correspondente a esta menor nota.

7. Implementação do algoritmo CAH+MDL estendido

   Nesta etapa são implementadas no classificador CAH+MDL inicial as duas extensões acima mostradas.

8. Comparação do desempenho experimental do classificador proposto

   Nesta etapa será feita a análise do desempenho experimental do classificador aqui proposto, CAH+MDL estendido, na classificação de sites da base de dados gerada na primeira etapa. Como não existe nenhum estudo anterior feito na base a ser utilizada que possa servir como parâmetro de comparação direta, optou-se por comparar o desempenho do classificador CAH+MDL estendido com um conjunto de classificadores binários SVM.

   Para a comparação dos resultados, são utilizadas as métricas Precisão, Recall e F_1 (F-measure), conforme descrito na Seção 2.4.

Após o término de todas as etapas anteriormente mostradas, o classificador resultante que fará o processo completo de classificação de sites Web e da geração das métricas de comparação será composto pelos seguintes grandes módulos, escritos em linguagem Java (a documentação detalhada do classificador desenvolvido se encontra na Seção A.3):

- **Tokenizador:** Será o responsável pela tokenização dos sites web a serem usados como treinamento ou classificados. Ele baixará os sites completos, através de conexões HTTP e os salvará em disco já tokenizados.

  Ele será capaz de entender toda a sintaxe HTML e de identificar o uso de comandos "Frames" e "IFrames", inclusive de forma recursiva. Quando estes comandos forem utilizados, as páginas destino de ambos será tokenizada como parte da página inicial.

  O tokenizador também será capaz de identificar o uso de comandos "Redirect", que indicam que uma página deve ser acessada a partir de outro endereço, para conseguir acessá-las no novo endereço.

- **Classificador SVM:** Será composto pelo classificador SVM e pelo treinamento deste classificador.

- **Classificador CAH+MDL:** Será composto pelo classificador CAH+MDL e pelo treinamento deste classificador.

- **Módulo Principal:** Identificará os comandos a serem executados e será capaz de gerar as estatísticas de classificação dos sites.

6.2.1 Domínio

Infelizmente, a única base pré-classificada de sites Web disponível na Internet para os idiomas inglês ou português que pôde ser identificada pelo autor é a WebKb [DLR00]. Esta base é composta apenas por sites de quatro universidades americanas, não é multi-label (é apenas multi-classe) e foi gerada entre 1997 e 2000, sendo, portanto, nem um pouco representativa da forma e conteúdo atuais dos sites da Internet.

Devido a isso, o treinamento e testes do classificador serão feitos em uma base de dados de sites Web, em português e inglês, composta de 3067 sites manualmente classificados pelo autor nas seguintes categorias:

- **Sexo Explícito:** Composta por sites de sexo em geral, sejam eles apenas textuais, e.g., sites de contos eróticos, ou que possuam textos e imagens. Esta é, teoricamente, uma das categorias mais fácil de ser identificada corretamente pelos classificadores em virtude de seu vocabulário bastante específico.
- **Material Adulto:** Composta por sites de acompanhantes, garotas de programa, *strippers*, ou de agências que oferecem tal serviço, desde que não apresentem imagens explícitas de sexo. Esta categoria se assemelha a de sexo no uso de vocabulário e está aqui com o objetivo de identificar se os classificadores conseguem separá-las corretamente.
- **Loja Virtual:** Esta categoria pode estar associada a qualquer uma das outras (para identificar uma loja que vende determinado tipo de produto) ou sozinha, para identificar uma loja genérica. O objetivo de sua inclusão é produzir uma boa quantidade de sites bastante diferentes entre si que compartilham uma mesma categoria, para verificar como se comportam ambos os classificadores.
- **Esportes:** Sites de esportes em geral, incluindo sites de notícias relacionadas exclusivamente ao tema e de apostas esportivas.
- **Jogos de Azar:** Sites de apostas, cassinos e jogos de azar em geral. No caso de sites de apostas em esportes, eles serão classificados nesta categoria e na de esportes, simultaneamente.
- **Jogos:** Sites de jogos tradicionais (tabuleiro, rpg, etc), para computadores ou celulares, on-line ou não.
- **Música:** Sites de música em geral, download de música ou mp3, instrumentos musicais, partituras, teoria musical e bandas.

Esta base foi criada com o objetivo de misturar categorias de fácil identificação juntamente com outras que possuem sobreposição de conteúdo, de modo a poder melhor avaliar os resultados de ambos algoritmos de classificação. Além disso, foram escolhidas categorias que permitam sua atribuição simultânea a um mesmo site, de modo a produzir uma base multi-label.

6.2.2 Como os resultados serão avaliados

Para a análise de resultados, será comparada a acurácia do classificador SVM linear, conhecido por ser um dos métodos que produz melhores resultados na classificação de padrões, especialmente para o caso de documentos textuais, com a acurácia do classificador CAH+MDL estendido. Desta forma, será possível verificar se o classificador CAH+MDL com as extensões propostas nesta pesquisa pode ser utilizado de forma satisfatória para a classificação de sites Web e, em especial, para categorias multi-label.

Para evitar que ocorra *overfitting* nos classificadores, i.e, fazer com que eles se ajustem extremamente bem aos dados do treinamento porém, por conta disso, percam capacidade de generalização, a análise de seus resultados será feita a partir do método de Validação Cruzada (ou *k-Fold Cross-Validation*, como é normalmente conhecido). Neste método, o conjunto total com os dados classificados manualmente, S, é dividido de forma randômica

em k subconjuntos mutualmente exclusivos, $S_1, S_2, \cdots, S_k$, de tamanho aproximadamente igual e com aproximadamente a mesma distribuição de categorias. Os procedimentos de treinamento e teste são então repetidos k vezes, onde para cada $i \in \{1, 2, \cdots, k\}$, o classificador será treinado com $S \backslash S_i$ e testado com S_i. Além disso, para poder comparar os resultados, ambos os classificadores serão treinados e testados com os mesmos conjuntos, $S_1, S_2, \cdots, S_k$. Nesta pesquisa, será usado o valor $k = 10$, que é o mais frequentemente utilizado.

Para a avaliação de cada classificador, serão calculados, para cada site classificado, o valor de Precisão, Recall e F_1, conforme mostrado na Seção 2.4. Será então feita uma tabela com os resultados de cada classificador em relação ao total de documentos avaliados, para cada uma das k passagens.

7. Experimentos Realizados

Este capítulo apresenta os diversos experimentos que foram feitos nesta pesquisa. Inicialmente, através de um experimento prévio, buscou-se verificar que para a classificação de documentos textuais e em especial para páginas Web uma *Support Vector Machine* linear produzia resultados melhores ou no mínimo tão bons quanto uma não linear (isso é importante já que a primeira seria usada como base comparativa para a avaliação dos resultados do algoritmo CAH+MDL). Este experimento está descrito na Seção 7.1.

O experimento seguinte, mostrado na Seção 7.2, buscou avaliar o quanto o aumento de peso para termos extraídos de partes semi-estruturadas de páginas HTML (metadados e título) contribuiriam para aumentar a precisão de sua classificação pelo algoritmo CAH+MDL.

O próximo experimento buscou avaliar se o algoritmo CAH+MDL sob a hipótese de mundo fechado, como descrito na Seção 5.3.1, apresentaria resultados de classificação de páginas Web multi-label suficientemente bons. Para esta avaliação, seus resultados foram comparados com os resultados de uma SVM linear na classificação dos mesmos dados. O experimento está descrito em detalhes na Seção 7.3.

Por fim, foi feita a avaliação do algoritmo CAH+MDL na classificação da mesma base de dados, porém com o relaxamento da hipótese do mundo fechado, i.e., com a adição de documentos que deveriam ser marcados como indefinidos. Este experimento está descrito na Seção 7.4.

7.1 Comparativo SVM Linear X Não Linear

A fim de verificar que uma SVM linear é adequada para a classificação de textos, i.e., produz resultados tão bons ou melhores que SVM não lineares, foi feito um experimento prévio de classificação da base Webkb [DLR00] com o uso do pacote WEKA [Wek] integrado com a LIBSVM [CL01].

O objetivo do experimento foi de comparar a acurácia do algoritmo de SVM linear e não linear, com kernels RBF e polinomial, com e sem normalização dos dados, para a classificação de páginas Web.

O experimento está descrito em detalhes na Seção 7.1.1 e os resultados na Seção 7.1.2. Por fim, a análise dos resultados é apresentada na Seção 7.1.3.

7.1.1 Descrição do Experimento

Para a realização do experimento, utilizou-se o pacote WEKA [Wek], que é uma coleção de algoritmos de mineração de dados e ferramentas estatísticas. Como este pacote não

suporta o uso de SVM lineares, apenas SVM com kernels polinomiais ou RBF, foi feita sua integração com a biblioteca LIBSVM [CL01], através da biblioteca adicional denominada WLSVM [EMH05]. A partir desta integração, a LIBSVM passou a estar disponível como um algoritmo de classificação adicional do pacote WEKA e pôde ser parametrizado para se definir o tipo de kernel utilizado e seus parâmetros adicionais. O uso de memória disponível para o WEKA teve que ser configurado manualmente de modo a passar do original 128M para 768M, caso contrário os experimentos não eram concluídos por falta de memória.

A base de dados utilizada foi a WebKb [DLR00], que consiste de páginas Web de quatro universidades americanas obtidas em 1997 e classificadas manualmente em sete categorias diferentes: *Project*, *Course*, *Faculty*, *Student*, *Department*, *Staff* e *Other*. Neste experimento, foi utilizada a versão disponibilizada por Cardoso-Cachopo [CC07], que já trazia as páginas Web tokenizadas e stemizadas.

Conforme descrito em [CC07], das sete categorias iniciais, foram removidas as *Department* e *Staff* em virtude delas apresentarem apenas um pequeno número de documentos. A categoria *Other* também foi removida em virtude das páginas pertencentes a ela serem bastante diferentes entre si. Em adição ao descrito pela autora, se removeu também uma instância da classe *Faculty* devido ao fato dela possuir um tamanho muito superior aos demais documentos.

A seguir, foi feita a conversão dos arquivos originais para dois outros, no formato arff esparso, que podem ser lidos diretamente pelo pacote WEKA: um arquivo para treinar os classificadores e outro para realizar seus testes. Na conversão, os arquivos resultantes mantiveram as mesmas instâncias dos arquivos originais e sua descrição está resumida na Tabela 7.1.

Classe	**Treinamento**	**Teste**	**Total**
Project	336	168	504
Course	620	310	930
Faculty	749	374	1123
Student	1097	544	1641
Total	**2802**	**1396**	**4198**

Tabela 7.1: Documentos da base Webkb nas quatro categorias estudadas

Cada um dos arquivos é composto de 7771 atributos, correspondentes às 7770 palavras únicas existentes nas páginas Web originais e sua classe.

Para realizar a transformação dos arquivos originais em formato arff esparso, foi escrito um programa com esta finalidade. O algoritmo de *stemming* utilizado originalmente foi o desenvolvido por Porter [Por80], considerado como padrão de facto para a língua inglesa.

A seguir, realizou-se o treinamento e testes de classificadores SVM linear, polinomial (com grau 3) e RBF, usando seis valores diferentes para o parâmetro C, de modo a verificar empiricamente qual deles produz os melhores resultados para cada tipo de kernel. Em adição, cada um dos classificadores foi testado também com e sem a normalização dos vetores de representação dos documentos.

7.1.2 Resultados Obtidos

Foram feitos 36 experimentos diferentes com os mesmos dados, variando-se o tipo de kernel, o parâmetro C e a normalização ou não dos dados. Os percentuais de documentos classificados corretamente através dos testes, para cada tipo de kernel, parâmetro C e uso ou não de normalização, estão resumidos nas tabelas 7.2 e 7.3.

	Sem Normalização					
Kernel	$C = 0,01$	$C = 0,1$	$C = 1$	$C = 10$	$C = 100$	$C = 1000$
Linear	**88,75%**	87,75%	84,96%	84,53%	84,53%	84,53%
Polinomial	38,97%	38,97%	39,61%	40,11%	42,62%	45,91%
RBF	38,97%	43,27%	61,89%	86,10%	**88,75%**	86,39%

Tabela 7.2: Percentual de documentos classificados corretamente sem normalização dos dados

	Com Normalização					
Kernel	$C = 0,01$	$C = 0,1$	$C = 1$	$C = 10$	$C = 100$	$C = 1000$
Linear	51,29%	82,23%	**85,24%**	83,24%	80,73%	80,01%
Polinomial	38,97%	38,97%	38,97%	38,97%	38,97%	38,97%
RBF	38,97%	38,97%	38,97%	40,76%	66,98%	83,60%

Tabela 7.3: Percentual de documentos classificados corretamente com normalização dos dados

Destas duas tabelas, pode-se verificar que os melhores resultados foram obtidos por duas combinações de kernel e parâmetro C: Linear com $C = 0,01$ e RBF com $C = 100$, ambos sem o uso de normalização, que atingiram o percentual idêntico de 88,75% dos documentos classificados corretamente.

Com a normalização dos dados, os resultados de classificação foram significativamente piores, para praticamente todos os casos. A única exceção foi com o uso do kernel Linear, com $C = 1$, que apresentou resultado ligeiramente melhor do que o obtido com esta mesma configuração sem o uso de normalização: 85,24% de documentos classificados corretamente. Este resultado, de qualquer forma, ainda é inferior a melhor acurácia obtida sem o uso de normalização, que foi de 88,75%.

A grande surpresa foi o kernel polinomial, que apresentou resultados extremamente ruins e muito inferiores aos obtidos com a SVM linear e a que utiliza o kernel RBF. Além disso, o kernel polinomial foi o único que não apresentou variações significativas de acurácia com a variação do parâmetro C.

7.1.3 Análise dos Resultados

Os resultados do experimento prévio foram interessantes: por um lado, foi confirmado que uma SVM linear classifica dados textuais tão bem, e até melhor, quando comparada com a que utiliza kernel RBF. Por outro lado, houve uma certa surpresa ao se constatar que os resultados de classificação dos dados normalizados foi bastante inferior ao dos dados

sem normalização, especialmente com o kernel RBF. Se esperava, com análises teóricas, justamente o oposto já que em documentos normalizados se despreza seu comprimento e isso, teoricamente, produziria uma melhor acurácia do classificador, que acabou se mostrando totalmente falsa na prática.

Os resultados também foram interessantes ao mostrar que a acurácia de uma SVM depende da combinação formada pelo tipo de kernel e do parâmetro C e que não existe um valor absoluto "ótimo" para este parâmetro, já que ele depende fundamentalmente do tipo de kernel usado.

A SVM linear, por apresentar acurácia igual a RBF (e superior no caso dos dados normalizados), se mostra uma excelente alternativa para a classificação de documentos textuais e páginas Web, já que ela apresenta duas vantagens em relação às não lineares:

- SVM lineares exigem muito menos memória para seu treinamento, já que é possível se armazenar apenas o vetor de coeficientes α_i em memória, ao contrário das não lineares que exigem que uma boa parte da matriz $N_{vs} \times N_{vs}$ fique armazenada.
- Ainda mais importante que o item anterior, é o fato de ser possível, para SVM lineares, se compor o vetor $\vec{w}$ a partir dos seus coeficientes α_i e, desta forma, se obter uma função de decisão (4.24) que é N_{vs} vezes mais rápida que a função de decisão equivalente para SVM não lineares (4.29). Como em geral o número de vetores de suporte está na casa das dezenas ou centenas, isso faz com que as SVM lineares sejam bastante mais rápidas que as não lineares e propiciam seu uso em aplicações onde a velocidade seja fundamental.

Diante disso, a comparação da acurácia do algoritmo CAH+MDL com uma SVM linear se mostra bastante interessante, já que se sua acurácia for próxima ou superior a obtida pela SVM, ele se tornará uma excelente alternativa para a classificação de documentos multi-label.

Outra conclusão que se obtém facilmente é que SVM com kernel polinomial não são adequadas para a classificação de textos nem páginas Web, já que possuem baixa acurácia e baixa velocidade, quando comparadas às SVM lineares.

7.2 Aumento do Peso para Meta-Dados

O algoritmo CAH+MDL possui dois parâmetros configuráveis, conforme descrito na Seção 5.3: o número máximo de repetições de um termo por documento e a penalidade adicionada a um termo ou palavra que não esteja presente em uma árvore no momento de calcular o número de bits necessários para sua codificação.

Este experimento foi realizado com o objetivo de se avaliar as combinações de número máximo de repetições e de penalidade, bem como a influência do aumento do peso, i.e., número de ocorrências, atribuído a termos oriundos de partes semi-estruturadas de páginas HTML (meta-dados e título, como descritos na Seção 3.6.2) que produziam os melhores resultados na classificação de uma base de páginas Web. Sua descrição e os resultados obtidos estão mostrados em detalhes na Seção 7.2.1 e a análise dos resultados na Seção 7.2.2.

7.2.1 Descrição e Resultados do Experimento

Para a realização deste experimento não foi possível a utilização da base WebKb, usada no experimento anterior, já que por esta ser muito antiga pouquíssimas páginas possuíam meta-dados. Diante disso, utilizou-se uma base de páginas Web composta de sete categorias e manualmente classificada pelo autor. Esta mesma base também foi posteriormente utilizada nos dois experimentos subsequentes, descritos nas Seções 7.3 e 7.4, e possui a distribuição de categorias e documentos mostrada na Tabela 7.4. Os critérios de classificação adotados pelo autor para a atribuição das categorias de cada página estão descritos na Seção 6.2.1.

Como descrito na Seção 6.2.2, para evitar que ocorresse *overfitting* nos classificadores, esta tabela foi separada de forma randômica em 10 pares de arquivos mutuamente exclusivos, de forma a permitir a realização da validação cruzada de 10 partes (*10-Fold Cross-Validation*). Cada um destes 10 pares de arquivos consistia de um arquivo de treinamento, com cerca de 90% dos documentos, e um arquivo de teste, com os 10% restantes. Ao separar a base de dados, todos os arquivos de treinamento e teste mantiveram tanto quanto possível a mesma proporção de categorias da base completa.

Categorias	**Descrição**	**Número de Documentos**
1	Material Adulto	502
2	Jogos de Azar	425
2, 3	Jogos de Azar / Esportes	76
3	Esportes	340
3, 5	Esportes / Loja Virtual	86
4	Jogos	418
4, 5	Jogos / Loja Virtual	84
5	Loja Virtual	130
5, 6	Loja Virtual / Música	94
5, 7	Loja Virtual / Sexo Explícito	120
6	Música	407
7	Sexo Explícito	385
Total		**3067**

Tabela 7.4: Base de páginas Web usada nos experimentos das Seções 7.2 e 7.3

Neste experimento, buscou-se apenas a realização da classificação de documentos em uma única categoria (as categorias adicionais foram atribuídas apenas nos experimentos 7.3 e 7.4) e avaliou-se a Precisão que o algoritmo CAH+MDL apresentava para diversas combinações de número máximo de repetições de um termo por documento, penalidade para termos não presentes em uma árvore e aumento de peso para termos oriundos de meta-dados e título de páginas Web. Inicialmente, buscou-se determinar o valor ideal da penalidade e do número máximo de repetições e, posteriormente, do aumento do peso. Os resultados estão resumidos nas Tabelas 7.5 e 7.6 (devido à grande quantidade de dados, omitiu-se nestas tabelas os valores de penalidade abaixo de 58, já que eles apresentavam resultados de Precisão bastante inferiores aos obtidos com os valores mostrados). A tabela 7.5 mostra a precisão da classificação da base descrita na Tabela 7.4, para a primeira categoria, com o número máximo de repetições de termo por documento igual a 1 e 3 e

cujos valores 58, 62, 65 e 70 correspondem aos valores da penalidade. A tabela 7.6 ilustra os mesmos valores, porém para o número máximo de repetições de termo por documento igual a 5 e 10.

	Repetições = 1				**Repetições = 3**			
Fold	**58**	**62**	**65**	**70**	**58**	**62**	**65**	**70**
1	98,70%	98,70%	98,70%	98,70%	97,43%	97,75%	97,75%	97,75%
2	96,14%	96,14%	96,14%	96,14%	96,78%	96,78%	96,78%	96,78%
3	96,77%	96,77%	96,77%	96,77%	97,42%	97,74%	97,74%	97,74%
4	97,10%	97,10%	97,10%	97,10%	96,77%	96,77%	96,77%	96,45%
5	95,78%	95,78%	96,10%	96,10%	95,78%	96,10%	96,10%	96,10%
6	95,75%	96,08%	96,41%	96,41%	96,73%	96,73%	96,73%	96,73%
7	96,05%	96,38%	96,38%	96,38%	96,05%	96,38%	96,38%	96,71%
8	98,35%	98,35%	98,02%	98,02%	98,02%	98,35%	98,35%	98,35%
9	98,01%	98,01%	98,01%	98,01%	98,34%	98,34%	98,68%	98,68%
10	98,35%	98,35%	98,35%	98,68%	97,02%	97,02%	97,35%	97,35%
Média	96,94%	97,00%	97,04%	97,07%	97,03%	97,20%	97,26%	97,26%

Tabela 7.5: Precisão da classificação com o número máximo de repetições igual a 1 e 3

	Repetições = 5				**Repetições = 10**			
Fold	**58**	**62**	**65**	**70**	**58**	**62**	**65**	**70**
1	97,43%	97,75%	98,07%	98,07%	97,43%	97,43%	98,07%	98,07%
2	96,78%	96,78%	96,78%	96,78%	97,43%	97,43%	97,43%	97,43%
3	97,10%	97,42%	97,42%	97,42%	96,77%	97,10%	97,10%	97,10%
4	97,10%	97,10%	97,10%	97,10%	96,45%	96,77%	96,77%	96,77%
5	96,10%	96,43%	96,43%	96,43%	95,78%	95,78%	96,10%	96,10%
6	96,73%	96,73%	97,06%	98,06%	97,06%	97,39%	97,39%	96,73%
7	96,71%	96,71%	96,38%	96,38%	96,38%	96,38%	96,38%	96,38%
8	98,02%	98,02%	98,35%	98,35%	98,02%	98,02%	98,02%	98,35%
9	98,34%	98,34%	98,34%	98,34%	98,34%	98,34%	98,34%	98,68%
10	97,02%	97,35%	97,35%	97,35%	97,35%	97,35%	97,35%	97,02%
Média	97,13%	97,26%	97,33%	97,33%	97,10%	97,20%	97,30%	97,26%

Tabela 7.6: Precisão da classificação com o número máximo de repetições igual a 5 e 10

Testes subsequentes com valores maiores para o número máximo de repetição e de penalidade apresentaram resultados inferiores e não estão mostrados aqui.

Com base nos resultados obtidos, escolheu-se o valor de 65 para a penalidade atribuída a termos não presentes em uma árvore e 10 como valor inicial para o número máximo de repetições de termos por documento. Realizou-se então novos experimentos com estes valores, porém, com diferentes pesos para termos provenientes dos meta-dados e do título da página (um peso n para determinados meta-dados, neste caso, corresponderia a multiplicar a contagem de ocorrência dos termos provenientes destes meta-dados por n), com o objetivo de avaliar o quanto isso contribuiria para a melhoria da Precisão do algoritmo.

A razão básica para isso é que, apesar dos meta-dados e título em geral possuírem termos que descrevem melhor o conteúdo de uma página do que os termos nela contidos, a sua contagem é bastante menor que a contagem dos termos gerais, fazendo com que eles acabem recebendo códigos com maiores números de bits. O aumento do peso visa acabar com esta diferença.

Inicialmente o aumento do peso para cada meta-dado e para o título da página foi testado individualmente para verificar o quanto ele contribuía positiva ou negativamente para a melhoria na Precisão de classificação. Com base nos resultados obtidos, foram testadas também combinações entre os termos que produziram os melhores resultados individuais, para avaliar também sua contribuição conjunta. O número máximo de repetições, inicialmente determinado como 10, foi também progressivamente sendo aumentado à medida em que o aumento do peso dos meta-dados trazia melhores resultados na classificação dos documentos, até se determinar o valor ideal como sendo 18, para os seguintes pesos dos meta-dados: *Description* = 5, *Classification* = 3, *Keywords* = 5 e Título = 5. Os resultados obtidos, todos utilizando o valor de penalidade = 65, estão resumidos nas Tabelas 7.7, 7.8, 7.9 e 7.10. Nestas tabelas, os valores dos pesos são mostrados da forma Di Cj Kk Tl, onde as iniciais D, C, K e T correspondem aos meta-dados *Description*, *Classification*, *Keywords* e ao Título da página, respectivamente. Os valores i, j, k e l correspondem aos valores atribuídos aos pesos dos respectivos meta-dados

	Repetições = 5			
Fold	**D1 C5 K1 T1**	**D5 C1 K1 T1**	**D1 C1 K5 T1**	**D1 C1 K1 T5**
1	97,43%	97,43%	97,43%	97,75%
2	96,46%	96,46%	98,07%	97,75%
3	96,13%	96,45%	96,45%	96,45%
4	97,42%	97,74%	97,74%	98,39%
5	98,38%	98,38%	98,38%	97,73%
6	97,06%	97,06%	97,06%	97,06%
7	98,03%	98,36%	98,03%	96,36%
8	98,02%	98,68%	97,69%	98,68%
9	97,02%	97,35%	96,69%	97,68%
10	97,02%	97,35%	97,68%	97,35%
Média	97,30%	97,53%	97,52%	97,72%

Tabela 7.7: Precisão da classificação com máximo de 5 repetições e variação de pesos

7.2.2 Análise dos Resultados Obtidos

Os melhores resultados foram obtidos com a penalidade para termos não presentes nas árvores igual a 65, o número máximo de repetições de um mesmo termo por documento igual a 18 e os seguintes pesos dos meta-dados: *Description* = 5, *Classification* = 3, *Keywords* = 5 e Título = 5. O meta-dado que mais contribuiu individualmente para a melhora na Precisão da classificação foi o título da página, seguido pela Descrição (*Meta-Description*). As palavras-chave (*Meta-Keywords*) apresentaram resultados intermediários, enquanto que a Classificação (*Meta-Classification*) praticamente não influenciou no resultado.

	Repetições = 10			
Fold	**D5 C1 K1 T1**	**D1 C1 K1 T5**	**D1 C1 K1 T10**	**D5 C1 K1 T5**
1	97,75%	97,43%	97,43%	97,75%
2	96,78%	98,07%	98,07%	98,39%
3	96,45%	96,13%	96,45%	96,77%
4	98,06%	98,39%	97,74%	98,71%
5	98,38%	98,05%	98,73%	98,05%
6	97,39%	97,39%	97,39%	98,04%
7	98,03%	98,36%	98,36%	98,03%
8	98,68%	98,68%	97,03%	99,01%
9	97,35%	98,01%	98,34%	97,68%
10	97,35%	97,02%	97,02%	97,68%
Média	97,62%	97,75%	97,56%	98,01%

Tabela 7.8: Precisão da classificação com máximo de 10 repetições e variação de pesos

	Repetições = 15			
Fold	**D5 C1 K1 T5**	**D5 C5 K5 T5**	**D5 C3 K3 T5**	**D5 C1 K3 T5**
1	98,07%	98,07%	97,75%	97,75%
2	98,71%	98,71%	98,71%	98,71%
3	96,45%	97,10%	97,10%	97,10%
4	98,71%	98,71%	98,71%	98,71%
5	98,70%	98,70%	99,03%	99,03%
6	98,04%	97,71%	98,04%	98,04%
7	98,36%	98,03%	98,36%	98,36%
8	99,01%	99,34%	99,34%	99,34%
9	97,68%	98,01%	97,68%	97,68%
10	97,35%	98,34%	98,01%	98,01%
Média	98,11%	98,27%	98,27%	98,27%

Tabela 7.9: Precisão da classificação com máximo de 15 repetições e variação de pesos

Uma das razões para a Classificação praticamente não influenciar na Precisão é que poucas páginas que compõem a base de dados estudada possuía este meta-dado e isso é válido para páginas da Internet em geral. Uma das razões para seu baixo uso é que ele é bastante simples de ser usado para realizar o bloqueio de acesso a páginas Web, desta forma, poucas empresas estão dispostas a utilizá-lo, principalmente aquelas que possuem páginas de conteúdo impróprio ou desaconselhável para menores, bastante presentes na base estudada.

Contrariamente ao que se podia imaginar, as Palavras-Chave não apresentaram um desempenho tão bom quanto o esperado. Por definição, este meta-dado deveria ser o capaz de melhor descrever o conteúdo de uma página Web, já que deveria trazer apenas termos relevantes e que especificariam com precisão seu conteúdo. A causa deste desempenho abaixo do esperado é que frequentemente este meta-dado é explorado para aumentar o potencial de visibilidade de uma páginas nos buscadores da Internet. Sendo assim, são adicionadas palavras-chave que não necessariamente possuem relação com o conteúdo da

	Repetições = 18			
Fold	**D5 C3 K3 T5**	**D6 C3 K3 T6**	**D5 C3 K3 T5**	**D5 C3 K3 T4**
1	98,07%	98,07%	98,71%	98,07%
2	98,71%	99,36%	98,39%	98,39%
3	97,42%	97,42%	96,45%	97,42%
4	98,71%	98,71%	98,39%	98,71%
5	99,03%	99,03%	99,03%	98,70%
6	98,04%	98,04%	98,04%	98,04%
7	98,36%	98,36%	97,70%	98,36%
8	99,34%	99,01%	98,02%	99,34%
9	98,01%	98,34%	98,01%	98,01%
10	98,01%	97,35%	96,69%	98,01%
Média	**98,37%**	98,37%	98,27%	98,27%

Tabela 7.10: Precisão da classificação com máximo de 18 repetições e variação de pesos

página, tendo como único objetivo aumentar seu número de visitantes.

Por fim, o aumento da Precisão do classificador CAH+MDL obtido com o aumento do peso atribuído aos meta-dados ficou em pouco mais de 1 ponto percentual, que pode ser considerado baixo. Entretanto, o fato dos termos provenientes dos meta-dados já serem marcados como tal, por si só já traz um grande ganho de acurácia, como mostrado no estudo feito por Sun et al. [SLN02], que apresentou ganhos na classificação de páginas Web por *Support Vector Machines* de até 7 pontos percentuais, simplesmente por marcar diferentemente as palavras provenientes dos títulos das páginas Web. Além disso, a Precisão do classificador CAH+MDL para a base estudada já era bastante elevada, o que faz com que aumentos substanciais sejam mais difíceis.

7.3 Algoritmo CAH+MDL sob Hipótese de Mundo Fechado

Este experimento buscou avaliar se o algoritmo CAH+MDL sob a hipótese de mundo fechado, como descrito na Seção 5.3.1, apresentaria resultados de classificação de páginas Web multi-label suficientemente bons para que pudesse ser utilizado em situações reais. Para realizar esta avaliação, seus resultados foram comparados com os de uma *Support Vector Machine* linear na classificação dos mesmos dados.

A descrição completa do experimento é mostrada na Seção 7.3.1 e os resultados na Seção 7.3.2.

7.3.1 Descrição do Experimento

Para a realização deste experimento, não foi possível a utilização da base WebKb, devido ao fato de que ela não possui documentos multi-label, apenas multiclasse. Devido a este fato e a inexistência na data da realização desta pesquisa, até onde pôde ser verificado, de bases de dados com páginas Web pré-classificadas em categorias multi-label em inglês e/ou português, optou-se por utilizar uma base de páginas Web composta de sete categorias

multi-label e manualmente classificada pelo autor. Esta base foi a mesma utilizada no experimento de avaliação do peso de meta-dados (Seção 7.2.1) e está descrita na Tabela 7.4.

Novamente, para evitar que ocorresse *overfitting* nos classificadores, esta tabela foi separada de forma randômica em 10 pares de arquivos mutuamente exclusivos, de forma a permitir a realização da validação cruzada de 10 partes (*10-Fold Cross-Validation*). Cada um destes 10 pares de arquivos consistia de um arquivo de treinamento, com cerca de 90% dos documentos, e um arquivo de teste, com os 10% restantes. Ao separar a base de dados, todos os arquivos de treinamento e teste mantiveram tanto quanto possível a mesma proporção de categorias da base completa.

Como o classificador SVM (que serviu como base comparativa para a avaliação da performance do classificador CAH+MDL) não consegue tratar diretamente o problema de classificação multi-label, ele foi treinado utilizando uma base de dados transformada: todas as vezes que havia um documento que pertencia a duas categorias, as duas eram combinadas para formar uma nova. Na base de dados utilizada, isso significa que o classificador SVM foi treinado para reconhecer 12 categorias distintas: as 7 categorias originais mais 5 criadas pelas combinações de (2,3), (3,5), (4,5), (5,6) e (5,7).

O classificador SVM utilizado foi implementado através do uso da LIBSVM [CL01], com a criação de um conjunto de classificadores binários treinados por meio da abordagem um-contra-um, descrita na Seção 2.2.2. Esta abordagem de treinamento foi considerada por Hsu at al. [wHjL02] como a que produz os melhores resultados de classificação, quando comparada com a um-contra-todos.

Para calcular os valores de Precisão, Recall e F_1 para o classificador SVM, as novas categorias eram decompostas novamente nas originais após cada documento ter sido classificado (isso significa, por exemplo, que se o classificador SVM reportava um determinado documento como pertencente à categoria 8, ele era considerado como pertencente às categorias 2 e 3).

Após a criação dos arquivos de treinamento, tanto o classificador SVM quando o CAH+MDL foram ajustados para atingir sua melhor performance: no caso do SVM, isso significa encontrar o melhor valor para o parâmetro C e o número máximo de repetições de um termo por documento que produzia os melhores resultados de classificação. Foram obtidos então os valores ideais como sendo $C = 0.1$ e Número Máximo de Repetições = 3.

Para o classificador CAH+MDL, os parâmetros ajustados foram a penalidade por inexistência de um termo em uma árvore e o número máximo de repetições de um termo por documento. Para este caso, foram utilizados os mesmos valores obtidos no experimento de avaliação do peso de meta-dados (Seção 7.2.1): Número Máximo de Repetições = 18 e Penalidade = 65.

Após terem sido ajustados para atingir seu melhor resultado, como descrito na Seção 7.3.1, tanto o classificador SVM quanto o CAH+MDL foram treinados com o mesmo arquivo de treinamento e usados para classificar o arquivo de teste correspondente. Este procedimento foi então repetido 10 vezes, uma vez para cada arquivo de treinamento distinto.

7.3.2 Resultados Obtidos

Após a repetição dos procedimentos de treinamento e teste para cada um dos 10 pares de arquivos, os resultados obtidos pelos classificadores foram anotados e estão listados na Tabela 7.11.

	SVM			**CAH+MDL**		
Fold	**Precisão**	**Recall**	$\boldsymbol{F_1}$	**Precisão**	**Recall**	$\boldsymbol{F_1}$
1	97,91%	97,27%	97,59%	94,69%	97,75%	96,10%
2	96,78%	95,98%	96,38%	96,78%	97,75%	97,26%
3	97,58%	97,10%	97,34%	93,23%	96,29%	94,74%
4	98,23%	97,10%	97,66%	94,84%	97,42%	96,11%
5	96,10%	95,62%	95,86%	96,43%	97,89%	97,15%
6	98,53%	98,37%	98,45%	95,75%	98,53%	97,12%
7	95,89%	94,74%	95,31%	96,22%	97,04%	96,63%
8	97,19%	96,70%	96,94%	94,55%	99,01%	96,73%
9	97,35%	96,52%	96,93%	95,53%	96,52%	96,02%
10	96,69%	96,69%	96,69%	95,53%	96,69%	96,11%
Média	97,22%	96,61%	96,92%	95,36%	97,49%	96,41%

Tabela 7.11: Resultados da classificação pelos classificadores SVM e CAH+MDL

Os resultados mostram que o classificador SVM apresentou maior Precisão, enquanto que o classificador CAH+MDL obteve maior Recall. A métrica F_1 aponta uma vantagem para o SVM por uma pequena margem, cerca de 0,5 pontos percentuais na média dos 10 arquivos.

O problema com o classificador SVM é que, apesar dele ter funcionado bastante bem para esta base de dados, o número de classificadores binários necessários para a abordagem de treinamento um-contra-um cresce de forma quadrática em relação ao número de categorias e, como cada nova combinação de categorias é transformada numa nova, isso causa um crescimento exponencial, fazendo com que esta abordagem se torne impraticável quando o número de categorias ou o número de suas combinações seja muito elevado. O algoritmo CAH+MDL, por sua vez, possui tempo de treinamento linear em relação ao número de categorias, independentemente de sua combinação, permitindo que ele trate bem problemas deste tipo.

Outra vantagem do classificador CAH+MDL, quando comparado ao classificador SVM, é que ele permite treinamento on-line e incremental, sendo assim, pode ser aplicado em diversos problemas onde a base de dados de treinamento muda com frequencia. Além disso, o algoritmo CAH+MDL pode ser usado mesmo com bases de dados multi-label bastante esparsas, i.e., bases de dados onde algumas combinações de categorias ocorrem muito infrequentemente, tornando impossível transformar estas combinações em novas categorias e treinar classificadores binários para reconhecê-las, que é o que foi feito neste experimento.

7.4 Algoritmo CAH+MDL sob Hipótese de Mundo Aberto

Este experimento objetiva identificar se o algoritmo CAH+MDL pode ser usado em situações onde não existe a prerrogativa de mundo fechado, i.e., onde um determinado documento pode não receber nenhuma categoria e ser marcado como "indefinido". A razão deste estudo é que, ainda que existam alguns poucos algoritmos que podem tratar um problema de classificação multi-label diretamente (como os citados na Seção 2.1.3), é ainda menor os que conseguem fazê-lo sem a hipótese de mundo fechado. Dos algoritmos citados, apenas o baseado em Redes Neurais, desenvolvido por Zhang [Zha06], poderia tratar um problema deste tipo diretamente.

Conforme já mencionado na Seção 5.3.2, não se pode simplesmente treinar o algoritmo para reconhecer uma categoria como sendo indefinida, já que muitos tipos de documentos "indefinidos" podem não ter nada em comum uns com os outros. A abordagem então adotada se baseia na rejeição de documentos que sejam considerados "muito diferentes" dos documentos pertencentes às categorias conhecidas.

O experimento completo com o algoritmo CAH+MDL é descrito na Seção 7.4.1 e seus resultados na Seção 7.4.2.

7.4.1 Descrição do Experimento

Para a realização deste experimento, não foi possível a utilização da base WebKb, devido ao fato de que ela não possui documentos multi-label, apenas multiclasse. Devido a este fato e a inexistência na data da realização desta pesquisa, até onde pôde ser verificado, de bases de dados com páginas Web pré-classificadas em categorias multi-label em inglês e/ou português, optou-se por utilizar uma base de páginas Web composta de sete categorias multi-label e manualmente classificada pelo autor. Esta base foi a mesma utilizada no experimento do Algoritmo CAH+MDL sob Hipótese de Mundo Fechado (Seção 7.3), com a diferença de que foram acrescentadas 215 novas Páginas Web, todas marcadas como indefinidas, i.e., não pertencentes a nenhuma das 7 categorias reconhecidas pelo classificador. Estes novos documentos foram obtidos de sites dos mais diferentes tipos: alguns apresentavam alguma semelhança com documentos pertencentes a categorias conhecidas enquanto outros eram bastante diferentes. A base completa está descrita na Tabela 7.12.

Mais uma vez, para evitar que ocorresse *overfitting* nos classificadores, esta tabela foi separada de forma randômica em 10 pares de arquivos mutuamente exclusivos, de forma a permitir a realização da validação cruzada de 10 partes (*10-Fold Cross-Validation*). Entretanto, antes de realizar a separação dos arquivos, os documentos marcados como não pertencentes a nenhuma categoria foram removidos e separados em uma base de dados exclusiva. Os demais documentos foram separados então em 10 pares de arquivos, que consistiam cada um em um arquivo de treinamento, com cerca de 90% dos documentos, e um arquivo de teste, com os 10% restantes. Ao separar a base de dados, todos os arquivos de treinamento e teste mantiveram tanto quanto possível a mesma proporção de categorias da base completa.

Após a criação dos arquivos de treinamento, o classificador CAH+MDL foi configurado com os mesmos parâmetros obtidos no experimento de avaliação do peso de meta-dados (Seção 7.2.1): Número Máximo de Repetições = 18 e Penalidade = 65. O classificador

Categorias	Descrição	Número de Documentos
0	Nenhuma Categoria	215
1	Material Adulto	502
2	Jogos de Azar	425
2, 3	Jogos de Azar / Esportes	76
3	Esportes	340
3, 5	Esportes / Loja Virtual	86
4	Jogos	418
4, 5	Jogos / Loja Virtual	84
5	Loja Virtual	130
5, 6	Loja Virtual / Música	94
5, 7	Loja Virtual / Sexo Explícito	120
6	Música	407
7	Sexo Explícito	385
Total		**3282**

Tabela 7.12: Base de páginas Web incluindo documentos marcados como "indefinidos"

foi em seguida treinado com cada um dos 10 arquivos de treinamento e, para cada um deles, usado para classificar separadamente o arquivo de teste correspondente e o arquivo com os sites indefinidos. Foram feitas duas passagens para cada um dos treinamentos, usando-se intervalos de confiança diferentes.

7.4.2 Resultados Obtidos

Os resultados obtidos após os 10 procedimentos de treinamento e teste, para os dois intervalos de confiança testados, estão resumidos nas Tabelas 7.13 e 7.14. Em ambas as tabelas, a coluna identificada como Precisão, sob o rótulo Indefinidos, indica o percentual de documentos indefinidos que foram corretamente identificados como tal. Para estes documentos, não existe como calcular o Recall, já que eles não possuem nenhuma categoria atribuída. É importante observar que a base de documentos indefinidos é sempre a mesma, para todos os 10 classificadores, já que eles não são treinados com documentos nela presentes.

O intervalo de confiança mostrado nas tabelas 7.13 e 7.14 corresponde a um novo parâmetro do algoritmo CAH+MDL que pode ser ajustado para a classificação de documentos sob a hipótese de mundo aberto: quando maior o intervalo de confiança escolhido, maior o Recall das categorias atribuídas a documentos conhecidos, porém menor é o percentual de documentos indefinidos corretamente identificados. Um intervalo de confiança de 100% corresponderia ao classificador CAH+MDL operando exatamente da mesma forma que sob a hipótese de mundo fechado. Neste caso, pelo menos uma categoria seria atribuída a todos os documentos indefinidos.

Novamente, o intervalo ideal vai depender da aplicação que o esteja utilizado: para aplicações onde busca-se selecionar ou classificar apenas uma amostra dos documentos com grande grau de acurácia (o que ocorre tipicamente em mineração de dados), pode-se utilizar um intervalo de confiança pequeno, fazendo com que o algoritmo tenha uma alta Precisão tanto na classificação das categorias conhecidas quanto na identificação de

	Grau de Significância = 77%			
	Classificados			Indefinidos
Fold	**Precisão**	**Recall**	F_1	**Precisão**
1	95,82%	70,10%	80,97%	94,88%
2	98,55%	76,05%	85,85%	94,88%
3	96,29%	78,55%	86,52%	94,88%
4	96,94%	76,13%	85,28%	94,42%
5	98,05%	81,98%	89,30%	92,56%
6	96,73%	76,47%	85,42%	94,88%
7	98,52%	69,24%	81,32%	94,42%
8	96,86%	78,71%	86,85%	93,02%
9	97,68%	78,15%	86,83%	95,35%
10	97,85%	76,99%	86,18%	94,42%
Média	97,33%	76,24%	85,45%	94,37%

Tabela 7.13: Resultados da classificação da base com intervalo de confiança de 77%

	Grau de Significância = 75%			
	Classificados			Indefinidos
Fold	**Precisão**	**Recall**	F_1	**Precisão**
1	96,14%	67,85%	79,55%	95,35%
2	98,87%	73,79%	84,51%	95,81%
3	96,29%	76,29%	85,13%	95,81%
4	97,10%	74,52%	84,32%	94,88%
5	98,21%	80,03%	88,19%	94,42%
6	96,73%	76,14%	85,21%	95,35%
7	98,52%	67,93%	80,41%	94,88%
8	96,86%	76,07%	85,22%	94,88%
9	97,68%	76,16%	85,59%	96,28%
10	97,85%	75,00%	84,91%	94,88%
Média	97,43%	74,38%	84,31%	95,25%

Tabela 7.14: Resultados da classificação da base com intervalo de confiança de 75%

documentos indefinidos, porém apresentando um baixo Recall. Por outro lado, para aplicações onde se deseja apenas eliminar documentos que sejam substancialmente diferentes dos conhecidos, pode-se usar um intervalo de confiança maior, o que fará com que o Recall do classificador seja alto, porém baixando a Precisão na identificação de documentos indefinidos.

É interessante notar que os valores da Precisão mostrados nas Tabelas 7.13 e 7.14 são maiores, ainda que não de forma significativa, do que os correspondentes da Tabela 7.11. A razão disso, é que alguns documentos que na hipótese de mundo fechado recebiam uma ou mais categorias incorretas, passaram a ser marcados como indefinidos, aumentando com isso a precisão do classificador.

Independentemente do intervalo de confiança adotado, os números mostrados nas Tabelas 7.13 e 7.14, comprovam que o classificador CAH+MDL pode perfeitamente ser

utilizado para aplicações onde determinados documentos não devem ser rotulados com nenhuma categoria, desde que não se exija que ele apresente um Recall muito elevado, ou ainda para o caso onde os documentos a serem rejeitados, i.e. não rotulados, sejam significativamente diferentes dos pertencentes a categorias conhecidas.

8. Conclusão e Trabalhos Futuros

Este capítulo apresenta a conclusão desta pesquisa, mostrada na Seção 8.1, e apresenta possíveis futuros trabalhos e pesquisas que podem ser desenvolvidos a partir dos resultados aqui obtidos, que estão descritos na Seção 8.2.

8.1 Conclusão

Nesta pesquisa, foi feita a extensão do algoritmo CAH+MDL, que originalmente só conseguia lidar com problemas de classificação textuais binários ou multiclasse, para que ele conseguisse tratar também, diretamente, problemas de classificação multi-label, com ou sem a hipótese de mundo fechado (que obriga que todos os documentos a serem classificados recebam ao menos uma categoria).

Sob a hipótese de mundo-fechado, sua acurácia média, medida pela métrica F_1, foi praticamente tão boa quanto a apresentada por um conjunto de classificadores baseados em SVM lineares, conforme mostrado na Seção 7.3.1. Além disso, apesar do algoritmo ter produzido valores inferiores para a Precisão, ele obteve um melhor Recall, tornando-o especialmente adequado para aplicações onde esta última métrica é mais importante. Aplicações em mineração de dados, portanto, se beneficiariam muito do algoritmo.

O algoritmo CAH+MDL, como demonstrado na Seção 7.4, pode também ser utilizado em aplicações que demandam a hipótese de mundo aberto, que é o caso típico dos problemas de classificação de páginas Web, como o que foi tratado nesta pesquisa. Neste tipo de problema, é fundamental que determinados documentos não recebam nenhuma categoria (novamente, considerando o problema estudado de classificação páginas Web, é praticamente impossível se considerar que se consiga criar um conjunto de categorias amplo o suficiente para que possa englobar a totalidade das páginas atualmente disponíveis na Internet, bem como as novas que surgem a cada dia). Nestas condições, o algoritmo CAH+MDL se comporta suficientemente bem, desde que não se exija que ele possua um alto Recall, ou ainda em condições onde os documentos a serem rejeitados, i.e., não rotulados com nenhuma categoria, sejam significativamente diferente dos que pertençam a categorias conhecidas, o que fará com que seu Recall seja maior do que o obtido no experimento da Seção 7.4.

As vantagens do algoritmo CAH+MDL quando comparado com um conjunto de classificadores, não necessariamente formados apenas por SVM, como estudado nesta pesquisa, mas por outros algoritmos de classificação em geral, é que ele permite treinamento on-line e incremental e sua complexidade cresce linearmente de acordo com o número de categorias conhecidas, independentemente de suas combinações. Conjuntos de classificadores, que são formados quando o algoritmo de classificação utilizado não consegue tratar di-

retamente problemas multi-label, e fazem portanto a decomposição do problema original em diversos problemas menores, tratados por classificadores binários ou multiclasse, possuem complexidade proporcional ao número de combinações destas categorias, que pode possuir crescimento exponencial.

Além disso, o algoritmo CAH+MDL pode ser utilizado mesmo com bases de dados multi-label extremamente esparsas, i.e., onde algumas combinações de categorias ocorrem muito infrequentemente, tornando impossível sua transformação em novas categorias, como o que foi feito nesta pesquisa para sua comparação com um conjunto de classificadores SVM, e o treinamento de classificadores para reconhecê-las.

Adicionalmente ao algoritmo CAH+MDL, esta pesquisa também conseguiu verificar que o aumento do peso de termos provenientes do título e meta-dados de páginas Web consegue aumentar o percentual de Precisão e Recall na sua classificação, ainda que esta melhora não tenha sido muito expressiva (cerca de 1 ponto percentual). A razão aparente para esta melhora não ter sido tão significativa foi o fato da base de comparação ter sido feita já com a separação dos termos oriundos do título e meta-dados das páginas, fato que por si só já garante um ganho significativo de acurácia, conforme já comentado na Seção 7.2.2.

8.2 Trabalhos Futuros

Como potenciais trabalhos futuros a serem realizados a partir dos resultados desta pesquisa, pode-se citar os seguintes:

- Extensão do algoritmo CAH+MDL para trabalhar com a classificação de outros tipos de documentos, já que a versão estudada e ampliada consegue classificar apenas documentos textuais e outros tipos derivados destes (páginas Web, entre outros). Poderia se verificar se é possível que o algoritmo consiga classificar documentos completamente diversos, como imagens, por exemplo, de forma a torná-lo um algoritmo mais geral.
- Estudo da geração das distribuições de probabilidade a partir de um subconjunto dos dados de treinamento e não de sua totalidade, como foi estudado aqui. O objetivo deste estudo seria fazer com que a etapa de treinamento ficasse mais rápida, verificando também se com o uso deste subconjunto de dados a acurácia do algoritmo seria mantida.
- Na hipótese de mundo aberto, análise de outras possibilidades para a rejeição de documentos que não apenas diretamente a partir do intervalo de confiança. Poderia se utilizar um subconjunto dos termos presentes em cada documento (usando somente os melhores, piores ou os médios, por exemplo) ou o número de termos não presentes nas árvores, de modo a ampliar o percentual de documentos corretamente identificados como indefinidos e/ou aumentar o Recall para os documentos pertencentes a categorias conhecidas.
- Estudo do treinamento feito em tempo real do algoritmo CAH+MDL estendido, de modo a verificar se as distribuições de probabilidades inicialmente calculadas, obtidas a partir do conjunto total de dados, podem ser utilizadas mesmo se o número

de documentos adicionais a serem treinados seja grande e, caso não seja possível, a partir de que número de documentos adicionais deve-se regerar completamente as distribuições de probabilidade, de modo a manter alta a acurácia do classificador.

- Uso de elementos de Web Semântica, e.g., ontologias, junto ao classificador CAH+MDL estendido de modo a analisar seu potencial ganho de performance com o uso destes elementos.

Apêndice A. Documentação do Classificador

Este capítulo visa documentar o classificador CAH+MDL desenvolvido como parte desta pesquisa, tanto em relação ao usuário final quanto em relação ao desenvolvedor do sistema. A Seção A.1 documenta os comandos necessários para se usar o classificador, a partir de um prompt de comando e a sequência de comandos usada nesta pesquisa. A Seção A.2 documenta a estrutura dos arquivos gerados com o conteúdo das páginas Web. Por fim, a Seção A.3 explica os detalhes da implementação e documenta as classes criadas.

A.1 Interface com o Usuário

O classificador CAH+MDL foi todo escrito em linguagem Java. Para que ele possa ser executado é necessário, portanto, que a máquina virtual Java versão 1.5 ou superior (JRE versão 5 ou 6) esteja instalada na máquina onde ele rodará.

Para instalar o classificador, basta copiar o conteúdo da pasta "webClassifier" para onde se desejar executá-lo. Esta pasta contém o código compilado do classificador CAH+MDL bem como as bibliotecas adicionais que ele necessita para sua operação. Supondo que a pasta "webClassifier" tenha sido copiada para o diretório padrão, "C:\", o comando necessário para executá-lo seria:

```
java -jar C:\webClassifier\webClassifier.jar
```

Toda a interface com o classificador CAH+MDL foi escrita para ser acessada através de linha de comando, de forma a possibilitar que ele possa ser utilizado em *scripts*. Ao ser executado sem nenhum parâmetro, ele mostrará sua sintaxe de execução, conforme mostrado abaixo:

```
java -jar webClassifier.jar
webClassifier - Numero de parametros invalido
Sintaxe:
  webClassifier tokeniza_url <URL> [dir]
  webClassifier tokeniza_arquivo <arquivo> [dir]
  webClassifier tokeniza_novos <arquivo> [dir]
  webClassifier cria <nome_cat id_cat>... [dir]
  webClassifier treina <nome_cat> <arquivo> [dir]
  webClassifier treina_diretorio <nome_cat> <diretorio> [dir]
  webClassifier 10fold <arquivo>
```

```
  webClassifier gera <MDL | SVM> <arquivo> [dir]
  webClassifier testa <MDL | SVM> <arquivo> [dir]
  webClassifier classifica_url <URL> [dir]

  tokeniza_url     = Gera os tokens da URL informada
  tokeniza_arquivo = Gera os tokens de todas as URLs presentes no arquivo
                     informado
  tokeniza_novos   = Gera os tokens das URLs do arquivo informado que ainda não
                     tenham sido gerados (o programa verifica isso pela
                     existencia ou não do arquivo com o nome de cada URL)
  cria             = Cria um novo classificador com as categorias informadas.
                     (E' necessario se especificar pelo menos 2 categorias)
  treina           = Treina o classificador em uma categoria com o conteudo do
                     arquivo informado
  treina_diretorio = Treina o classificador em uma categoria com o conteudo de
                     todos os arquivos do diretorio informado
  10fold           = Separa o arquivo VSM informado em 10 pares de arquivos, que

                     podem ser usados para fazer 10 fold cross-validation
  gera             = Gera a base de classificação do classificador. Esta opcao
                     deve ser invocada apos finalizados os treinamentos
                     necessario apenas para o classificador MDL)
  testa            = Testa classificador com conteudo do arquivo VSM informado
  classifica_url   = Classifica uma URL em uma das categorias do classificador
Parametros
  dir = Nome do diretorio onde serao gerados os arquivos com os tokens
```

A primeira coisa que deve ser feita ao se começar a usar o classificador é "tokenizar", i.e., transformar em termos ou *tokens*, uma base de páginas Web, que será posteriormente utilizada para treiná-lo, quando ele então poderá ser usado para classificar novas páginas. Para realizar a "tokenização" de páginas, existem três comandos possíveis:

Para "tokenizar" uma única página Web:

```
java -jar webClassifier.jar tokeniza_url <url> [dir]
```

Neste caso, o parâmetro <url> deve ser substituído pela URL da página que deve ser "tokenizada" e o parâmetro opcional [dir] poderá ser informado para indicar o diretório onde o arquivo resultante será salvo. Este arquivo terá o nome igual ao da url, porém com as '/' substituídas por ',' e com a extensão ".wct".

Para "tokenizar" uma lista de páginas Web, cujos nomes deverão estar listados em um arquivo texto, onde cada nova URL deverá estar presente em uma linha diferente, usa-se o seguinte comando:

```
java -jar webClassifier.jar tokeniza_arquivo <arquivo> [dir]
```

Este comando produzirá uma série de arquivos, com o mesmo padrão de nome acima descrito, onde cada arquivo representará uma página diferente.

Opcionalmente, pode-se "tokenizar" apenas as URLs que ainda não tenham sido transformadas em *tokens*, a partir de um arquivo com sua lista, através do seguinte comando:

```
java -jar webClassifier.jar tokeniza_novos <arquivo> [dir]
```

Este comando lerá as URLs do arquivo informado, uma por linha, e verificará se o arquivo correspondente já se encontra criado. Caso não esteja, a página será lida e seus *tokens* salvos. Caso o arquivo já exista, nada será feito e a próxima URL será verificada.

Após todas as páginas que serão usadas como treinamento tenham sido convertidas em *tokens*, é necessário criar a definição das categorias que posteriormente serão utilizadas para se treinar e gerar classificadores CAH+MDL e SVM. Para se criar esta definição de categorias, usa-se o comando abaixo:

```
java -jar webClassifier cria <nome_cat id_cat>... [dir]
```

Para cada categoria, deve-se especificar seu nome através do parâmetro <nome_cat> e um inteiro positivo que será usado como sua identificação em <id_cat>. Esta identificação será usada no momento da geração do arquivo VSM que será lido pelos classificadores quando estes forem ser treinados. Podem ser criadas quantas categorias se desejar, desde que cada uma tenha um nome e um identificador distintos. O parâmetro opcional [dir] poderá ser informado para indicar o diretório onde a definição das categorias será salva.

Depois de definidas as categorias, deve-se inserir o conteúdo dos arquivos das páginas Web "tokenizadas" no arquivo VSM, que terá conteúdo totalmente numérico e será então lido pelos classificadores CAH+MDL e SVM e utilizado como treinamento. Para fazer isso, deve-se usar um dos comandos abaixo:

```
java -jar webClassifier treina <nome_cat> <arquivo> [dir]
java -jar webClassifier treina_diretorio <nome_cat> <diretorio> [dir]
```

O primeiro comando insere apenas um arquivo com os *tokens* de uma página Web no arquivo VSM com os dados de treinamento, enquanto que o segundo insere todos os arquivos presentes no diretório especificado. Em ambos os casos, deve-se especificar a categoria à qual o arquivo ou diretório pertencem em <nome_cat>. Caso o arquivo ou diretório com arquivos pertençam a mais de uma categoria, deve-se repetir o comando correspondente para cada categoria distinta.

Para o primeiro comando, especifica-se o arquivo a ser inserido no parâmetro <arquivo> e para o segundo, especifica-se o diretório de onde os arquivos a serem inseridos serão lidos em <diretorio>. O parâmetro opcional [dir] poderá ser informado para indicar o diretório onde a definição das categorias está salva, caso não seja o diretório corrente.

Caso se deseje realizar a validação cruzada (*10-Fold Cross-Validation*) dos classificadores, pode-se separar o arquivo VSM com todo o conteúdo do treinamento em 10 pares de arquivos, onde cada par consistirá de um arquivo de treinamento e outro para testes. Para isso, basta utilizar o comando abaixo:

```
java -jar webClassifier 10fold <arquivo>
```

O parâmetro <arquivo> deve ser utilizado para se especificar o caminho completo do arquivo VSM que será separado em 10. Os nomes dos arquivos resultantes serão os mesmos do arquivo original, porém com os sufixos "_train_n" ou "_test_n", que correspondem aos arquivos de treinamento e teste, respectivamente, e onde n será um número de 0 a 9.

Após todas as páginas que serão usadas como treinamento tenham sido inseridas no arquivo VSM, e este opcionalmente separado para a realização de treinamento cruzado, deve-se gerar o(s) classificador(es) CAH+MDL e/ou SVM. Para isso, usa-se o comando a seguir:

```
java -jar webClassifier gera <MDL | SVM> <arquivo> [dir]
```

É possível se especificar o tipo de classificador a ser criado especificando-se MDL, que criará um classificador CAH+MDL ou SVM, que criará um classificador baseado em *Support Vector Machines*. Em ambos os casos, em <arquivo> deve-se especificar o nome do arquivo VSM de onde os dados do treinamento serão lidos. O parâmetro opcional [dir] poderá ser informado para indicar o diretório onde o classificador será criado, caso não seja o diretório corrente.

Por fim, pode-se testar o(s) classificador(es) gerados, através do seguinte comando:

```
java -jar webClassifier testa <MDL | SVM> <arquivo> [dir]
```

Deve-se especificar o tipo de classificador a ser testado através do parâmetro MDL, que testará o classificador CAH+MDL ou SVM, que testará o classificador baseado em *Support Vector Machines*. Em ambos os casos, em <arquivo> deve-se especificar o nome do arquivo VSM de onde os dados de teste serão lidos. Este arquivo deverá possuir categoria(s) manualmente atribuída(s) a cada página Web, da mesma que no arquivo de treinamento, porém as categorias presentes serão ignoradas no momento da classificação, sendo utilizadas apenas para a verificação do percentual de acertos do classificador. O parâmetro opcional [dir] poderá ser informado para indicar o diretório onde o classificador foi criado, caso não seja o diretório corrente.

A.1.1 Sequência de Comandos Usados Nesta Pesquisa

Nesta seção será mostrada a sequência completa de comandos que foram utilizados para gerar e testar os classificadores CAH+MDL e SVM usados nesta pesquisa e que produziram os dados aqui apresentados. Esta sequência de comandos pressupõe que existem arquivos chamados de adulto.txt, azar.txt, jogos.txt, lojas.txt, musica.txt, sexo.txt, e esportes.txt, que contêm a lista de URLs correspondentes às categorias de Material Adulto, Jogos de Azar, Jogos, Lojas Virtuais, Música, Sexo e Esportes, respectivamente.

1. "Tokeniza" cada um dos arquivos com a lista de páginas Web

```
java -jar webClassifier tokeniza_novos adulto.txt adulto
java -jar webClassifier tokeniza_novos azar.txt azar
java -jar webClassifier tokeniza_novos jogos.txt jogos
java -jar webClassifier tokeniza_novos lojas.txt lojas
java -jar webClassifier tokeniza_novos musica.txt musica
java -jar webClassifier tokeniza_novos sexo.txt sexo
java -jar webClassifier tokeniza_novos esporte.txt esportes
```

2. Cria a definição das categorias

```
java -jar webClassifier cria adulto 1 azar 2 esportes 3 jogos
                         4 lojas 5 musica 6 sexo 7 classificador
```

3. Gera arquivo VSM com o conteúdo de todas as páginas Web "tokenizadas".

```
java -jar webClassifier treina_diretorio adulto adulto classificador
java -jar webClassifier treina_diretorio azar azar classificador
java -jar webClassifier treina_diretorio esportes esportes classificador
java -jar webClassifier treina_diretorio jogos jogos classificador
java -jar webClassifier treina_diretorio lojas lojas classificador
java -jar webClassifier treina_diretorio musica musica classificador
java -jar webClassifier treina_diretorio sexo sexo classificador
```

4. Separa o arquivo VSM em 10 pares de arquivos para treinamento cruzado

```
java -jar webClassifier 10fold classifier.vsm
```

5. Gera e testa cada um classificadores a partir dos arquivos VSM (variando os nomes dos arquivos classifier_train_0 e classifier_train_0 de 0 a 9).

```
java -jar webClassifier gera mdl classifier_train_0.vsm classificador
java -jar webClassifier testa mdl classifier_train_0.vsm classificador

java -jar webClassifier gera svm classifier_train_0.vsm classificador
java -jar webClassifier testa svm classifier_test_0.vsm classificador
```

A.2 Sintaxe dos arquivos

Esta seção apresenta a estrutura dos arquivos com o conteúdo das páginas Web processadas, da forma que são gerados e utilizados pelo classificador implementado. A Seção A.2.1 descreve os arquivos que representam páginas Web enquanto a Seção A.2.2 mostra a estrutura dos arquivos finais, em formato VSM, que são utilizados para treinar e testar o classificador.

A.2.1 Arquivos de Páginas Web

Cada página Web é gravada em um arquivo separado, cujo nome é igual a URL completa, sem os parâmetros de scripts que possam existir, com a substituição de todos os separadores de diretório "/", por "," e com a inclusão da extensão ".wct". O nome do arquivo gerado para a URL "`http://home22.inet.tele.dk/hightower`", por exemplo, seria home22.inet.tele.dk,hightower.wct.

Cada arquivo de Páginas Web consiste de uma coleção de palavras, seguidas pelo separador ":", seguidas pelo seu número de ocorrências, sem quebra de linha. As diferentes palavras são separadas por um espaço. Por exemplo, se a palavra "imagem" aparecesse cinco vezes em uma página, ela seria inserida no arquivo como "imagem:5".

Palavras provenientes do título e dos metadados das páginas, são precedidas dos seguintes identificadores:

C* Indica que a palavra veio de *Meta-Classification*.

D* Indica que a palavra veio de *Meta-Description*.

K* Indica que a palavra veio de *Meta-Keywords*.

T* Indica que a palavra veio do título da página

Desta forma, seguindo o exemplo anterior, se a palavra "imagem" aparecesse uma vez no título de uma página, ela seria inserida no arquivo como "T*imagem:1".

Um exemplo de um arquivo completo representando uma página Web é mostrado a seguir:

```
D*allmusic:1 occasion:1 shades:1 interview:1 decent:1 new:1 enthusiast:1
K*biography:1 D*artists:1 releases:1 know:1 lewis:2 album:2 west:2 art:2
make:1 hair:1 blog:1 bloc:1 classical:4 release:1 dynamics:1 brut:2
K*releases:1 world:1 editors:1 recorder:2 east:2 article:1 michala:2 1st:1
amga:1 glissandi:1 ripping:1 improvisations:1 artist:2 searches:1
D*music:1 tune:1 D*releases:1 performer:1 D*reviews:1 composers:1
possible:1 phenomenon:1 bamboo:1 instrument:1 average:1 long:1 D*new:1
human:1 spoke:1 videos:1 end:1 K*artists:1 uncle:2 D*biography:1 K*music:1
ross:1 chen:2 pull:1 spotlight:1 meets:1 work:1 register:1 dialogue:2
login:1 stand:1 note:1 choice:1 petri:3 voice:1 read:1 K*reviews:1
corner:1 K*new:1 group:1 satan:1 flutist:1 music:3 allmusic:1
T*allmusic:1 reviews:1 logged:1 song:1 hendricks:1 happy:1 think:1
K*allmusic:1 dave:2 lambert:1 astounding:1 writers:1 quite:1 yue:2
established:1
```

A.2.2 Arquivos VSM

Os arquivos VSM, com conteúdo exclusivamente numérico, são utilizados para o treinamento e avaliação do classificador. Eles são gerados a partir dos arquivos de representação

das páginas Web, descritos na Seção A.2.1, e possuem formato compatível com o especificado pela LIBSVM [CL01].

Nestes arquivos, cada linha representa uma página Web distinta, codificada em uma representação vetorial esparsa, i.e., onde apenas as dimensões com valores diferentes de zero são incluídas. As dimensões começam sempre de zero, são ordenadas em ordem crescente, são separadas por espaços e são incluídas após o(s) número(s) da(s) categoria(s), que são sempre colocados no início da linha.

A sintaxe de cada uma das linhas é a seguinte:

```
categoria[, categoria] dimensão ‘‘:’’ valor
```

Categoria - Representa o valor numérico associado a uma determinada categoria

Dimensão - Código da dimensão associada a uma determinada palavra

Valor - Número de ocorrências da palavra correspondente à dimensão.

Um exemplo da parte inicial de um arquivo VSM é mostrado a seguir:

```
1 0:1 3:5 4:1 6:2 7:2 8:12 9:3 10:9 11:2 12:5 13:1 14:3 15:2 16:1 19:5
1 5:1 7:44 8:15 38:5 66:45 67:15 93:3 136:6 150:20 181:3 186:1 202:10
2,3 6:2 41:1 43:1 61:1 67:4 101:1 126:1 153:6 156:5 160:1 162:1 173:1
3,5 36:1 79:1 427:1 561:1 566:1 596:6 599:2 612:1 619:2 636:1 648:2
3,5 41:1 596:3 658:1 877:1 1065:1 1074:1 1241:1 1491:1 2479:1 2675:1
```

Neste arquivo, as duas primeiras linhas representam uma página classificada apenas na categoria 1. A terceira linha representa uma página que foi classificada nas categorias 2 e 3, enquanto as duas últimas representam a classificação nas categorias 3 e 5.

A.3 Implementação

Esta seção descreve e documenta as classes desenvolvidas para a implementação dos classificadores CAH+MDL e *Support Vector Machine*. Inicialmente são descritas as interfaces, na Seção A.3.1, e posteriormente as classes, na Seção A.3.2. Em cada um dos casos, todos os seus métodos e atributos são apresentados e explicados.

O diagrama de classes, mostrado na figura A.1 apresenta uma visão geral da hierarquia das interfaces e classes implementadas e de seus usos. Ele foi montado de acordo com a sintaxe da linguagem UML (*Unified Modeling Language*):

- Círculos correspondem às interfaces
- Retângulos correspondem às classes.
- Linhas contínuas sem setas representam uma derivação, i.e., uma hierarquia de classes (que para o classificador CAH+MDL e SVM só existe a partir de uma interface para uma classe e nunca entre duas classes).
- Setas entre duas classes representam o uso da classe sendo apontada pela classe apontadora.

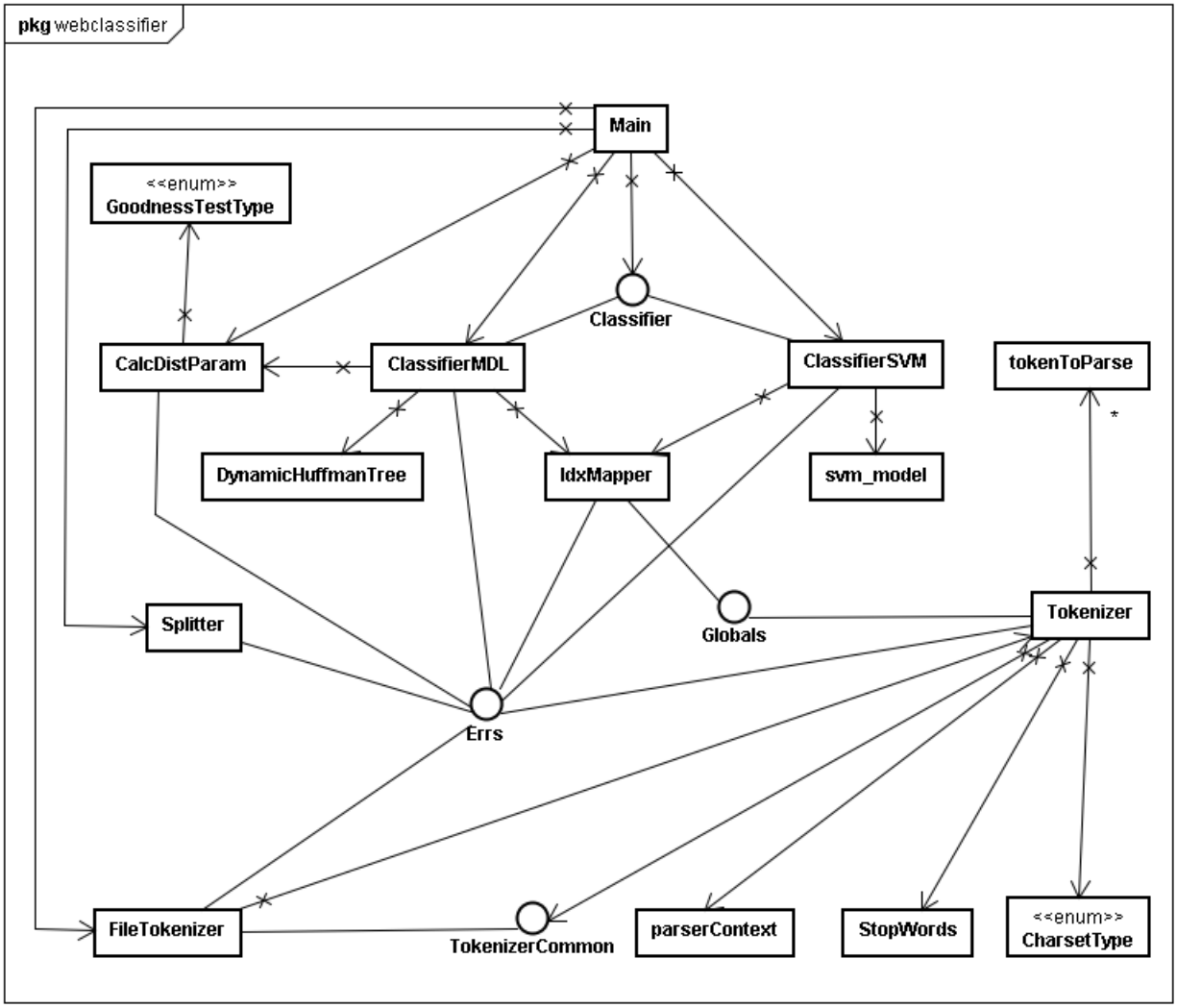


Figura A.1: Diagrama de Classes do Classificador CAH+MDL e SVM

A.3.1 Interfaces

INTERFACE **Classifier**

Esta é a interface padrão que os classificadores implementam

DECLARAÇÃO

```
public interface Classifier
```

MÉTODOS

- *createClassifier*
  `public int` **createClassifier**`( java.lang.String` **trainFile**`, java.lang.String` **dir**`, java.util.Map` **statMap** `)`
  - **Uso**
    - Este método cria um novo classificador, a partir do arquivo de treinamento informado. Ele faz todas as operações necessárias para que o classificador possa ser utilizado para a classificação de dados.
  - **Parâmetros**
    - `trainFile` - Arquivo de treinamento em formato VSM numérico (igual ao padrão utilizado pela libsvm)
    - `dir` - Diretório onde os arquivos do classificador serão criados
    - `statMap` - Mapeamento que sera preenchido na saída com as estatísticas do classificador gerado. A chave é uma string com o número de cada categoria (ou cat,cat em caso de multi-label). O valor é o número de documentos da(s) categoria(s). Se for `null`, não serão geradas estatísticas.
  - **Retorno** - Um dos valores definidos na interface Errs
  - **Veja Também**
    - `webclassifier.Classifier.openClassifier(String)`

- *openClassifier*
  `public int` **openClassifier**`( java.lang.String` **dir** `)`
  - **Uso**
    - Este método abre um classificador já criado
  - **Parâmetros**
    - `dir` - Diretório onde os arquivos do classificador foram criados
  - **Retorno** - Um dos valores definidos na interface Errs
  - **Veja Também**
    - `webclassifier.Classifier.createClassifier(String, String, Map)`

- *setMapper*
  `public int` **setMapper**`( java.lang.String` **dir** `)`
  - **Uso**
    - Este método associa um mapper com o classificador, de modo a possibilitar a classificação de dados (URLs, arquivos) não numéricos
  - **Parâmetros**
    - `dir` - Diretório onde os arquivos do mapper foram criados

---

- *testClassifier*
  `public int` **testClassifier**`( java.lang.String` **testFile**`, double []` **precisionRecall**`, java.util.Set` **badClass** `)`
  - **Uso**

    * Este método testa o classificador a partir de um arquivo com exemplos pré-classificados manualmente. As categorias presentes neste arquivo serão utilizadas apenas para gerar os índices de Precisão e Recall do classificador, não sendo utilizadas na propria classificação.
      Antes deste método ser chamado, é necessário abrir ou criar o classificador, através dos métodos openClassifier ou createClassifier.
  - **Parâmetros**
    * `testFile` - Arquivo de teste do classificador em formato VSM numérico (igual ao padrão utilizado pela libsvm)
    * `precisionRecall` - Um array de 2 doubles cujos valores serão alterados no retorno deste método (no caso de sucesso) para os valores obtidos de precision[0] e recall[1]
    * `badClass` - Conjunto que, se informado, sera preenchido com as linhas do arquivo que tiveram classificação incorreta. Se for `null` esta informação não será retornada.
  - **Retorno** - Um dos valores definidos na interface Errs
  - **Veja Também**
    * `webclassifier.Classifier.openClassifier(String)`
    * `webclassifier.Classifier.createClassifier(String, String, Map)`

Interface **Errs**

Esta interface define os códigos de erro do webClassifier

Declaração

```
public interface Errs
```

Atributos

- public static final int WCL_ERR_NO_ERROR
  - Operação OK
- public static final int WCL_ERR_CREATING_FILE
  - Erro ao criar arquivo
- public static final int WCL_ERR_OPENING_FILE
  - Erro ao abrir arquivo
- public static final int WCL_ERR_CONNECT
  - Erro ao conectar no site

- public static final int WCL_ERR_RECV
  - Erro ao receber dados
- public static final int WCL_ERR_DNS
  - Erro ao resolver nome DNS
- public static final int WCL_ERR_INVALID_URL
  - URL inválida
- public static final int WCL_ERR_TOO_MANY_REDIRECTS
  - Vários redirects encadeados
- public static final int WCL_ERR_HTTP_ERROR
  - Servidor HTTP retornou um erro
- public static final int WCL_ERR_INVALID_DATA
  - Dados inválidos recebidos do servidor HTTP
- public static final int WCL_ERR_NOT_TEXT_HTML
  - Lida uma pagina que não é HTML
- public static final int WCL_ERR_HTML_PARSER_ERROR
  - Erro ao fazer parser na página
- public static final int WCL_ERR_WRITE
  - Erro ao gravar dados no arquivo
- public static final int WCL_ERR_READ
  - Erro ao ler arquivo
- public static final int WCL_ERR_INVALID_DIR
  - Diretório inválido
- public static final int WCL_ERR_INVALID_CAT
  - Categoria inválida
- public static final int WCL_ERR_READ_ONLY
  - Arquivo aberto como somente leitura
- public static final int WCL_ERR_NOT_HTTP
  - Nao é protocolo HTTP
- public static final int WCL_ERR_INVALID_REDIRECT
  - Redirect inválido ou sem location
- public static final int WCL_ERR_INVALID_CHARSET

  - Charset inválido
- public static final int WCL_ERR_NOT_INITIALIZED
  - Objeto não foi inicializado
- public static final int WCL_ERR_INTERNAL
  - Erro interno

Interface **Globals**

Esta interface define as constantes globais do classificador

Declaração

public interface Globals

Atributos

- public static final int WCL_GLOBAL_MIN_TOKEN_LEN
  - Tamanho mínimo de um token
- public static final String WCL_GLOBAL_TOK_CLASSIFICATION
  - Marca token como vindo de Meta - Classification
- public static final String WCL_GLOBAL_TOK_DESCRIPTION
  - Marca token como vindo de Meta - Description
- public static final String WCL_GLOBAL_TOK_KEYWORDS
  - Marca token como vindo de Meta - Keywords
- public static final String WCL_GLOBAL_TOK_TITLE
  - Marca token como vindo do título da página
- public static final float WCL_GLOBAL_WEIGHT_CLASSIFICATION
  - Peso atribuído ao Meta - Classification
- public static final float WCL_GLOBAL_WEIGHT_DESCRIPTION
  - Peso atribuído ao Meta - Description
- public static final float WCL_GLOBAL_WEIGHT_KEYWORDS
  - Peso atribuído ao Meta - Keywords

- public static final float WCL_GLOBAL_WEIGHT_TITLE
  - Peso atribuído ao Título da Página
- public static final boolean WCL_GLOBAL_DO_SIMPLE_STEMMING
  - Indica se deve remover o plural das palavras

Interface **TokenizerCommon**

Esta interface provê a interface comum que deve ser implementada por qualquer classe que for utilizar o tokenizador

Declaração

```
public interface TokenizerCommon
```

Métodos

- *processToken*
  `public int` **processToken**`( java.lang.String` **token**`, int` **count** `)`
  - **Uso**
    - Este é o método que sera chamado como callback pelo tokenizador para cada token novo que for gerado. Cada classe deve realizar seu tratamento dos tokens neste método.
  - **Parâmetros**
    - `token` - Token a ser processado
    - `count` - Quantas vezes o token apareceu na página analisada
  - **Retorno** - Um dos valores definidos na interface Errs. Caso ele seja diferente de WCL_ERR_NO_ERROR, o tokenizador vai abortar a geração de tokens.

A.3.2 Classes

Classe **CalcDistParam**

Esta classe identifica a distribuição que melhor representa um conjunto de dados. Ela implementa o cálculo dos parâmetros das distribuições da família Johnson (SB e SU), enquanto que o calculo dos parâmetros das demais distribuições é feito através dos MLEs já oferecidos em cada uma delas.

O calculo dos parâmetros é feito através do algoritmo descrito no artigo "The Johnson System: Selection and Parameter Estimation", escrito por James F. Slifker e Samuel S. Shapiro.

Para compilá-la, é necessário o uso da biblioteca SSJ, disponível em
http://www.iro.umontreal.ca/ simardr/ssj/indexf.html

Declaração

public class CalcDistParam
extends java.lang.Object
implements Errs

Métodos

- *acosh*
  `public double` **acosh**`( double` **value** `)`
  - **Uso**
    - Retorna o inverso do cosseno hiperbólico do valor informado

---

- *addRate*
  `public void` **addRate**`( java.lang.String` **set**`, java.lang.String` **strCat**`, java.lang.Double` **rate** `)`
  - **Uso**
    - Este método adiciona a nota informada na lista de notas correspondente ao conjunto de categorias informado
  - **Parâmetros**
    - `set` - Conjunto de dados no qual a nota sera incluída
    - `strCat` - String que identifica as categorias na(s) qual(is) o site foi manualmente classificado
    - `rate` - Nota a ser incluída

- *asinh*
  `public double` **asinh**`( double` **value** `)`
  - **Uso**
    - Retorna o inverso do seno hiperbólico do valor informado

---

- *calcParams*
  `public void` **calcParams**`( java.lang.String` **set**`, java.util.Map` **probDistMap** `)`
  - **Uso**
    - Este método calcula o tipo e os parâmetros das distribuições para cada uma das combinações de categorias existentes no(s) arquivo(s) de treinamento
  - **Parâmetros**
    - `set` - Conjunto que será utilizado para gerar as distribuições
    - `probDistMap` - Mapeamento a ser preenchido com as distribuições de probabilidade

---

- *createCalc*
  `public int` **createCalc**`( java.lang.String` **dir** `)`
  - **Uso**
    - Este método cria um novo calculador de distribuições
  - **Parâmetros**
    - `dir` - Diretório onde os arquivos do mapeamento serão criados
  - **Retorno** - Um dos valores da interface Errs
  - **Veja Também**
    - `webclassifier.CalcDistParam.openCalc(String dir)`
    - `webclassifier.CalcDistParam.saveCalc()`

- *openCalc*
  `public int` **openCalc**`( java.lang.String` **dir** `)`

- *saveCalc*
  `public int` **saveCalc**`( )`
  - **Uso**
    - Este método salva as notas do calculador de distribuições atual
  - **Retorno** - Um dos valores da interface Errs
  - **Veja Também**
    - `webclassifier.CalcDistParam.createCalc(String)`
    - `webclassifier.CalcDistParam.openCalc(String dir)`

Classe **ClassifierMDL**

Esta classe implementa o classificador CAH+MDL, i.e., Codificação Adaptativa de Huffman + Princípio do Menor tamanho de descrição

Para compilá-la, é necessário o uso da biblioteca SSJ, disponível em http://www.iro.umontreal.ca/ simardr/ssj/indexf.html

Declaração

public class ClassifierMDL
extends java.lang.Object
implements Classifier, Errs

Métodos

- *createClassifier*
  `public int` **createClassifier**`( java.lang.String` **trainFile**`, java.lang.String` **dir**`, java.util.Map` **statMap** `)`

- *openClassifier*
  `public int` **openClassifier**`( java.lang.String` **dir** `)`
- *setMapper*
  `public int` **setMapper**`( java.lang.String` **dir** `)`
- *testClassifier*
  `public int` **testClassifier**`( java.lang.String` **testFile**`, double []` **precisionRecall**`, java.util.Set` **badClass** `)`

Classe **ClassifierSVM**

Esta classe implementa o classificador SVM a partir das operações contidas na libSVM
Para compilá-la, é necessário o uso da biblioteca libSVM, disponível em
http://www.csie.ntu.edu.tw/ cjlin/libsvm/index.html

Declaração

public class ClassifierSVM
extends java.lang.Object
implements Classifier, Errs

Construtores

- *ClassifierSVM*
  `public` **ClassifierSVM**`( )`

Métodos

- *createClassifier*
  `public int` **createClassifier**`( java.lang.String` **trainFile**`, java.lang.String` **dir**`, java.util.Map` **statMap** `)`
- *openClassifier*
  `public int` **openClassifier**`( java.lang.String` **dir** `)`
- *setMapper*
  `public int` **setMapper**`( java.lang.String` **dir** `)`
- *testClassifier*
  `public int` **testClassifier**`( java.lang.String` **testFile**`, double []` **precisionRecall**`, java.util.Set` **badClass** `)`

Classe **DynamicHuffmanTree**

Esta classe implementa o algoritmo de Vitter, para a construção dinâmica das árvores de Huffman. Ele é baseado no código original escrito em C por Karl Malbrain <karl_m@acm.org>, porém extensa e amplamente modificado, com o objetivo de se atingir máxima performance e incluir a remoção de documentos da árvore.

Devido à sua otimização, aparentemente existem pequenos trechos de código redundantes, porém que causam um grande ganho de velocidade.

Declaração

```
public class DynamicHuffmanTree
extends java.lang.Object
```

Métodos

- *containsSymbol*
  `public boolean` **containsSymbol**`( int` **symbol** `)`
  - **Uso**
    - Este método indica se o símbolo informado está presente na árvore
  - **Parâmetros**
    - `symbol` - Símbolo a ser verificado
  - **Retorno** - `true` Se o símbolo está presente na árvore ou `false` caso contrário.
  - **Exceções**
    - `java.lang.IllegalArgumentException` -

- *decrementCount*
  `public void` **decrementCount**`( int` **symbol**`, int` **count** `)`
  - **Uso**
    - Decrementa a contagem do símbolo na árvore de Huffman. Se a contagem chegar a 0, o símbolo é removido da árvore.
  - **Parâmetros**
    - `symbol` - Símbolo a ter sua contagem diminuída
    - `count` - Valor a diminuir
  - **Exceções**
    - `java.lang.IllegalArgumentException` -

- *getBits*
  `public int` **getBits**`( int` **symbol** `)`
  - **Uso**
    - Retorna o número de bits necessários para codificar o símbolo informado

  - **Parâmetros**
    * `symbol` - Símbolo a ser codificado
  - **Retorno** - Número de bits necessários para codificar o símbolo informado
  - **Exceções**
    * `java.lang.IllegalArgumentException` -

- *incrementCount*
  `public void` **incrementCount**`( int` **symbol**`, int` **count** `)`

  - **Uso**
    * Incrementa a contagem do símbolo na árvore de Huffman
  - **Parâmetros**
    * `symbol` - Símbolo a ter sua contagem aumentada
    * `count` - Valor a incrementar
  - **Exceções**
    * `java.lang.IllegalArgumentException` -

Classe **FileTokenizer**

Esta classe realiza a tokenização da página HTTP informada, gravando o resultado em um arquivo, que depois pode ser utilizado para treinar um classificador

Declaração

public class FileTokenizer **extends** java.lang.Object **implements** TokenizerCommon, Errs

Métodos

- *processToken*
  `public int` **processToken**`( java.lang.String` **token**`, int` **count** `)`

  - **Uso**
    * Este é o método que sera chamado como callback pelo tokenizador para gravar os tokens no disco
  - **Parâmetros**
    * `count` - Quantas vezes o token apareceu na página analisada
  - **Retorno** - Um dos valores definidos na interface Errs. Caso ele seja diferente de WCL_ERR_NO_ERROR, o tokenizador vai abortar a geração de tokens.

- *tokenizeFile*
  `public int` **tokenizeFile**`( java.net.URL` **url**`, java.lang.String` **filePath** `)`

  - **Uso**
    - Este método tokeniza a URL informada e grava os tokens no arquivo especificado. Caso a URL seja um "redirect", os tokens serão gerados da página de destino.
  - **Parâmetros**
    - `url` - URL completa da página a ser tokenizada
    - `filePath` - Caminho completo do arquivo a ser gerado com os tokens presentes na página
  - **Retorno** - Um dos valores definidos na interface Errs

Classe **IdxMapper**

Esta classe faz o mapeamento entre valores numéricos das categorias e dos atributos e seus nomes correspondentes. Ela faz também a transformação dos arquivos tokenizados individualmente em um arquivo único com mapeamento global dos atributos e categorias, em formato numérico e compatível com o da libsvm. Este arquivo pode então ser lido diretamente pelos classificadores para realizar seu treinamento.

Declaração

public class IdxMapper
extends java.lang.Object
implements Globals, Errs

Métodos

- *addNameId*
  `public int` **addNameId**`( java.lang.String` **filename** `)`

  - **Uso**
    - Este método retorna o índice da linha de um arquivo tokenizado no arquivo VSM, da mesma forma que o método `getNameId`, porém, caso o arquivo já exista na lista, sera retornado -1, que corresponde a um erro.
  - **Parâmetros**
    - `filename` - Nome do arquivo a ser incluído no mapeamento
  - **Retorno** - O índice da linha ou `-1` caso o arquivo já exista

- *addTokenizedFile*
  `public int` **addTokenizedFile**`( java.lang.String` **cat**`, java.lang.String` **filename** `)`
  - **Uso**
    - Este método inclui os tokens presentes no arquivo informado no arquivo global, que será posteriormente utilizado no treinamento dos classificadores.
  - **Parâmetros**
    - `cat` - Nome da categoria a qual o arquivo a ser incluído pertence
    - `filename` - Caminho completo do arquivo tokenizado que será incluído no treinamento
  - **Retorno** - Um dos valores definidos na interface Errs

- *addWordId*
  `public int` **addWordId**`( java.lang.String` **word** `)`
  - **Uso**
    - Este método retorna o índice (da dimensão) de uma palavra no vetor de representação do documento, da mesma forma que o método `getWordId`, porém, caso a palavra ainda não exista na lista ela será incluída.
  - **Parâmetros**
    - `word` - Palavra a ser pesquisada/incluída
  - **Retorno** - O índice da palavra ou `-1` caso ela não possa ser incluída

---

- *createMapper*
  `public int` **createMapper**`( java.util.Map` **categories**`, java.lang.String` **dir** `)`
  - **Uso**
    - Este método cria um novo mapeamento de categorias/palavras. **Caso já exista algum mapeamento anterior no diretório especificado no construtor, ele será substituído.**
  - **Parâmetros**
    - `categories` - Lista de pares (nome, id) das categorias que farão parte dos classificadores criados usando este mapeamento
    - `dir` - Diretório onde os arquivos do mapeamento serão criados
  - **Retorno** - Um dos valores da interface Errs
  - **Veja Também**
    - `webclassifier.IdxMapper.openMapper(String)`

- *getCatId*
  `public Integer` **getCatId**`( java.lang.String` **cat** `)`
  - **Uso**

    * Este método retorna o ID de uma categoria dado o seu nome. É necessário que a lista de categorias já tenha sido carregada ou criada antes que este método possa ser chamado.
  - **Parâmetros**
    * `cat` - Nome da categoria a ser pesquisada
  - **Retorno** - O ID da categoria ou `null` caso ela não exista
  - **Veja Também**
    * `webclassifier.IdxMapper.getCatName(int)` (in A.3.2, page 99)
    * `webclassifier.IdxMapper.createMapper(Map, String)`
    * `webclassifier.IdxMapper.openMapper(String)`

- *getCatList*
  `public Set` **getCatList**`( )`
  - **Uso**
    * Este método retorna a lista com todas as categorias.
  - **Retorno** - A lista de categorias conhecida ou `null` no caso de erros
  - **Veja Também**
    * `webclassifier.IdxMapper.openMapper(String)`
    * `webclassifier.IdxMapper.getCatId(String)`

- *getCatName*
  `public String` **getCatName**`( int` **id** `)`
  - **Uso**
    * Este método retorna o nome de uma categoria dado o seu ID. É necessário que a lista de categorias já tenha sido carregada ou criada antes que este método possa ser chamado.
  - **Parâmetros**
    * `id` - Id da categoria a ser pesquisada
  - **Retorno** - O nome da categoria ou `null` caso ela não exista
  - **Veja Também**
    * `webclassifier.IdxMapper.getCatId(String)`
    * `webclassifier.IdxMapper.createMapper(Map, String)`
    * `webclassifier.IdxMapper.openMapper(String)`

- *getIdName*
  `public String` **getIdName**`( int` **id** `)`
  - **Uso**
    * Este método retorna o nome de um arquivo tokenizado correspondente a linha informada do arquivo VSM. Ele só é usado para fins de identificação de exemplos que não foram bem classificados, não sendo usado na operação normal do classificador.
  - **Parâmetros**

    * `id` - Número da linha do VSM a pesquisar
  - **Retorno** - O nome do arquivo ou `null` caso a linha não exista

---

- *getNameId*
  `public int` **getNameId**`( java.lang.String` **filename** `)`
  - **Uso**
    * Este método retorna o índice da linha correspondente a um arquivo tokenizado no arquivo VSM.
  - **Parâmetros**
    * `filename` - Nome do arquivo a ser pesquisado
  - **Retorno** - O índice do arquivo ou `-1` caso ele não exista

---

- *getWordId*
  `public int` **getWordId**`( java.lang.String` **word** `)`
  - **Uso**
    * Este método retorna o índice (da dimensão) de uma palavra no vetor de representação do documento.
  - **Parâmetros**
    * `word` - Palavra a ser pesquisada
  - **Retorno** - O índice da palavra ou `-1` caso ela não exista

---

- *openMapper*
  `public int` **openMapper**`( java.lang.String` **dir** `)`
  - **Uso**
    * Este método abre um mapeamento de categorias/palavras. Ele deve ser chamado antes dos demais métodos desta classe. Caso o mapeamento ainda não exista, ele pode ser criado por `createMapper`.
  - **Parâmetros**
    * `dir` - Diretório onde se encontram os arquivos do mapeamento que será aberto
  - **Retorno** - Um dos valores da interface Errs
  - **Veja Também**
    * `webclassifier.IdxMapper.createMapper(Map, String)`
    * `webclassifier.IdxMapper.saveMapper()`

- *saveMapper*
  `public int` **saveMapper**`( )`
  - **Uso**
    * Este método salva os dados do mapeamento atual no diretório especificado ao se criar ou abrir o mapeamento.
  - **Retorno** - Um dos valores da interface Errs
  - **Veja Também**
    * `webclassifier.IdxMapper.createMapper(Map, String)`
    * `webclassifier.IdxMapper.openMapper(String)`

Classe **Main**

Classe principal do Classificador

Declaração

public class Main
extends java.lang.Object

Construtores

- *Main*
  `public` **Main**`( )`

Métodos

- *main*
  `public static void` **main**`( java.lang.String [] args )`
  - **Uso**
    - Método principal do programa. Faz parser da linha de comando e executa a operação solicitada pelo usuário
  - **Parâmetros**
    - `args` - Argumentos da linha de comando

- *splitVsm*
  `public static int` **splitVsm**`( java.lang.String` **filename** `)`
  - **Uso**
    - Este método separa o arquivo VSM informado em 10 pares de arquivos, usados para treinamento ou teste do classificador em 10-fold cross validation.
  - **Parâmetros**
    - `filename` - Nome do arquivo VSM a ser separado

Classe **Splitter**

Esta classe serve para separar um arquivo VSM em 20 arquivos. 10 para treinar um classificador e 10 para testá-lo.

Declaração

public class Splitter
extends java.lang.Object
implements Errs

Construtores

- *Splitter*
  `public` **Splitter**`( )`

Métodos

- *splitVSM*
  `public int` **splitVSM**`( java.lang.String` **sourceFile** `)`
  - **Uso**
    - Este método separa o arquivo VSM informado em 10 pares de arquivos. Cada par consiste de um arquivo de treinamento e um de teste de um classificador. Os nomes gerados serão idênticos ao do arquivo original, exceto que eles terão um sufixo que informa o número do arquivo e se ele é para ser usado para treinamento ou teste.
  - **Parâmetros**
    - `sourceFile` - Nome do arquivo VSM a ser separado
  - **Retorno** - Um dos valores da interface Errs

Classe **StopWords**

Esta classe consiste na lista de stopwords que devem ser ignoradas na geração dos tokens de uma página e do método para pesquisá-las.

Declaração

public class StopWords
extends java.lang.Object

Construtores

- *StopWords*
  `public` **StopWords**`( )`

Métodos

- *isStopword*
  `public static boolean` **isStopword**`( java.lang.String` **token** `)`
  - **Uso**
    - Este método faz pesquisa binária na lista de stopwords para ver se o token informado se encontra nela
  - **Parâmetros**
    - `token` - Token a ser procurado
  - **Retorno** - `false` se o token não está na lista de stopwords ou `true` caso ele esteja na lista de stopwords

Classe **Tokenizer**

Esta classe realiza a tokenização da página HTTP informada, extraindo os termos, que serão depois utilizados pelo sistema de classificação. Somente a primeira página de cada URL informada será tokenizada, mas caso ela possua frames, todos serão analisados como se fossem uma mesma página.

Declaração

public class Tokenizer
extends java.lang.Object
implements Globals, Errs

Métodos

- *generateTokens*
  `public int` **generateTokens**`( java.net.URL` **url** `)`
  - **Uso**
    - Este método gera os tokens de todos os frames da página principal a ser tokenizada. Ele basicamente chama o método doTokenGeneration quantas vezes quanto for necessário para incluir todos os frames
  - **Parâmetros**
    - `url` - URL completa do site a ser tokenizado
  - **Retorno** - Um dos valores definidos na interface Errs

Referências

[AH08] Jesse Alpert and Nissan Hajaj. We knew the web was big. `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`, Jul 2008.

[BFC+06] Andrej Bratko, Bogdan Filipič, Gordon V. Cormack, Thomas R. Lynam, and Blaž Zupan. Spam filtering using statistical data compression models. *J. Mach. Learn. Res.*, 7:2673–2698, 2006.

[BGMFM02] Janez Brank, Marko Grobelnik, Natasa Milic-Frayling, and Dunja Mladenic. Feature selection using linear support vector machines. Technical report, Microsoft Research, Microsoft Corporation, Redmond, WA, USA, Jun 2002.

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

[BL07] Ígor Assis Braga and Marcelo Ladeira. Um modelo adaptativo para a filtragem de spam. In *Anais do VI Encontro Nacional de Inteligência Artificial (ENIA'2007)*, Rio de Janeiro, RJ, Brasil, Jul 2007.

[BLC95] T. Berners-Lee and D. Connolly. Hypertext markup language - 2.0. RFC 1866, Nov 1995.

[BLSB04] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004.

[BM98] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, New York, NY, USA, 1998. ACM.

[BS00] António Branco and João Silva. Very high accuracy rule-based nominal lemmatization with a minimal lexicon. Technical report, University of Lisbon, Department of Informatics, Lisbon, Portugal, 2000.

[Bur96] Christopher J. C. Burges. Simplified support vector decision rules. In *Proceedings of the 13th International Conference on Machine Learning*, pages 71–77. Morgan Kaufmann, 1996.

[Bur98] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[cbi08] Over 90 minutes a week spent on personal web-surfing at work - cbi. `http://www.cbi.org.uk/ndbs/press.nsf/0363c1f07c6ca12a8025671c00381cc7/94d596bf6bcd69708025745e003b722b?OpenDocument`, Jun 2008.

[CC07] Ana Cardoso-Cachopo. Datasets for single-label text categorization. `http://www.gia.ist.utl.pt/~acardoso/datasets`, 2007.

[CCO03] Ana Cardoso-Cachopo and Arlindo L. Oliveira. An empirical comparison of text categorization methods. In Mario A. Nascimento, Edleno S. De Moura, and Arlindo L. Oliveira, editors, *Proceedings of SPIRE-03, 10th International Symposium on String Processing and Information Retrieval*, pages 183–196, Manaus, BR, 2003. Springer Verlag, Heidelberg, DE. Published in the "Lecture Notes in Computer Science" series, number 2857.

[CDI98] Soumen Chakrabarti, Byron Dom, and Piotr Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD Conference*, pages 307–318, 1998.

[CL01] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CM07] Michelangelo Ceci and Donato Malerba. Classifying web documents in a hierarchy of categories: a comprehensive study. *J. Intell. Inf. Syst.*, 28(1):37–78, 2007.

[Cro08] Martyn Crof. Social networking sites costs uk 6.5 billion pounds in lost productivity. `http://www.gss.co.uk/press/?&id=17`, Jan 2008.

[CS00] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. In *In Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 35–46, 2000.

[CS02] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, 2002.

[CV95] Corinna Cortes and Vladimir Vapnik. Support vector networks. In *Machine Learning*, pages 273–297, 1995.

[CWZH06] Zhouyao Chen, Ou Wu, Mingliang Zhu, and Weiming Hu. A novel web page filtering system by combining texts and images. *Web Intelligence, IEEE / WIC / ACM International Conference on*, pages 732–735, 2006.

[DC00] Susan Dumais and Hao Chen. Hierarchical classification of web content. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 256–263, New York, NY, USA, 2000. ACM.

[dCGT03] F. de Comite, R. Gilleron, and M. Tommasi. Learning multi-label alternating decision trees from texts and data. In *Proceedings of the 3rd International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM 2003)*, pages 35–49, Leipzig, Germany, Mai-Jul 2003.

[DLR00] A. P. Dempster, N. M. Laird, and D. B. Rubin. CMU World Wide Knowledge Base (WebKB) project. `http://www.cs.cmu.edu/~WebKB`, 2000.

[DPHS98] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *CIKM '98: Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155, New York, NY, USA, 1998. ACM.

[DS03] Franca Debole and Fabrizio Sebastiani. Supervised term weighting for automated text categorization. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 784–788, New York, NY, USA, 2003. ACM.

[DS07] M. Indra Devi and K. Selvakuberan. Fast web page categorization without the web page. *International Conference on Semantic Web and Digital Libraries (ICSD-2007). ARD Prasad & Devika P. Madalli*, May 2007.

[EMH05] Yasser EL-Manzalawy and Vasant Honavar. *WLSVM: Integrating LibSVM into Weka Environment*, 2005. Software available at `http://www.cs.iastate.edu/~yasser/wlsvm`.

[FCW00] Eibe Frank, Chang Chui, and Ian H. Witten. Text categorization using compression models. In *DCC '00: Proceedings of the Conference on Data Compression*, page 555, Washington, DC, USA, 2000. IEEE Computer Society.

[Fel98] C. Fellbaum. *WordNet: an electronic lexical database*. MIT Press, Cambridge, MA, 1998.

[Fle87] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, second edition, 1987.

[For03] George Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289–1305, 2003.

[Für02] Johannes Fürnkranz. Round robin classification. *J. Mach. Learn. Res.*, 2:721–747, 2002.

[GMS05] Yann Guermeur, Myrian Maumy, and Frédéric Sur. Model selection for multi-class svms. In *ASMDA'05: Proceedings of the 11th Symposium on ASMDA*, pages 507–517, Brest, France, 2005. ASMDA.

[GQ04] Teresa Gonçalves and Paulo Quaresma. The impact of nlp techniques in the multilabel text classification problem. In *Intelligent Information Systems*, pages 424–428, 2004.

[Grü05] Peter Grünwald. *A Tutorial Introduction to the Minimum Description Length Principle*, chapter 1-2. MIT Press, April 2005.

[GS04] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 22–30. Springer, 2004.

[GTL+02] Eric J. Glover, Kostas Tsioutsiouliklis, Steve Lawrence, David M. Pennock, and Gary W. Flake. Using web structure for classifying and describing web pages. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 562–569, New York, NY, USA, 2002. ACM.

[Gue07] Yann Guermeur. *SVM multiclasses, théorie et applications*. Habilitation à diriger des recherches, Université Nancy 1, Nov 2007.

[HBLH94] William Hersh, Chris Buckley, T. J. Leone, and David Hickam. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 192–201, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[Huf52] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[Joa98] Thorsten Joachims. Making large-scale svm learning practical. LS8-Report 24, Universität Dortmund, LS VIII-Report, 1998.

[Joa02] Thorsten Joachims. *Learning to Classify Text using Support Vector Machines*. Kluwer Academic Publishers, Dordrecht, NL, 2002.

[Kan04] Min-Yen Kan. Web page classification without the web page. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 262–263, New York, NY, USA, 2004. ACM.

[KPD90] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In J. Fogelman, editor, *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag, 1990.

[KR01] Asirvatham Kranthi and Kumar Ravi. Web page categorization based on document structure. *IEEE National Convention*, Dec 2001.

[LDG00] Fabien Letouzey, François Denis, and Rémi Gilleron. Learning from positive and unlabeled examples. In *Procs. of the 11th International Conference on Algorithmic Learning Theory*, pages 71–85, 2000.

[Lew97] David D. Lewis. Reuters-21578 text categorization collection. `http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`, 1997.

[LLW04] Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99:67–81, January 2004.

[LLYL02] Bing Liu, Wee Sun Lee, Philip S. Yu, and Xiaoli Li. Partially supervised classification of text documents. In *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, pages 387–394, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[LMR94] G. P. Lopes, N. Marques, and V. Rocio. Polaris: Portuguese lexicon acquisition and retrieval interactive system. In L. Sterling, editor, *Proceedings of the Second International Conference on the Practical Applications of Prolog*, pages 665–665. The Practical Application Company, 1994.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[OFG97a] E. Osuna, R. Freund, and F. Girosit. Training support vector machines: an application to face detection. *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 130–136, 1997.

[OFG97b] Edgar Osuna, Robert Freund, and Federico Girosi. An improved training algorithm for support vector machines. In *Proceedings IEEE NNSP'97*, pages 276–285. IEEE, 1997.

[OFG97c] Edgar Osuna, Robert Freund, and Federico Girosi. Support vector machines: Training and applications. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1997.

[OH01] V. M. Orengo and C. Huyck. A Stemming Algorithm for Portuguese Language. In *Proc. of Eigth Symposium on String Processing and Information Retrieval (SPIRE 2001) - Chile*, pages 186–193, 2001.

[PCSt00] John C. Platt, Nello Cristianini, and John Shawe-taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems*, pages 547–553. MIT Press, 2000.

[Pie00] John M. Pierre. Practical issues for automated categorization of web sites. In *Electronic Proc. ECDL 2000 Workshop on the Semantic Web*, 2000.

[Pla98] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines, 1998.

[Pla99] John C. Platt. Using analytic qp and sparseness to speed training of support vector machines. In *In Neural Information Processing Systems 11*, pages 557–563. MIT Press, 1999.

[Por80] M. Porter. The Porter Stemming Algorithm. *Program, 14 no. 3*, pages 130–137, Jul 1980.

[QD06] Xiaoguang Qi and Brian D. Davison. Knowing a web page by the company it keeps. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 228–237, New York, NY, USA, 2006. ACM.

[Rij79] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 1979.

[Ris78] J. J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Ris96] J. J. Rissanen. Fisher information and stochastic complexity. *IEEE Trans on Information Theory*, 42(1):40–47, 1996.

[RK04] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141, 2004.

[SB06] Péter Schönhofen and András A. Benczúr. Exploiting extremely rare features in text categorization. In *ECML*, pages 759–766, 2006.

[Seb99] Fabrizio Sebastiani. A tutorial on automated text categorisation. In Analia Amandi and Ricardo Zunino, editors, *Proceedings of the 1st Argentinian Symposium on Artificial Intelligence (ASAI'99)*, pages 7–35, Buenos Aires, AR, 1999.

[SH68] Samuel S. Shapiro and Gerald J Hahn. *Statistical models in Engineering*. John Wiley & Sons, New York, 1968.

[SLN02] Aixin Sun, Ee-Peng Lim, and Wee-Keong Ng. Web classification using support vector machine. In *WIDM '02: Proceedings of the 4th international workshop on Web information and data management*, pages 96–99, New York, NY, USA, 2002. ACM.

[SS80] James F. Slifker and Samuel S. Shapiro. The johnson system: Selection and parameter estimation. *Technometrics*, 22(2):239–246, May 1980.

[SSWB00] Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New support vector algorithms. *Neural Comput.*, 12(5):1207–1245, 2000.

[STC04] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[TK07] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007.

[Vap82] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[Vit87] Jeffrey Scott Vitter. Design and analysis of dynamic huffman codes. *J. ACM*, 34(4):825–845, 1987.

[WAB+07] Robert Wetzker, Tansu Alpcan, Christian Bauckhage, Winfried Umbrath, and Sahin Albayrak. An unsupervised hierarchical approach to document categorization. In *Web Intelligence*, pages 482–486, 2007.

[Web08] Web server survey. `http://news.netcraft.com/archives/2008/09/30/september_2008_web_server_survey.html`, Sep 2008.

[Wek] Weka Machine Learning Project. The Waikato Environment for Knowledge Analysis - WEKA. URL http://www.cs.waikato.ac.nz/~ml/weka.

[wHjL02] Chih wei Hsu and Chih jen Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

[Wor08] Internet world users by language. `http://www.internetworldstats.com/stats7.htm`, 2008.

[WW98] J. Weston and C. Watkins. Multi-class support vector machines. Technical report, University of London, 1998.

[XHL06] Weimin Xue, Weitong Huang, and Yuchang Lu. Application of svm in web page categorization. In *Granular Computing, 2006 IEEE International Conference on*, pages 469–472. IEEE, May 2006.

[YHcC04] Hwanjo Yu, Jiawei Han, and Kevin Chen chuan Chang. Pebl: web page classification without negative examples. *IEEE Transactions on Knowledge and Data Engineering*, 16:70–81, 2004.

[YL99] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 42–49, Berkeley, US, 1999. ACM Press, New York, US.

[YP97] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[ZG09] Xiaojin Zhu and Andrew B. Goldberg. *Introduction to Semi-Supervised Learning*. Morgan & Claypool Publishers, 2009.

[Zha06] Min-Ling Zhang. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Trans. on Knowl. and Data Eng.*, 18(10):1338–1351, 2006. Senior Member-Zhi-Hua Zhou.

[ZL03] Dell Zhang and Wee Sun Lee. Question classification using support vector machines. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 26–32, New York, NY, USA, 2003. ACM.

[ZMN05] Yan Zhou, Madhuri S. Mulekar, and Praveen Nerellapalli. Adaptive spam filtering using dynamic feature space. In *ICTAI '05: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence*, pages 302–309, Washington, DC, USA, 2005. IEEE Computer Society.

[ZZ07] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recogn.*, 40(7):2038–2048, 2007.