O Algoritmo de Aprendizado Semi-Supervisionado CO-TRAINING e sua Aplicação na Rotulação de Documentos

Edson Takashi Matsubara

O Algoritmo de Aprendizado Semi-Supervisionado CO-TRAINING e sua Aplicação na Rotulação de Documentos

Edson Takashi Matsubara

Orientadora: *Profª Drª Maria Carolina Monard*

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

USP - São Carlos
Abril/2004

Dedicatória

à minha família

Agradecimentos

Todo grande sonho não nasce sozinho. Cada pedaço da realização tem o suor, a dedicação e a harmonia de quem sonhou junto. Eu tenho um sonho, não tão grandioso, e as poucas partes desse sonho que já foram realizados aconteceram não somente pela minha vontade, mas pela sorte de ter a família, os mestres e amigos que tenho. Sem essas pessoas, eu nada seria.

A começar pela minha família, a espiritualidade e a magia que os caminhos da vida nos levam, muitas vezes torna a vida assustadora. A minha família é o meu porto seguro, é a minha fortaleza, não importa onde, nem quando, ela está sempre lá para me dar a segurança de seguir esse meu sonho. Devo muito ao meu pai Ryuiti Matsubara, cujo nome significa o primeiro dragão, que me deu exemplos de garra e coragem os quais me dão forças para enfrentar os maiores desafios da vida. A minha mãe Ritie Tomonaga Matsubara, sobrenome significa amizade e companheirismo, que dedica sua vida para harmonia da minha família e uma eterna companheira de meu pai. A meus irmãos, Koiti e Sayuri, por serem os melhores irmãos que alguém pode ter. Ao Ditian (avô), pela sabedoria e abnegação. A todos os meus antepassados pelas minhas boas origens.

Nada posso ser sem um bom mestre. A senhora, Prof. Carolina, tem sido para mim um exemplo de postura e competência intelectual, seus ensinamentos vão além de teoremas e modelos complicados, a cada dia de convívio aprendo coisas que guardo para minha vida. A Prof. Solange, cuja garra e eloqüência ajudaram na minha formação durante a graduação, e, graças a isso, hoje estou aqui tento esta oportunidade de agradecer.

A vida é cheia de mistérios. Só depois de anos de convivência a gente percebe que seu melhor amigo e você podem fazer qualquer coisa, ou nada, e terem bons momentos juntos. Marcio, Alex, Sidão, Kleber, Mauro e Testa, voces são pessoas formidáveis. Ao pensar que cada um deve tomar seu rumo um dia já sinto saudades dos anos de convivência e bons momentos que todos passamos juntos. Bons amigos são a família que nos permitiram escolher.

Gustavo, voce tem sido um grande amigo e também um segundo orientador, sem as conversas que tivemos e sua ajuda, provavelmente este trabalho não seria nem metade do que é. Ronaldo também faz parte disso. Claudinha, obrigado pela ajuda, e por sempre poder contar com voce.

Ao pessoal do LABIC, Roberta, Jaqueline, Rodrigo, Débora, Marcos (Rodomoço), Marcos Geromini, Eduardo, Katti, Lorena, Gedson, Quiles, Christiane, Mariza, Richardson, Jean, Edson Melanda, Marina, Veronica, Nagaisan, Flávia, Huei, Vinicius e Patricia, obrigado pelos momentos de descontração que fazem do laboratório uma lugar cada vez melhor.

Kelly Cristiane Ito, não temos que mudar as pessoas se entendermos que as pessoas mudam. Apesar de nossas diferenças você é uma pessoa muito especial para mim.

Meus sinceros agradecimentos a todos que me auxiliaram a solidificar este trabalho.

Resumo

Em Aprendizado de Máquina, a abordagem supervisionada normalmente necessita de um número significativo de exemplos de treinamento para a indução de classificadores precisos. Entretanto, a rotulação de dados é freqüentemente realizada manualmente, o que torna esse processo demorado e caro. Por outro lado, exemplos não-rotulados são facilmente obtidos se comparados a exemplos rotulados. Isso é particularmente verdade para tarefas de classificação de textos que envolvem fontes de dados *on-line* tais como páginas de internet, *email* e artigos científicos. A classificação de textos tem grande importância dado o grande volume de textos disponível on-line. Aprendizado semi-supervisionado, uma área de pesquisa relativamente nova em Aprendizado de Máquina, representa a junção do aprendizado supervisionado e não-supervisionado, e tem o potencial de reduzir a necessidade de dados rotulados quando somente um pequeno conjunto de exemplos rotulados está disponível. Este trabalho descreve o algoritmo de aprendizado semi-supervisionado CO-TRAINING, que necessita de duas descrições de cada exemplo. Deve ser observado que as duas descrições necessárias para CO-TRAINING podem ser facilmente obtidas de documentos textuais por meio de pré-processamento. Neste trabalho, várias extensões do algoritmo CO-TRAINING foram implementadas. Ainda mais, foi implementado um ambiente computacional para o pré-processamento de textos, denominado PRETEXT, com o objetivo de utilizar CO-TRAINING em problemas de classificação de textos. Os resultados experimentais foram obtidos utilizando três conjuntos de dados. Dois conjuntos de dados estão relacionados com classificação de textos e o outro com classificação de páginas de internet. Os resultados, que variam de excelentes a ruins, mostram que CO-TRAINING, similarmente a outros algoritmos de aprendizado semi-supervisionado, é afetado de maneira bastante complexa pelos diferentes aspectos na indução dos modelos.

Abstract

In Machine Learning, the supervised approach usually requires a large number of labeled training examples to learn accurately. However, labeling is often manually performed, making this process costly and time-consuming. By contrast, unlabeled examples are often inexpensive and easier to obtain than labeled examples. This is especially true for text classification tasks involving on-line data sources, such as web pages, email and scientific papers. Text classification is of great practical importance today given the massive volume of online text available. Semi-supervised learning, a relatively new area in Machine Learning, represents a blend of supervised and unsupervised learning, and has the potential of reducing the need of expensive labeled data whenever only a small set of labeled examples is available. This work describes the semi-supervised learning algorithm CO-TRAINING, which requires a partitioned description of each example into two distinct views. It should be observed that the two different views required by CO-TRAINING can be easily obtained from textual documents through pré-processing. In this works, several extensions of CO-TRAINING algorithm have been implemented. Furthermore, we have also implemented a computational environment for text pre-processing, called PRETEXT, in order to apply the CO-TRAINING algorithm to text classification problems. Experimental results using CO-TRAINING on three data sets are described. Two data sets are related to text classification and the other one to web-page classification. Results, which range from excellent to poor, show that CO-TRAINING, similarly to other semi-supervised learning algorithms, is affected by modelling assumptions in a rather complicated way.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AC Aquisição de Conhecimento

AM Aprendizado de Máquina

DSX *Discover Standard Sintax*

DOL *Discover Object Library*

EM *Expectation Maximization*

EC Engenheiro de Conhecimento

FIP Uma Ferramenta Inteligente de Apoio à Pesquisa

IA Inteligência Artificial

KDD *Knowledge Discovery in Databases*

Labic Laboratório de Inteligência Computacional

MT Mineração de Texto

SVM *Support Vector Machines*

UCI *University of California, Irvine*

Lista de Algoritmos

CAPÍTULO

1

Introdução

Máquinas pensantes que possam aprender e explicar o conhecimento adquirido têm sido o sonho que vem, há anos, povoado a mente de visionários e cientistas. A criação dessas fabulosas máquinas nos obriga a explorar o pensamento do homem, conhecido cientificamente como *homo sapiens* (*homem com sabedoria*).

Muitos filósofos acreditam na existência de dois mundos: o mundo das idéias e o mundo real. O mundo das idéias é um lugar onde tudo é possível e perfeito. O mundo real é o mundo onde nem tudo é possível e muito menos perfeito. O pensamento é o que permite ao ser humano entrar no mundo das idéias. Ao invés de nos perguntarmos, "o que veio antes, o ovo ou a galinha ?", talvez seja mais interessante a seguinte pergunta, "o que veio antes, a galinha ou a *idéia* da galinha ?".

Vários filósofos que antecederam Platão (427-347 a.C.) queriam encontrar algo de eterno e de imutável em meio a todas as mudanças. Foi assim que ele chegou às idéias perfeitas, que estão acima do mundo sensorial. Para Platão, o grau máximo de realidade era o mundo das idéias. Para ele, tudo surgia do mundo das idéias, como o amor (ideal ou platônico). Portanto, para Platão, a *idéia* da "galinha" vinha antes da galinha e do ovo.

Aristóteles (384-322 a.C.) concordava que, em si, a *idéia* da galinha era eterna e imutável. Mas para ele, a *idéia* galinha não passava de um conceito criado pelos homens e para os homens, depois de eles terem visto um certo número de galinhas. Para Aristóteles, a *idéia* galinha consiste nas características da galinha. Em outras palavras, Aristóteles entendia por *idéia* da galinha aquilo que todas as galinhas têm em comum.

Aristóteles achava que todas as nossas idéias tinham entrado em nossa

consciência através do que víamos e ouvíamos. Mas nós também temos uma razão inata. Temos uma capacidade inata de ordenar em diferentes grupos e classes as nossas impressões sensoriais. É assim que surgem conceitos como "pedra", "planta", "animal" e a nossa famosa "galinha". Com a idéia de por ordem nos conceitos, Aristóteles fundou a *Lógica*, e estabeleceu uma série de normas rígidas para que conclusões ou provas pudessem ser consideradas logicamente válidas, criando, com o pensamento, um laço entre o mundo ideal e o mundo real. Aristóteles acreditava que a razão era algo inato e natural ao homem, sendo uma das principais características para a sua inteligência. Entretanto, ele ressalta que a razão permanece totalmente vazia enquanto não percebemos nada. Uma pessoa, portanto, não possui "idéias" inatas. Desse modo, a percepção do mundo é de vital importância para a criação de conceitos. Restringir essa percepção do mundo pode, muitas vezes levar a conceitos equivocados e sem o menor sentido.

As idéias de Aristóteles sobre o uso da razão para a criação de conceitos, são muito semelhantes as utilizadas pelos principais algoritmos que possibilitam algum aprendizado. Normalmente, é fornecida a uma máquina uma determinada capacidade de ordenar em diferentes classes todas as suas impressões "sensoriais" para construir os conceitos.

Com o passar dos anos, importantes nomes que vão de Leonardo da Vinci (1452-1519), com sua calculadora mecânica que não pôde ser construída em sua época por estar muito além do seu tempo, Blaise Pascal (1623-1662), com sua Pascalina, até Jonh Mauchly e John Eckert, com seu segredo militar chamado ENIAC que, quando ligado, quase causava um *blackout* em boa parte da Pensilvânia, e muitos outros, contribuíram para a construção dos primeiros computadores. Assim, o sonho da construção de uma máquina pensante começa a dar seus primeiros passos. Em 1958, Rosenblatt (1958) propõe o Perceptron, um modelo baseado em neurônios que possuía a capacidade de aprender alguns padrões. Surge, assim, os primórdios de uma área de pesquisa de Inteligência Artificial (IA) denominada Aprendizado de Máquina (AM).

Desde então, AM teve grandes avanços. Paradigmas como o estatístico, *instance-based*, conexionista, genético e simbólico têm sido amplamente explorados e pesquisados para a construção de melhores algoritmos de aprendizado. Entretanto, o aprendizado de máquina é muito mais restrito e simplificado que o aprendizado humano. Por outro lado, os computadores tem capacidade de trabalhar com volumes de dados muito maiores que os humanos.

Atualmente, a humanidade vive um momento em que as inovações tecnológicas ocorrem muito rapidamente. Gordon Moore, co-fundador da Intel, em 1965 fez uma observação sobre o número de transistores por polegada qua-

drada e constatou que esse número dobra a cada 18 meses. Essa é a lei de Moore, na qual a velocidade de processamento computacional duplica a cada 18 meses, crescendo exponencialmente. Assim como a capacidade de processamento, a capacidade de armazenamento de dados cresce nas mesmas proporções. Sistemas de comércio eletrônico, automação industrial, seqüenciamento genético, entre muitos outros, facilmente geram alguns terabytes de dados. Mas, na verdade, apenas uma pequena parte desses dados são explorados porque, na maioria das vezes, os volumes de dados são muito grandes ou muito complexos para serem analisados. Esse é um dos grandes desafios para uma área de aplicação de aprendizado de máquina chamando *Knowledge Discovery in Databases* (KDD).

A restrita percepção do mundo através da tabela atributo-valor utilizada pela maioria dos algoritmos de AM, e a dificuldade em se explorar os dados, tem sido dois dos grandes desafios para pesquisadores de AM. Além do que, a tarefa de aprendizado está normalmente associado a um professor que possa orientar o aprendizado e dizer, por exemplo, isso é um mamífero, isso é uma casa, aquilo é um carro. Em AM, existem diversos algoritmos que necessitam de dados com esses conceitos associados (dados rotulados) para realizar algum aprendizado. Entretanto, dado o grande tamanho das atuais bases de dados, tem sido cada vez mais difícil encontrar bases rotuladas. Surge, então, uma mistura de aprendizado de máquina supervisionado, que precisa de dados rotulados, e aprendizado não supervisionado, que não precisa de dados rotulados, denominado aprendizado semi-supervisionado.

Os seres humanos podem identificar um cachorro utilizando vários sentidos, seja pelo cheiro, pela visão, pelo tato, e até mesmo pelo paladar. Cada um desses sentidos por si só pode ser suficiente para identificar um cachorro. Em AM, essa percepção é normalmente realizada por uma tabela atributo-valor e raramente os algoritmos utilizam mais de uma percepção ou descrição no aprendizado. Em casos em que é possível ter duas descrições diferentes dos objetos observados por meio da representação de tabela atributo-valor, é possível ter um ganho significativo no aprendizado. O algoritmo CO-TRAINING, principal tema deste trabalho, faz uso de duas descrições dos objetos (ou exemplos), além de ser um algoritmo semi-supervisionado que pode ser utilizado em bases com poucos exemplos rotulados.

Um conceito aprendido errado por um aluno pode ser um problema, um conceito aprendido errado por um professor é um problema maior ainda, pois este ensina vários alunos. Em aprendizado semi-supervisionado, um conceito aprendido errado pode ser pior que as duas coisas juntas. O aprendizado semi-supervisionado deve ser utilizado com muito cuidado, pois o algoritmo é aluno e professor ao mesmo tempo, em outras palavras, uma vez que alguns

exemplos são rotulados errados, os erros podem se propagar nas próximas iterações, degradando o modelo induzido.

Assim, em domínios nos quais erros de classificação podem ser fatais, como na área médica, os algoritmos de aprendizado semi-supervisionados devem ser utilizados com os devidos cuidados. Por outro lado, em Mineração de Texto (MT) devido aos baixos custos de classificação incorreta, alguns erros de classificação de páginas de internet, emails e textos raramente podem causar danos maiores, tornando-se um domínio no qual o aprendizado semi-supervisionado pode ser utilizado com grande benefícios. Além disso, em MT, a segunda descrição dos exemplos pode ser facilmente obtida utilizando palavras compostas

1.1 Objetivos e Principais Contribuições Desta Dissertação

Aprendizado semi-supervisionado tem atraído um número significativo de pesquisadores da área de aprendizado de máquina. A idéia de utilizar um pequeno número de exemplos rotulados, já que um número maior de exemplos rotulados é geralmente caro e difícil de se obter, e um grande número de exemplos não-rotulados, os quais encontram-se facilmente disponíveis, com o objetivo de rotular mais desses exemplos para melhorar a performance de algoritmos de AM, é uma idéia muito tentadora. O objetivo desta dissertação é realizar um estudo aprofundado do algoritmo de aprendizado semi-supervisionado CO-TRAINING, proposto por Blum & Mitchell (1998), para ser utilizado na classificação de bases de textos como parte do processo de Mineração de Textos. CO-TRAINING necessita no mínimo de duas descrições (visões) dos dados para a sua execução. Essas duas descrições, como mostrado neste trabalho, podem ser automaticamente construídas quando os dados estão relacionados à bases de textos.

Um outro objetivo, não menos importante, é o projeto e desenvolvimento da ferramenta PRETEXT para o pré-processamento de textos. Essa ferramenta é de fundamental importância para a construção das duas descrições dos dados necessários para a execução do CO-TRAINING. Tanto a implementação do algoritmo semi-supervisionado CO-TRAINING quanto a implementação da ferramenta PRETEXT, estão inseridas em um projeto maior em desenvolvimento por pesquisadores e alunos do grupo de pesquisa de Inteligência Computacional do ICMC-USP, denominado DISCOVER.

1.2 Organização da Dissertação

O restante da dissertação esta organizado da seguinte maneira: no Capítulo 2 são apresentados alguns conceitos de Aprendizado de Máquina em geral e de modo mais detalhado o aprendizado semi-supervisionado; no Capítulo 3 é descrito o algoritmo, o projeto, a implementação e os parâmetros de execução do CO-TRAINING implementado; no Capítulo 4 é apresentada a ferramenta de pré-processamento de textos PRETEXT; no Capítulo 5 são apresentados os resultados e as análise dos experimentos realizados com o objetivo de avaliar CO-TRAINING e, finalmente no Capítulo 6 são apresentadas as conclusões e os trabalhos futuros.

CAPÍTULO

2

Aprendizado de Máquina

Aprendizado de Máquina pode ser definido como uma área que pesquisa de métodos computacionais relacionados à aquisição automática de novos conhecimentos, novas habilidades e novas formas de organizar o conhecimento já existente (Mitchell, 1997). Neste capítulo são apresentados o aprendizado supervisionado, não-supervisionado e semi-supervisionado organizados na forma de uma hierarquia, bem como alguns algoritmos de aprendizado semi-supervisionado propostos na literatura.

2.1 Descrevendo Exemplos

Para viabilizar o Aprendizado de Máquina é preciso representar um mundo que nos cerca em termos computacionais. Existem diferentes linguagens de representação com diferentes complexidade e força de descrição. Essas linguagens devem ter o poder de descrever exemplos (casos observados), hipóteses (realizadas sobre essas observações) e conhecimento de domínio. Entre os diversos tipos de linguagem, uma delas, baseada na lógica de atributos, é amplamente utilizada pela maioria dos algoritmos de AM.

Nesse tipo de linguagem é utilizada um conjunto de atributos, que podem assumir diversos valores para descrever os exemplos. Cada exemplo é descrito pela disjunção ou conjunção dos valores dos atributos. Por exemplo:

$$\text{cabelo=liso} \wedge \text{pele=amarelo} \wedge \text{olhos=puxados} \wedge \text{classe=asiático}$$

Isso pode ser representado em uma linha de uma tabela atributo-valor, muito comum em bancos de dados e planilhas eletrônicas de cálculo. Esse tipo de representação é praticamente onipresente, sejam nas transações ban-

cárias, nos registros escolares, até mesmo nas compras de supermercado. Esse formato tem povoado milhares de bancos de dados gerando diariamente milhares de terabytes de dados.

No presente trabalho, a maneira utilizada para a representação de exemplos é uma tabela no formato atributo-valor. Na Tabela 2.1 é apresentado o formato geral de uma tabela atributo-valor com N exemplos E_i com $i = 1, ..., N$, na forma $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ para alguma função desconhecida $y = f(\mathbf{x})$. Os $\mathbf{x}_i$ são vetores da forma $< x_{i1}, x_{i2}, ..., x_{iM} >$, com valores discretos ou contínuos. Assim, x_{ij} refere-se ao valor do atributo (ou *feature*) j, denominado $\mathbf{X}_j$, do exemplo E_i, como mostra a Tabela 2.1. Os valores y_i referem-se ao valor do atributo $\mathbf{Y}$, que é o atributo *classe*.

A classe é o conceito-meta que descreve o fenômeno de interesse. Por exemplo, em uma base de dados médica, o conceito classe pode ser *doente* ou *não-doente*, ou então em uma base de mineração de textos o conceito classe pode ser *economia*, *esporte*, *política*, *informática* ou *ciência*.

	X_1	X_2	...	X_M	Y
E_1	x_{11}	x_{12}	⋮	x_{1M}	y_1
E_2	x_{21}	x_{22}	⋮	x_{2M}	y_2
⋮	⋮	⋮	⋱	⋮	⋮
E_N	x_{N1}	x_{N2}	⋮	x_{NM}	y_N

Tabela 2.1: Exemplos no formato atributo-valor

Os valores y_i pertencem a um conjunto C de classes C_v com $v = 1, ..., N_{Cl}$, ou seja, $C = \{C_1, ..., C_{N_{Cl}}\}$, quando se trata de *classificação* (valores discretos) ou ao conjunto de números reais (valores contínuos) no caso de *regressão*. CO-TRAINING é um algoritmo de classificação.

Dado o conjunto de exemplos, esse conjunto é normalmente dividido em outros três conjuntos denominados conjunto de treinamento, teste e validação, descritos a seguir:

- conjunto de treinamento: este conjunto é a principal entrada dos algoritmos de AM. É a partir dele que são induzidas as hipóteses, e portanto ele deve ser representativo da distribuição da população para que se possa realizar com sucesso a indução de classificadores. Na literatura esse conjunto também é conhecido como *seen cases*, que se refere aos exemplos que foram "vistos" pelo algoritmo indutor durante a construção da hipótese;

- conjunto de teste: este conjunto é utilizado para avaliar o modelo induzido. Desse modo, para que essa avaliação seja válida estatisticamente,

os exemplos contidos nesse conjunto não devem ser apresentados ao algoritmo indutor durante a construção da hipótese (*unseen cases*). Em outras palavras, esse conjunto normalmente não deve ter nenhuma intersecção com o conjunto de treinamento.

- conjunto de validação: em alguns casos, pode ser necessário utilizar exemplos para realizar ajustes no modelo induzido. Esses exemplos não são utilizados diretamente na indução do modelo, mas são utilizados na escolha da complexidade mais adequada para o modelo. Dessa maneira, esses casos são indiretamente "vistos" durante o processo de indução, o que obriga que os exemplos de validação sejam distintos dos de teste.

2.2 Hierarquia do Aprendizado

Sob algumas restrições, é possível criar máquinas que sejam capazes de aprender e melhorar por meio da observação. Existem diversas estratégias que podem ser utilizadas por um sistema computacional para realizar esse aprendizado, como aprendizado por hábito, instrução, dedução, analogia e indução. A estratégia de aprendizado por hábito é a mais simples, enquanto que nas estratégias de aprendizado por analogia e indução o aprendiz necessita de maior esforço para o aprendizado (Mitchell, 1997). O aprendizado indutivo é um dos mais utilizados entre os algoritmos de AM, e portanto, os algoritmos utilizados nesse trabalho utilizam essa estratégia.

A indução é a forma de inferência lógica que permite obter conclusões genéricas partindo de um conjunto particular de exemplos ou casos previamente observados. É caracterizado como o raciocínio que parte do específico para o geral, do particular para o universal, da parte para o todo. É importante observar que as hipóteses geradas através da inferência indutiva podem ou não preservar a verdade. Mesmo assim, a inferência indutiva é um dos principais métodos utilizados para derivar conhecimento novo e predizer eventos futuros.

O aprendizado indutivo pode ser dividido em *supervisionado* e *não-supervisionado*. No aprendizado supervisionado é fornecido ao algoritmo de aprendizado, ou *indutor*, um conjunto de exemplos de treinamento para os quais o rótulo da classe associada é conhecido. No aprendizado não-supervisionado não há classe associada aos exemplos e o indutor analisa os exemplos fornecidos e tenta determinar se alguns deles podem ser agrupados de alguma maneira, formando *agrupamentos* ou clusters (Cheeseman & Stutz, 1996).

Portanto, no Aprendizado de Máquina supervisionado, o objetivo é induzir conceitos de exemplos que estão rotulados com uma classe conhecida. Se as classes possuem valores discretos, como as diferentes marcas de carro (FORD, GM, FIAT, RENAULT, MITSUBISHI), o problema é conhecido como *classifica-*

ção. Caso as classes possuam valores contínuos, como o peso ou altura de uma pessoa, o problema é conhecido como *regressão*. Na Figura 2.1 é mostrada a hierarquia de aprendizado descrita, na qual os nós sombreados levam ao aprendizado supervisionado utilizando classificação, que é o principal modo de aprendizado utilizado neste trabalho.

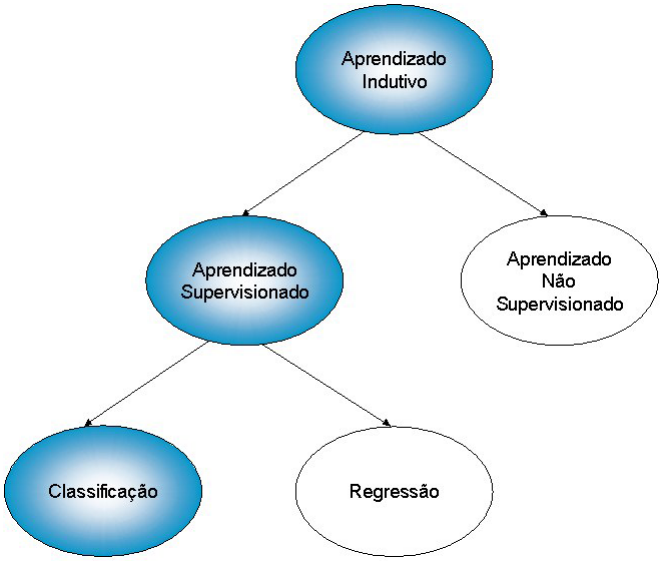


Figura 2.1: Hierarquia do aprendizado

Uma das maiores restrições do aprendizado supervisionado é a necessidade de um conjunto de exemplos com uma quantidade expressiva de exemplos rotulados para a indução de um bom classificador, o que nem sempre acontece nas bases de dados. Recentemente surgiu uma terceiro tipo de aprendizado de máquina no qual são utilizados poucos exemplos rotulados ao invés dessa quantidade expressiva utilizada no aprendizado supervisionado usual. Essa área é denominada aprendizado *semi-supervisionado*. O algoritmo CO-TRAINING, principal tema deste trabalho, é classificado como algoritmo de aprendizado semi-supervisionado.

Em resumo, os exemplos fornecidos a um algoritmo de AM podem, ou não, estar rotulados com uma classe conhecida. De acordo com esse critério, três tipos de Aprendizado de Máquina podem ser distinguidos:

- **Aprendizado de Máquina supervisionado**: utilizado quando existe um número expressivo de exemplos rotulados;
- **Aprendizado de Máquina não-supervisionado**: utilizado quando os exemplos não estão rotulados;
- **Aprendizado de Máquina semi-supervisionado**: utilizado quando existem poucos exemplos rotulados.

Portanto, a escolha de qual tipo de aprendizado (supervisionado, não-supervisionado ou semi-supervisionado) utilizar, depende muitas vezes da existência, ou da quantidade disponível de exemplos rotulados com o atributo classe. A seguir, esses tipos de AM são descritos com maiores detalhes.

2.2.1 Aprendizado Supervisionado

O processo de aprendizado supervisionado se dá pela apresentação de um conjunto de exemplos de treinamento rotulados a um indutor. A tarefa do indutor é então gerar uma hipótese (classificador), também denominada *descrição de conceito*, tal que, dado um novo exemplo não rotulado, o classificador é capaz de predizer a sua classe. O processo de classificação, ilustrado na Figura 2.2, consiste em selecionar alguns exemplos, ou ainda fornecer algumas informações adicionais, utilizando o conhecimento sobre o domínio, para o indutor. Após o classificador ser induzido, ele deve ser avaliado e se necessário o processo de classificação pode ser repetido.

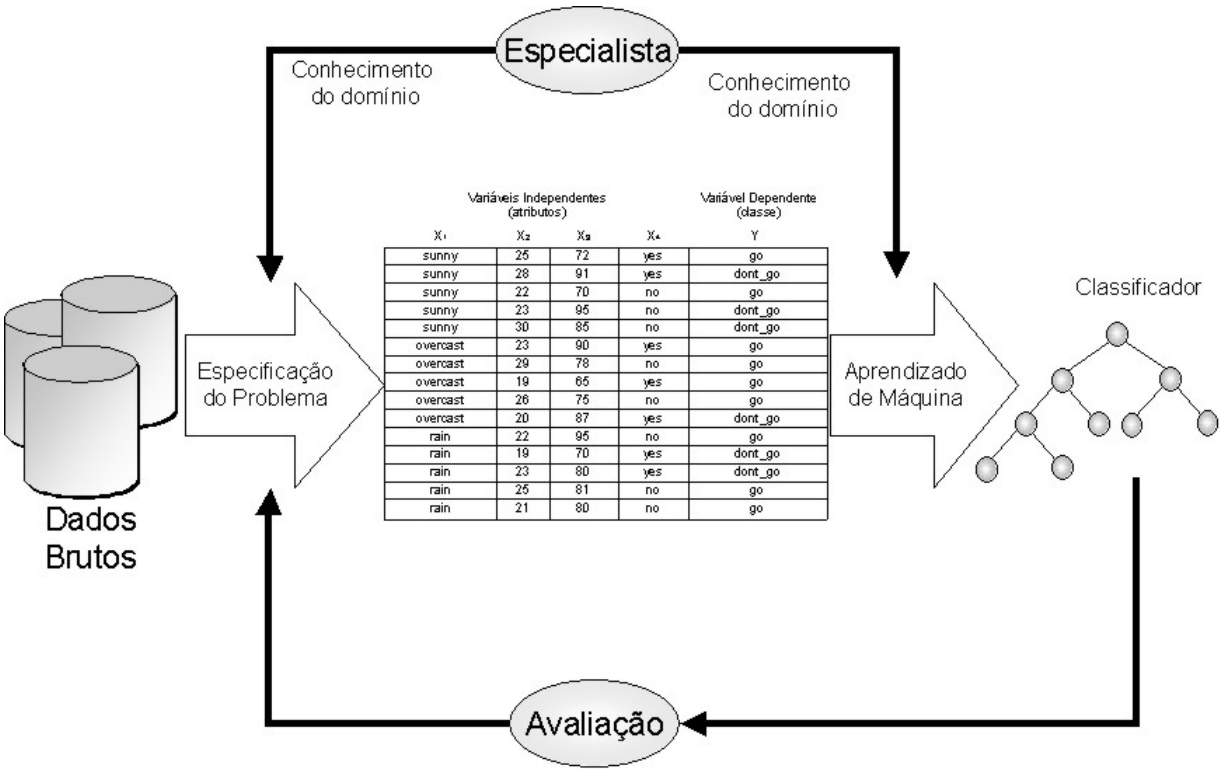


Figura 2.2: Processo de classificação (Monard & Baranauskas, 2003)

Muitas vezes, é necessário compreender o classificador (ou hipótese) induzido por um sistema de aprendizado de modo a fornecer uma explicação para o usuário. De acordo com Michalski, Carbonell, & Mitchell (1983) e Kubat, Holte, & Matwin (1998), os sistemas de aprendizado podem ser classificados em duas grandes categorias, considerando o grau de compreensibilidade proporcionado ao ser humano:

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

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

Na primeira categoria, ilustrada na Figura 2.3 (Gomes, 2002), o conceito descrito pela hipótese **h** não é facilmente interpretada por humanos, no qual a representação geralmente é dada por fórmulas matemáticas.

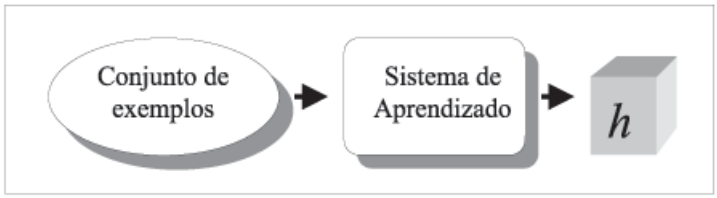


Figura 2.3: **h** vista como caixa preta

A segunda categoria é uma visão mais apurada, na qual a "caixa-preta" pode ser aberta — Figura 2.4 — e o conceito descrito pela hipótese **h** é facilmente interpretável por seres humanos. No caso ilustrado, **h** consiste de um conjunto de regras com N_R regras R_u com $u = 1, ..., N_R$, ou seja, $\mathbf{h} = \{R_1, ..., R_{N_R}\}$, denominado de *classificador simbólico*.

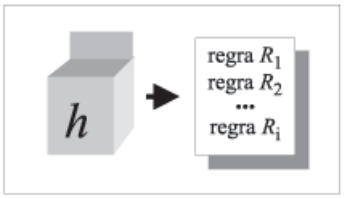
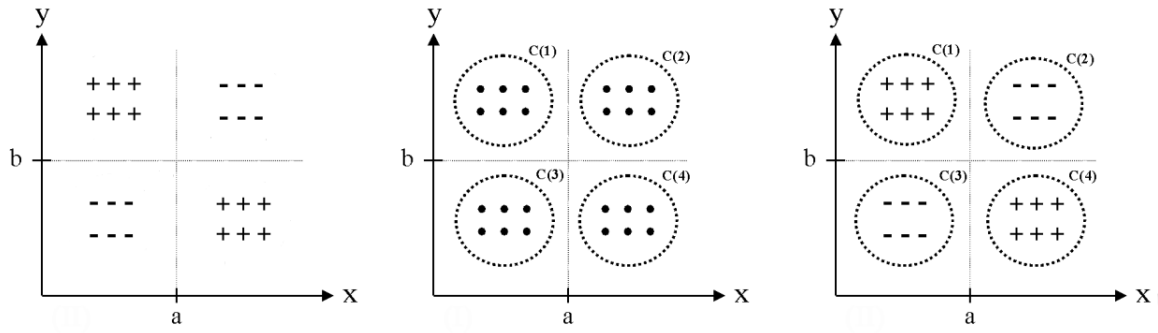


Figura 2.4: **h** vista como um conjunto de regras

2.2.2 Aprendizado Não-Supervisionado

No Aprendizado de Máquina não-supervisionado, também conhecido como aprendizado por observação e descoberta, a tarefa do algoritmo é agrupar exemplos não rotulados, *i.e.*, exemplos que não possuem o atributo classe especificado. Nesse caso, é possível utilizar algoritmos de aprendizado para descobrir padrões nos dados a partir de alguma caracterização de regularidade, sendo esses padrões denominados clusters. Exemplos contidos em um mesmo cluster são mais similares, segundo alguma medida de similaridade, do que aqueles contidos em clusters diferentes. O processo de formação dos clusters é geralmente denominado *clustering*.

É importante observar que os clusters, em AM não-supervisionado, e as classes, em AM supervisionado, não agrupam, ou representam necessariamente, os mesmos exemplos. Ainda que possível que dois ou mais clusters possam eventualmente agrupar exemplos que referem-se a uma mesma instância de um determinado conceito (a classe dos exemplos em aprendizado supervisionado), clusters e classes são diferentes, como ilustrado na Figura 2.5.



(a) (b) (c)

Figura 2.5: Clusters *versus* classes: (a) conjunto de treinamento (b) clusters encontrados (c) clusters diferentes expressando o mesmo conceito (mesma classe) (Martins, 2003)

O caso (a) ilustra um conjunto de exemplos de treinamento rotulados com a classe "$+$" e "$-$". Após submeter esses exemplos a um algoritmo de aprendizado supervisionado que induz o conceito utilizando uma árvore de decisão, por exemplo, as regras induzidas representadas graficamente na Figura 2.5 (b) seriam do tipo

if $x < a$ **and** $y < b$ **then** classe "-"
if $x \geq a$ **and** $y \geq b$ **then** classe "-"
if $x < a$ **and** $y \geq b$ **then** classe "+"
if $x \geq a$ **and** $y < b$ **then** classe "+"

Entretanto, no caso de desconhecer a classe, os exemplos de treinamento poderiam ser vistos pelo algoritmo de *clustering* como mostrado na Figura 2.5(b), *i.e.*, apenas como pontos no espaço de busca que podem ser agrupados de acordo com algum critério de similaridade. Nesse caso, o algoritmo de *clustering* encontraria 4 (quatro) clusters distintos — $C(1)$, $C(2)$, $C(3)$ e $C(4)$. Pode ser observado que esse conjunto de treinamento representa apenas duas instâncias de um determinado conceito, representadas pelas classes "$+$" (clusters 1 e 4) e "$-$" (clusters 2 e 3) na Figura 2.5 (c).

Evidentemente, seria interessante, e às vezes desejado, que todos os exemplos representando a mesma instância de um conceito, ou a mesma classe, fossem agrupados em um mesmo cluster. Infelizmente, isso não é garantido em um processo de *clustering*, já que a classe dos exemplos não é fornecida.

Para se atingir tal objetivo faz-se necessária uma interpretação, por um especialista, dos clusters encontrados, a fim de se tentar extrair o significado conceitual de cada cluster e identificar possíveis “fusões". Na Figura 2.5(c) as fusões “desejáveis" seriam entre os clusters $C(1)$ e $C(4)$ e entre $C(2)$ e $C(3)$. Entretanto, interpretar clusters não é uma tarefa trivial. De fato, isso pode ser mais difícil do que a própria geração dos clusters.

Desse modo, deve ficar claro a diferença entre cluster e classe. Enquanto o cluster é composto por exemplos similares, seguindo algum critério de semelhança, a classe é composta por exemplos que representam uma instância do mesmo conceito.

2.2.3 Aprendizado Semi-Supervisionado

Como já mencionado, o Aprendizado de Máquina supervisionado necessita de uma quantidade expressiva de exemplos rotulados, *i.e.*, exemplos de treinamento, para a indução de um bom classificador. Entretanto, obter esses exemplos rotulados nem sempre é uma tarefa trivial a qual muitas vezes pode ser um processo manual, lento e altamente custoso, envolvendo especialistas no domínio de aplicação. Assim, dependendo da aplicação, pode ser difícil, quando não impossível, encontrar bases de dados com exemplos rotulados, restringindo nesses casos, o uso de AM supervisionado. Ainda que nesses casos podem ser utilizados algoritmos de AM não-supervisionados, não é uma tarefa trivial descobrir o conceito embutido nos cluster encontrados, como mencionado anteriormente.

Porém, ainda que dado um conjunto de exemplos não-rotulados, submeter os mesmos a um especialista para ele rotular 1000 desses exemplos não seja uma tarefa factível, é possível solicitar ao especialista no domínio que rotule uns poucos exemplos, 20 ou 30, para os quais ele tem uma alta certeza sobre o rótulo associado. Nesse caso, é possível utilizar algoritmos de AM semi-supervisionados.

2.3 Algoritmos de Aprendizado Semi-Supervisionado

Um dos objetivos de aprendizado semi-supervisionado é, entre outros, rotular mais exemplos para fornecer a um algoritmo de AM supervisionado para que este, por sua vez, possa induzir melhores hipóteses.

A seguir são descritos alguns dos algoritmos de aprendizado semi-supervisionado propostos na literatura

2.3.1 COP–k-means

O algoritmo *COP–k-means* utiliza o algoritmo não-supervisionado k-means (Macqueen, 1967) com algumas modificações (Wagstaff, Cardie, Rogers, & Schroedl, 2001). A principal diferença está na utilização de conhecimento prévio (*background knowledge*), descrito na forma de relações entre os exemplos, o qual é utilizado no processo de formação dos clusters. Esse conhecimento consiste em fornecer alguns exemplos que podem ou não podem ser agrupados em um mesmo cluster, os quais determinam dois tipos de restrições:

- *Must-link*, que especifica que dois exemplos devem pertencer ao mesmo cluster;
- *Cannot-link*, que especifica que dois exemplos não devem pertencer ao mesmo cluster.

Essa abordagem é uma possível solução para o problema do aprendizado semi-supervisionado, pois permite a utilização tanto de exemplos rotulados, dos quais são extraídas as restrições, como de exemplos não rotulados.

Os clusters encontrados pelo *COP–k-means* devem respeitar todas as relações *must-link* e *cannot-link* impostas pelo usuário nos exemplos rotulados. Durante a construção dos k clusters, cada exemplo do conjunto de exemplos não rotulados é associado ao cluster mais próximo.

2.3.2 SEEDED–k-means

O algoritmo *SEEDED–k-means* (Basu, Banerjee, & Mooney, 2002), também utiliza o k-means para particionar o conjunto de dados em k clusters. A diferença mais característica é o fato do *SEEDED–k-means* utilizar exemplos inicialmente rotulados como os centróides iniciais dos clusters, *i.e.*, as sementes (*SEED*, em inglês), e não escolhê-los aleatoriamente.

Dado um conjunto de exemplos E, toma-se um subconjunto $\mathcal{S} \subset E$ como sendo o conjunto de sementes. Na inicialização do algoritmo, o usuário é responsável por atribuir cada exemplo $\mathbf{x}_i \in \mathcal{S}$ a um dos k clusters a serem encontrados, dividindo o conjunto $\mathcal{S}$ em k subconjuntos $\mathcal{S}_l$, de tal forma que $\mathcal{S} = \cup_{l=1}^{k} \mathcal{S}_l$. Uma exigência do algoritmo é que para cada cluster seja atribuído, no mínimo, uma semente. Dessa maneira, cada cluster terá o seu centróide inicial inicializado como sendo a média das sementes a ele atribuídas pelo usuário. Os próximos passos do algoritmo são idênticos ao k-means.

2.3.3 CONSTRAINED–k-means

O algoritmo *CONSTRAINED–k-means*, também proposto por Basu, Banerjee, & Mooney (2002), consiste em algumas melhorias do algoritmo *SEEDED–*

k-means. A diferença está nos passos seguintes a inicialização dos centróides, nos quais os exemplos que fazem parte do conjunto de sementes, e que foram inicialmente associados a um dado cluster pelo usuário, não poderão ser associados a um outro cluster. Dessa maneira, apenas os exemplos não selecionados como sementes terão o cluster correspondente reestimado. Ao passo que no *SEEDED*–k-means as sementes podem vir a pertencer a clusters diferentes daqueles inicialmente associados, no *CONSTRAINED*–k-means isso não ocorre. Assim, o *CONSTRAINED*–k-means é mais adequado quando as sementes, relacionadas aos exemplos rotulados, estão livres de ruídos. Caso contrário, recomenda-se o uso do *SEEDED*–k-means.

2.3.4 k-*means*$_{ki}$

O algoritmo k-*means*$_{ki}$, proposto por Sanches (2003), também se baseia no algoritmo k-means. Sua principal diferença está na fase de seleção de centróides. O algoritmo k-means seleciona aleatoriamente seus centróides, no k-*means*$_{ki}$ os centróides iniciais são os exemplos iniciais (rotulados) de treinamento. A idéia é dificultar que exemplos de diferentes classes sejam agrupados no mesmo cluster.

Uma segunda diferença está no processo de *clustering*. Quando o k-means é utilizado, cada exemplo é associado ao cluster (centróide) mais próximo. No caso do k-*means*$_{ki}$, é estipulado a princípio um limiar *t*. O exemplo somente poderá ser associado a um dado cluster caso esteja a uma distância menor ou igual a *t* de seu respectivo centróide. Em resumo, o algoritmo é composto basicamente por duas etapas:

1. *Clustering*: consiste em encontrar clusters utilizando apenas os poucos exemplos rotulados. Este é um ponto muito importante, pois o rótulo dos exemplos não tem significado especial em AM não supervisionado. Como já mencionado, clusters em AM não-supervisionado e classes em AM supervisionado, não são, necessariamente, equivalentes. Assim, a idéia do algoritmo k-*means*$_{ki}$ é atribuir ao atributo classe um alto peso na medida de similaridade utilizada, tal que esses clusters agrupem exemplos com o mesmo valor do atributo classe.

2. Rotulação: consiste em analisar o conjunto de exemplos não rotulados com o intuito de determinar os exemplos que pertencem aos clusters encontrados, mas com um *alto grau de similaridade*. Uma vez escolhidos esses exemplos, eles serão rotulados com a classe dos exemplos do cluster correspondente, incrementando assim o conjunto de exemplos rotulados.

2.3.5 Outros Algoritmos

Existem algoritmos de AM semi-supervisionados que são variações do algoritmo não-supervisionado *Expectation Maximization* (EM) (Dempster, Laird, & Rubin, 1977), como o algoritmo apresentado em (Bruce, 2001), no qual os passos E e M são incrementados com uma variável denominada de pseudo-contador. Com essa alteração é possível estimar o rótulo dos dados não-rotulados.

Ainda utilizando EM, em (Nigam, McCallum, Thrun, & Mitchell, 2000) é apresentado uma extensão do algoritmo EM com *Naive Bayes* (Mitchell, 1997) que utiliza máxima probabilidade ou estimação máxima posterior de problemas com dados incompletos. No caso desse algoritmo, os dados não-rotulados são tratados como se fossem dados incompletos. O algoritmo *Naive Bayes* induz o classificador apenas com os dados rotulados. Como EM também é um algoritmo bayesiano, no passo E é inserido o classificador *Naive Bayes*, calculado anteriormente, para estimar algumas probabilidades, e no passo M os cálculos são realizados como no algoritmo EM básico.

Um outro algoritmo, baseado em ensemble é ASSEMBLE (Adaptative Semi Supervise ensEMBLE) (Bennett & Demiriz, 2002). Usando árvores da decisão como classificador, esse algoritmo foi o primeiro de 34 algoritmos apresentados durante a competição de algoritmos semi supervisionados NIPS 2001 (Kremer & Stacey, 2001).

Também foram propostos algoritmos semi-supervisionados que utilizam *Support Vector Machines* (Cortes & Vapnik, 1995) com algumas alterações (Bennett & Demiriz, 1998; Fung & Mangasarian, 1999; Seeger, 2002). Além desses métodos, existem alguns outros que utilizam desde aprendizado por reforço até cortes em gráficos (Blum & Chawla, 2001; Ivanov, Blumberg, & PentLand, 2001).

2.4 Considerações Finais

A maioria dos algoritmos de aprendizado semi-supervisionado propostos consistem de uma variação dos algoritmos de aprendizado de máquina tradicionais. Muitos desses algoritmos tentam obter, de alguma maneira, um ganho sobre os poucos exemplos rotulados. Normalmente, esse ganho é obtido com algum método de escolha dos exemplos a serem rotulados. Esse método é de vital importância para esses algoritmos semi-supervisionados, pois, uma vez que um "erro" for introduzido ao rotular um novo exemplo, esse erro pode se propagar nas próximas iterações, acumulando-se e tendo como conseqüência um aumentando no erro do classificador induzido por um algoritmo de AM supervisionado utilizando os exemplos rotulados pelo algoritmo

semi-supervisionado.

No Capítulo 3 é apresentado o algoritmo CO-TRAINING, um dos principais tema deste trabalho. Uma importante característica é o método de escolha dos exemplos a serem rotulados, diferente dos métodos utilizados pelos algoritmos apresentados neste capítulo.

CAPÍTULO

3

O Algoritmo CO-TRAINING

No mundo real existem inúmeras bases de dados (exemplos) das quais há interesse em extrair se conhecimento. Porém, se torna um problema quando muitos desses exemplos tenham que ser rotulados manualmente por um especialista de domínio, com o objetivo de induzir, classificadores precisos utilizando algoritmos de aprendizado supervisionados. Entretanto, solicitar ao especialista que rotule alguns poucos exemplos protótipos é uma tarefa factível e, após, esses poucos exemplos podem ser submetidos a um algoritmo de aprendizado semi-supervisionado, para assim incrementar o número de exemplos rotulados. Neste capítulo é descrito o algoritmo de aprendizado semi-supervisionado CO-TRAINING (Blum & Mitchell, 1998), tema deste trabalho, bem como a sua implementação.

3.1 Uma Visão Geral do Algoritmo

Para ilustrar a idéia de CO-TRAINING, pode ser utilizada a seguinte analogia: o ser humano tem a capacidade de descrever objetos, ou situações observadas, de formas diferentes. Dadas, por exemplo, as duas descrições que descrevem a mesma situação, o ser humano é capaz de aprender o mesmo conceito, mas induzindo duas hipóteses diferentes, cada uma delas consistente com uma dessas descrições da mesma situação do mundo real. Após o aprendizado, as duas descrições de um novo exemplo podem ser fornecidas para as respectivas hipóteses induzidas, e elas serão capazes de classificar com alta probabilidade cada uma dessas duas descrições com o mesmo rótulo da classe associada. O método de CO-TRAINING baseia-se nessa idéia para tentar rotular mais exemplos com o objetivo de melhorar a performance de

AM supervisionado, quando o número original de exemplos rotulados não é suficiente para um bom aprendizado. Em outras palavras, CO-TRAINING, utilizando duas descrições de mundo, usa dois algoritmos de AM supervisionado, ou o mesmo algoritmo, para induzir duas hipóteses (classificadores). Cada hipótese é induzida utilizando como conjunto de treinamento o conjunto de exemplos que representa uma dessas descrições de mundo. Dessa maneira, tem-se duas hipóteses sobre a mesma situação, mas cada uma delas induzida sobre os mesmos exemplos mas descritos segundo uma visão diferente.

3.1.1 Criação das Duas Descrições

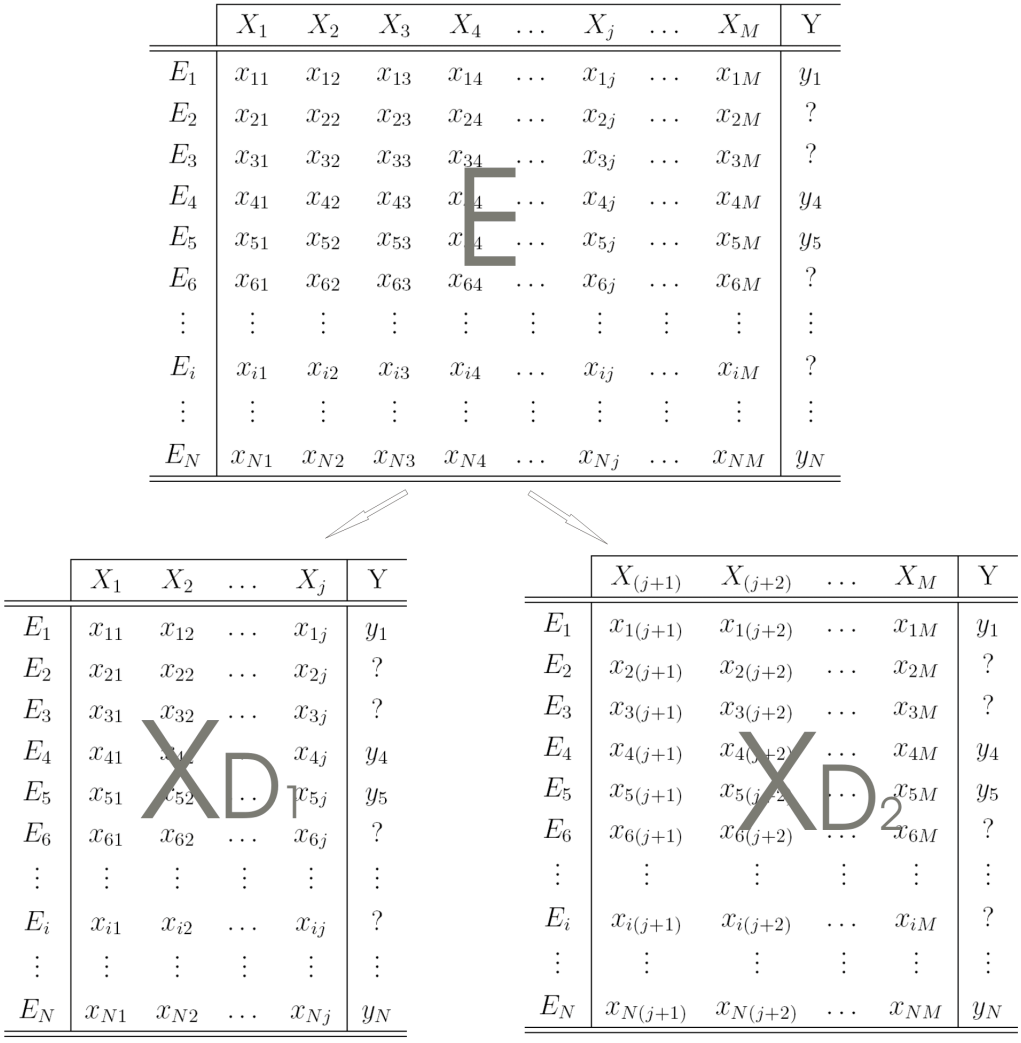
Para criar as duas descrições usadas por CO-TRAINING é necessário particionar o conjunto de atributos $\mathbf{X}$, que descreve os exemplos, em dois subconjuntos, $\mathbf{X}_{D_1}$ e $\mathbf{X}_{D_2}$, que constituem as duas descrições da situação e descrevem os mesmos exemplos tal que $\mathbf{X} = \mathbf{X}_{D_1} \cup \mathbf{X}_{D_2}$ e $\mathbf{X}_{D_1} \cap \mathbf{X}_{D_2} = \emptyset$ como ilustra a Figura 3.1 na próxima página, na qual, por simplicidade, é considerado que $\mathbf{X}_{D_1} = \{X_1, X_2, ..., X_j\}$ e $\mathbf{X}_{D_2} = \{X_{j+1}, X_{j+2}, ..., X_M\}$. Exemplos com valor de Y igual a "?" representam exemplos não rotulados.

Além da separação em duas descrições, o conjunto de exemplos $E = \{E_1, ..., E_N\}$ deve ser separado em dois subconjuntos L e U, também conhecidos como conjunto de exemplos rotulados (*Labeled*) e não-rotulados (*Unlabeled*) respectivamente. O subconjunto $L \subset E$ que contém exemplos que possuem o atributo classe conhecido é, por sua vez, dividido em duas descrições L_{D_1} e L_{D_2}, tal que $L = L_{D_1} \cup L_{D_2}$ e $L_{D_1} \cap L_{D_2} = \emptyset$. Analogamente, o subconjunto $U \subset E$ que contém os exemplos nos quais o atributo classe é desconhecido é também dividido em duas descrições U_{D_1} e U_{D_2}, tal que $U = U_{D_1} \cup U_{D_2}$ e $U_{D_1} \cap U_{D_2} = \emptyset$. Os dados de entrada do algoritmo de CO-TRAINING consistem desses quatro conjuntos de exemplos L_{D_1}, L_{D_2}, U_{D_1} e U_{D_2}. Na Figura 3.2 encontram-se ilustrados esses quatro conjuntos de exemplos.

3.1.2 Descrição do Algoritmo

No Algoritmo 1 na página 23 são descritos os principais passos de CO-TRAINING comentados a seguir.

No primeiro passo do algoritmo antes do laço, são criados dois subconjuntos U'_{D_1} e U'_{D_2}, tal que $U' = U'_{D_1} \cup U'_{D_2}$ e $U'_{D_1} \cap U'_{D_2} = \emptyset$, sendo U' um subconjunto de, geralmente, poucos exemplos de U, ou seja, esses dois subconjuntos de exemplos não rotulados U'_{D_1} e U'_{D_2} são subconjuntos de U_{D_1} e U_{D_2} respectivamente. Os exemplos que compõem U'_{D_1} e U'_{D_2} são removidos de U_{D_1} e U_{D_2} a fim de verificar as condições $U'_{D_1} \cap U_{D_1} = \emptyset$ e $U'_{D_2} \cap U_{D_2} = \emptyset$ criando assim conjuntos disjuntos. Após cada iteração com os conjuntos de exemplos rotulados L_{D_1}



	X_1	X_2	X_3	X_4	...	X_j	...	X_M	Y
E_1	x_{11}	x_{12}	x_{13}	x_{14}	...	x_{1j}	...	x_{1M}	y_1
E_2	x_{21}	x_{22}	x_{23}	x_{24}	...	x_{2j}	...	x_{2M}	?
E_3	x_{31}	x_{32}	x_{33}	x_{34}	...	x_{3j}	...	x_{3M}	?
E_4	x_{41}	x_{42}	x_{43}	x_{44}	...	x_{4j}	...	x_{4M}	y_4
E_5	x_{51}	x_{52}	x_{53}	x_{54}	...	x_{5j}	...	x_{5M}	y_5
E_6	x_{61}	x_{62}	x_{63}	x_{64}	...	x_{6j}	...	x_{6M}	?
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
E_i	x_{i1}	x_{i2}	x_{i3}	x_{i4}	...	x_{ij}	...	x_{iM}	?
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
E_N	x_{N1}	x_{N2}	x_{N3}	x_{N4}	...	x_{Nj}	...	x_{NM}	y_N

	X_1	X_2	...	X_j	Y
E_1	x_{11}	x_{12}	...	x_{1j}	y_1
E_2	x_{21}	x_{22}	...	x_{2j}	?
E_3	x_{31}	x_{32}	...	x_{3j}	?
E_4	x_{41}	x_{42}	...	x_{4j}	y_4
E_5	x_{51}	x_{52}	...	x_{5j}	y_5
E_6	x_{61}	x_{62}	...	x_{6j}	?
⋮	⋮	⋮	⋮	⋮	⋮
E_i	x_{i1}	x_{i2}	...	x_{ij}	?
⋮	⋮	⋮	⋮	⋮	⋮
E_N	x_{N1}	x_{N2}	...	x_{Nj}	y_N

	$X_{(j+1)}$	$X_{(j+2)}$	...	X_M	Y
E_1	$x_{1(j+1)}$	$x_{1(j+2)}$	...	x_{1M}	y_1
E_2	$x_{2(j+1)}$	$x_{2(j+2)}$	...	x_{2M}	?
E_3	$x_{3(j+1)}$	$x_{3(j+2)}$	...	x_{3M}	?
E_4	$x_{4(j+1)}$	$x_{4(j+2)}$	...	x_{4M}	y_4
E_5	$x_{5(j+1)}$	$x_{5(j+2)}$	...	x_{5M}	y_5
E_6	$x_{6(j+1)}$	$x_{6(j+2)}$	...	x_{6M}	?
⋮	⋮	⋮	⋮	⋮	⋮
E_i	$x_{i(j+1)}$	$x_{i(j+2)}$	...	x_{iM}	?
⋮	⋮	⋮	⋮	⋮	⋮
E_N	$x_{N(j+1)}$	$x_{N(j+2)}$	...	x_{NM}	y_N

Figura 3.1: Duas descrições do conjunto de exemplos

e L_{D_2} são induzidos, respectivamente, dois classificadores h_{D_1} e h_{D_2} os quais são utilizados para rotular exemplos não rotulados de U'_{D1} e U'_{D2} respectivamente. No fim deste processo, R'_{D_1} e R'_{D_2} contém, respectivamente, o conjunto de exemplos rotulados nesse passo da iteração, os quais são argumentos da função *melhores/2* que é responsável pela seleção de exemplos que foram "melhor" rotulados, e com o mesmo rótulo, pelos respectivos classificadores h_{D_1} e h_{D_2}. A função *melhores/2* pode ser considerada como a mais importante na implementação do algoritmo CO-TRAINING, e é descrita em maiores detalhes na Seção 3.2.3 na página 30 (parâmetro `-lsm`). Após a execução da função *melhores/2*, os exemplos que foram "melhor" rotulados são adicionados, respectivamente, aos conjuntos de exemplos de treinamento L_{D_1} e L_{D_2}. No caso de ainda existirem exemplos não rotulados, são utilizados mais exemplos ainda não rotulados em U_{D1} e U_{D2} os quais são adicionados, respectivamente, aos conjuntos U'_{D1} e U'_{D2} e o processo é repetido.

Ainda que a saída do algoritmo consista do conjunto de exemplos rotulados por CO-TRAINING, para serem utilizados por qualquer algoritmo de AM

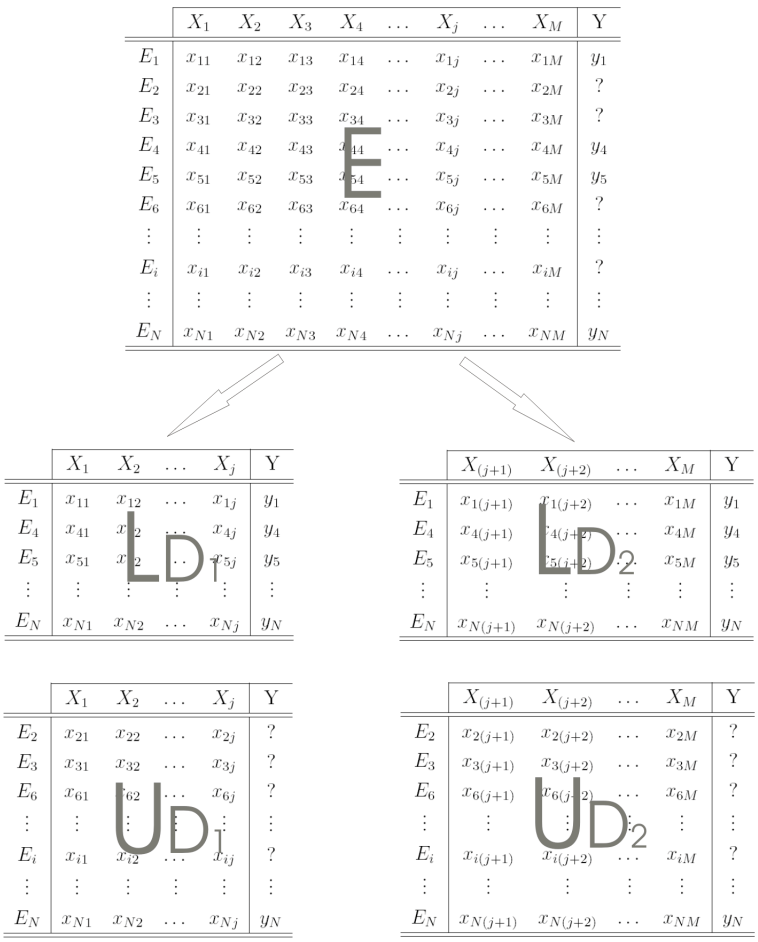


Figura 3.2: Conjunto de exemplos L_{D_1}, L_{D_2} U_{D_1} e U_{D_2}

supervisionado. No artigo original do método (Blum & Mitchell, 1998) são também considerados, em cada iteração, dois classificadores h_{D_1} e h_{D_2} para criar um terceiro classificador baseado nos dois primeiros, denominado classificador combinado. Esse classificador utiliza uma particularidade do algoritmo $NaiveBayes$ que fornece, além da classe, a probabilidade do exemplo pertencer a cada uma das classes.

O classificador combinado, descrito pelo Algoritmo 2 calcula a probabilidade $P(C_v|E_i)$ da classe C_v, onde $v = 1, \ldots N_{Cl}$ e N_{Cl} é o número de classes possíveis e E_i é o exemplo a ser rotulado. No entanto, os dois classificadores h_{D_1} e h_{D_2} fornecem apenas as probabilidades respectivas as suas descrições. Para calcular a probabilidade $P(C_v|E_i)$ considerando ambos os classificadores como se fosse um único classificador, cada exemplo E_i pode ser dado como $(\mathbf{D1}_i \cup \mathbf{D2}_i)$ no qual $\mathbf{D1}_i$ e $\mathbf{D2}_i$ correspondem aos valores dos atributos da primeira e segunda descrição, respectivamente, do exemplo E_i. Portanto os dois classificadores, h_{D_1} e h_{D_2}, fornecem $P(C_v|\mathbf{D1}_i)$ e $P(C_v|\mathbf{D2}_i)$ ou seja, a probabilidade do exemplo E_i ser da classe C_v dado pelo classificador induzido a partir de L_{D_1} e a probabilidade de ser da classe C_v dado pelo classificador induzido a partir de L_{D_2}. Como v varia de 1 a N_{Cl}, os classificadores fornecem as pro-

Algoritmo 1: Cotraining

Input: $L_{D_1}, L_{D_2}, U_{D_1}, U_{D_2}, k$

Output: L_{D_1}, L_{D_2}

Construir U'_{D_1} e U'_{D_2} como descrito anteriormente;
$U_{D_1} = U_{D_1} - U'_{D_1}$;
$U_{D_2} = U_{D_2} - U'_{D_2}$;
for $i = 0$ *to* k **do**
    induzir o classificador h_{D_1} a partir dos exemplos de treinamento L_{D_1};
    induzir o classificador h_{D_2} a partir dos exemplos de treinamento L_{D_2};
    R'_{D_1} = todos os exemplos de U'_{D_1} rotulados com o classificador h_{D_1};
    R'_{D_2} = todos os exemplos de U'_{D_2} rotulados com o classificador h_{D_2};
    $(R_{D_1}, R_{D_2}) = melhores(R'_{D_1}, R'_{D_2})$;
    if $R_{D_1} = \emptyset$ **then Return** L_{D_1}, L_{D_2} ;
    $L_{D_1} = L_{D_1} \cup R_{D_1}$;
    $L_{D_2} = L_{D_2} \cup R_{D_2}$;
    if $U_{D_1} = \emptyset$ **then Return** L_{D_1}, L_{D_2} **else**
        retirar aleatoriamente alguns exemplos de U_{D_1} e os exemplos respectivos de U_{D_2} e adicioná-los a U'_{D_1} e U'_{D_2} respectivamente;
    end
end
Return L_{D_1}, L_{D_2};

babilidades do exemplo pertencer a todas as N_{Cl} classes. Para obter $P(C_v|E_i)$ são multiplicadas as probabilidades $P(C_v|\mathbf{D1}_i)$ e $P(C_v|\mathbf{D2}_i)$ calculadas por h_{D_1} e h_{D_2} para cada classe como mostrado no Algoritmo 2.

Algoritmo 2: Classificador Combinado

for $i = 1$ *to* N **do**
    for $k = 1$ *to* N_{Cl} **do**
        $P(C_v|E_i) \leftarrow P(C_v|\mathbf{D1}_i)P(C_v|\mathbf{D2}_i)$;
    end
end

Esse terceiro classificador permite representar CO-TRAINING como um único classificador, possibilitando assim comparações com outros algoritmos.

3.2 Projeto e Implementação do Algoritmo

O algoritmo CO-TRAINING implementado neste trabalho foi desenvolvido em Perl (Wall, Christiansen, & Schwartz, 1996) utilizando a biblioteca de classes *Discover Object Library* (DOL) (Batista & Monard, 2003). A DOL faz parte de um projeto maior, denominado DISCOVER (Baranauskas & Batista, 2000), em desenvolvimento no Laboratório de Inteligência Computacional (Labic)[1]. O DISCOVER tem como objetivo fornecer um ambiente integrado para apoiar

[1] `http://labic.icmc.usp.br`

todas as etapas do processo de descoberta de conhecimento[2], oferecendo funcionalidades voltadas para o aprendizado de máquina, mineração de dados e mineração de textos. De uma maneira geral, o sistema DISCOVER pode ser entendido como um conjunto de métodos que são aplicados sobre os dados ou sobre o conhecimento extraído a partir dos dados. Assim, é muito importante que o ambiente DISCOVER ofereça uma base sólida para manipular dados e conhecimento. Essa base é composta por sintaxes padrões para a representação de dados e de conhecimento, e por bibliotecas que oferecem um conjunto de funcionalidades básicas de manipulação de dados e conhecimento (Batista & Monard, 2003; Martins, 2003; Prati, Baranauskas, & Monard, 2001b,a; Melanda & Rezende, 2003; Pugliesi, 2004), o sistema terá uma interface gráfica (Geromini, 2002) e um *framework* de integração (Prati, 2003; Baranauskas, 2001; Milaré, 2003). Atualmente, existem definidas sintaxes padrões para a representação de dados e para a representação do conhecimento induzido de diversos algoritmos de aprendizado de máquina, bem como bibliotecas que oferecem diversas funcionalidades sobre essas sintaxes padrão. Entre as diversas funcionalidades da biblioteca de classes DOL, pode-se citar: a manipulação de atributos e dados utilizados por sistemas de aprendizado bem como a integração de vários sistemas de aprendizado implementados pela comunidade e pelo nosso grupo de pesquisa; integração com sistemas gerenciadores de bases de dados; filtros para ocultar temporariamente subconjuntos de exemplos e atributos de um conjunto de dados; estatísticas e correlações e métodos de re-amostragem de dados, entre outros.

A integração do CO-TRAINING no projeto DISCOVER, por meio de uso da DOL, representa um ganho importante tanto na implementação do CO-TRAINING quanto na implementação de qualquer outro sistema que faça uso dos formatos padrões e das facilidade já implementadas no DISCOVER. Isso deve-se ao fato dessas bibliotecas estarem bem implementadas, além de já terem sido testadas e validadas pelos integrantes do nosso grupo de pesquisa e usuários externos.

Para utilizar a DOL é necessário que o conjunto de exemplos, no formato atributo-valor, obedeça a sintaxe padrão do sistema DISCOVER, a qual é denominada *Discover Standard Sintax* (DSX). Durante a execução, a DOL converte o conjunto desses exemplos, os quais encontram-se armazenados em arquivos texto, para a sintaxe padrão do algoritmo de aprendizado indicado pelo usuário. A DOL é responsável pela execução do algoritmo de aprendizado e pelo retorno dos resultados obtidos, além de calcular várias estatísticas. Portanto, todos os conjuntos de exemplos utilizados pelo CO-TRAINING devem estar na DSX, assim como os conjuntos de exemplos rotulados pelo algoritmo.

[2]KDD– Knowledge Discovery in Databases.

Além da implementação do algoritmo CO-TRAINING como parte integrante do DISCOVER, para ser executado em bases de dados reais, com poucos exemplos rotulados e muitos exemplos não-rotulados é também importante pesquisar o comportamento do algoritmo avaliando os resultados obtidos em cada iteração bem como a influência dos valores de diversos parâmetros implementados para a sua execução. Assim, na implementação realizada, o CO-TRAINING pode ser executado em dois modos:

1. modo simulação: o modo simulação foi implementado para facilitar a avaliação do algoritmo de CO-TRAINING, permitindo verificar, passo a passo, o seu comportamento. Neste modo, o conjunto de exemplos não rotulados U é criado a partir de um conjunto de exemplos rotulados ignorando o atributo classe. Dessa maneira, é possível verificar quais e quantos exemplos o algoritmo está rotulando "errado", e em quais iterações isso está ocorrendo e várias outras informações.

2. modo real: o modo real foi construído para executar bases do mundo real nas quais tem-se uma grande quantidade de exemplos não-rotulados e uma pequena quantidade de exemplos rotulados. O modo real é ativado automaticamente pelo programa, caso seja encontrado algum valor do atributo classe dos exemplos igual a "?" que indica exemplos não rotulados. Ao encontrar exemplos com o valor do atributo classe igual a "?", esses exemplos são colocados no conjunto de exemplos não rotulados U e todos os exemplos com o atributo classe diferente de "?" são colocados no conjunto de exemplos rotulados L.

Na Tabela 3.1 é ilustrada uma base com essas características. Nessa tabela os exemplos E_1, E_5, E_7 e E_9 irão compor o conjunto de exemplos rotulados L e os exemplos E_2, E_3, E_4, E_6 e E_8 irão compor o conjunto de exemplos não rotulados U.

	X_1	X_2	...	X_M	Y
E_1	x_{11}	x_{12}	...	x_{1M}	y_1
E_2	x_{21}	x_{22}	...	x_{2M}	?
E_3	x_{31}	x_{32}	...	x_{3M}	?
E_4	x_{41}	x_{42}	...	x_{4M}	?
E_5	x_{51}	x_{52}	...	x_{5M}	y_5
E_6	x_{61}	x_{62}	...	x_{6M}	?
E_7	x_{71}	x_{72}	...	x_{7M}	y_7
E_8	x_{81}	x_{82}	...	x_{8M}	?
E_9	x_{91}	x_{92}	...	x_{9M}	y_9

Tabela 3.1: Exemplos rotulados e não-rotulados na mesma base

```
S ::= <diretorio>/d<numero>.<algoritmo>/<arquivos-dsx>

<diretorio> ::= <identificador>

<algoritmo> ::= nb|svm

<arquivo-dsx> ::= <identificador>.data
                 |<identificador>.names
                 |<identificador>.test
```

Figura 3.3: Sintaxe da linguagem para definir os nomes dos diretórios e arquivos

3.2.1 Dados de Entrada

A entrada para a execução do algoritmo é o nome de um diretório que contém as bases de dados necessárias. A implementação percorre esse diretório procurando pelas bases de dados. Assim, esse diretório deve ser definido seguindo alguns padrões para que o algoritmo possa executar corretamente.

No modo simulação, o usuário deve fornecer como entrada o nome do diretório que contém, pelo menos, dois subdiretórios cujos nomes especificam o algoritmo de aprendizado a ser utilizado no conjunto de exemplos (descrição) nele definidos. Cada um desses subdiretórios deve conter uma das visões do conjunto de exemplos, na sintaxe DSX. A sintaxe da linguagem que deve ser utilizada para nomear esses diretórios e arquivos é mostrada na Figura 3.3, onde `<identificador>` é qualquer seqüência de letras, números ou *underscores* (_) e `<algoritmos>` é o nome do algoritmo de Aprendizado de Máquina a ser utilizado por CO-TRAINING para rotular exemplos da descrição `d<número>` correspondente. A atual implementação utiliza o algoritmo *Naive Bayes* (`nb`), e futuramente será considerados o algoritmo *Support Vector Machines* (`svm`) (Cortes & Vapnik, 1995).

Por exemplo, suponha que se deseja executar uma simulação com uma descrição denominada `oneGram` e a outra `twoGram`[3]. Então, ambos conjuntos de exemplos devem estar definidos nos respectivos arquivos `.names` e `.data` na sintaxe DSX. Sejam esses arquivos `oneGram.names` e `oneGram.data` para a primeira visão, `twoGram.names` e `twoGram.data` para a segunda visão. Considere que na primeira visão é desejado executar o algoritmo *Naive Bayes* o qual é representado pelo identificador `nb` e na segunda visão é desejado executar o algoritmo *Support Vector Machines* (SVM) representado pelo identificador `svm`. Assim, para criar os diretórios para executar CO-TRAINING com esses conjuntos de dados e algoritmos, deve ser criada pelo usuário a estrutura de diretórios e arquivos descrita na Figura 3.4.

[3]No Capítulo 4 é discutido em detalhes o procedimento para a criação dessas descrições.

```
baseLNAI/
         d1.nb/
               oneGram.data
               oneGram.names
         d2.svm/
               twoGram.data
               twoGram.names
```

Figura 3.4: Exemplo de estrutura de diretórios de entrada

Nessa figura, o diretório `baseLNAI` contém dois subdiretórios `d1.nb` e `d2.svm`. Esses dois subdiretórios indicam que os arquivos neles contidos representam, respectivamente, as duas descrições (visões) do conjunto de exemplos a ser utilizado por CO-TRAINING e também indicam que o algoritmo *Naive Bayes* deve ser executado na primeira visão e SVM na segunda visão do conjunto de dados. Ou seja, nesses dois subdiretórios `d1.nb` e `d2.svm`, devem ser encontrados os respectivos arquivos `.data` e `.names` que definem, na sintaxe DSX, o conjunto de exemplos para cada descrição. No exemplo na Figura 3.4, esses arquivos são: `oneGram.data` e `oneGram.names` para o conjunto de exemplos da descrição 1 e `twoGram.data` e `twoGram.names` para o conjunto de exemplos da descrição 2.

Deve ser observado que o algoritmo necessita de, no mínimo, duas descrições do conjunto de exemplos, mas poderia utilizar mais de duas descrições. A implementação realizada suporta mais de duas descrições. Nesse caso, o usuário deve definir os diretórios que contém cada uma dessas descrições. Supondo que o usuário quer executar CO-TRAINING utilizando 4 (quatro) descrições, então deve definir os diretórios correspondentes, por exemplo `d3.nb` e `d4.svm`, e os respectivos arquivos em cada diretório.

Na implementação realizada, antes da execução do CO-TRAINING, existe uma etapa prévia que realiza algumas verificações nos conjuntos de exemplos fornecidos pelo usuário. Essa etapa consiste na verificação dos arquivos estarem nos seus devidos diretórios, se o nomes dos diretórios e arquivos estão corretos, se os exemplos e rótulos estão em suas devidas posições em todas as bases e, no final dessa etapa, as bases de exemplos rotulados L, não rotulados U e a base de teste são criadas em memória. Detalhes sobre todas as verificações realizadas encontram-se em (Matsubara & Monard, 2004b).

Após essa etapa, a execução do algoritmo CO-TRAINING começa.

3.2.2 Alocação em Memória dos Conjuntos de Exemplos

Com os arquivos que compõem as bases devidamente verificados, essas bases podem ser manipuladas para a construção dos conjuntos de exemplos requeridos por CO-TRAINING — Algoritmo 1 na página 23. Inicialmente é criada

ou carregada em memória a base de teste. O programa procura por essa base de teste, ou seja, verifica se existe um arquivo com extensão `.test`, definido pelo usuário, no subdiretório correspondente. Caso essa base de teste não seja encontrada, ela é criada a partir de uma porcentagem do conjunto de exemplos original. Essa porcentagem pode ser especificada pelo usuário. Após a base de teste devidamente alocada em memória, o programa cria as bases de exemplos rotulados L e não rotulados U. Nessa fase, são construídas as bases L e U somente em memória, ou seja, não são criados os arquivos `.test`, `.data` e `.names`. Somente no final da execução o programa grava todas as bases. A seguir é apresentado uma descrição de cada uma dessas bases.

base de teste: a base que contém os exemplos de teste pode ser criada de duas maneiras: por meio de um arquivo `.test` ou, no modo simulação, por uma porcentagem do conjunto de exemplos original, como explicado a seguir.

1. *arquivo de teste*: o usuário pode fornecer uma base de teste que deverá ter a extensão `.test`. Os exemplos contidos nesse arquivo devem estar no mesmo formato dos exemplos do arquivo `.data`.
2. *uma porcentagem do conjunto de exemplos original*: caso não seja fornecida uma base de teste, no modo simulação, o programa detecta a falta dessa base e cria uma nova base de teste a partir de uma porcentagem do conjunto de exemplos original. Essa porcentagem pode ser definida pelo parâmetro `-t porcentagem`. Para exemplificar, suponha que o CO-TRAINING seja executado com uma base de 1000 exemplos invocando a seguinte linha de execução:

   ```
   cotraining.pl -t 0.3 baseLNAI
   ```

   o parâmetro `-t` definido como `0.3` indica 30% da base original. Portanto, a base de exemplos de teste será composta de 30% dos 1000 exemplos, ou seja 300 exemplos. Os 700 exemplos restantes irão compor o que denominamos base parcial.
   Valor padrão : 0.25;

base de exemplos rotulados L e não rotulados U: no modo simulação, essas bases podem ser criadas de duas maneiras: indicando uma porcentagem da base parcial[4], ou fornecendo uma lista de exemplos. Essas duas maneiras são definidas pelos seguintes parâmetros:

[4]A base parcial e a base de dados original coincidem caso o usuário fornecer a base de teste. Caso contrário, a base parcial é obtida após retirar o subconjunto de exemplos de teste da base de dados original

-l <porcentagem>: define uma porcentagem sobre a base parcial, construída na criação da base de teste, que irá compor a base de exemplos rotulados L. Por exemplo, se o CO-TRAINING for executado em uma base com 100 exemplos invocando a seguinte linha de execução:

```
cotraining.pl -l 0.3 baseLNAI
```

na qual o parâmetro `-l` é definido como `0.3` então L consiste de 30% dos exemplos da base parcial.

A base de teste sempre é alocada antes das bases de exemplos rotulados e não rotulados. Assim, no caso do usuário utilizar o valor padrão para `-t` e uma base original com 1000 exemplos, inicialmente é construída uma base de teste com 25% (parâmetro padrão) de 1000 exemplos, ou seja, uma base de teste com 250 exemplos e a base parcial com 750 exemplos. Dessa base parcial são retirados 30% dos 750 exemplos, ou seja, 225 exemplos para comporem a base de exemplos rotulados L e os restantes, 525 (750-225 = 525) exemplos, constituem a base de exemplos não rotulados U. Resumidamente, utilizando uma base original com 1000 exemplos, como o valor padrão para o parâmetro `-t` e `-l` 0.3, os exemplos da base original serão distribuídos da seguinte maneira:

- 250 exemplos para a base de teste (25% de 1000);
- 225 exemplos para a base de exemplos rotulados (30% de 750);
- 525 exemplos para a base de exemplos não rotulados (70% de 750).

Valor padrão: 0.05;

-seeds <lista de exemplos>: O parâmetro anterior, `-l`, retira exemplos aleatoriamente da base parcial para criar a base de exemplos rotulados L. No entanto, em algumas situações o usuário pode desejar escolher os exemplos que deseja inserir em L especificando-os um a um. Com o parâmetro `-seeds` o usuário pode especificar esses exemplos. Assim, caso o usuário escolha o primeiro, o terceiro e o quinto exemplo da base parcial para comporem o conjunto de exemplos rotulados L, o CO-TRAINING deve ser invocado com a seguinte linha de execução:

```
cotraining.pl --seeds "0,2,4" baseLNAI
```

Cada elemento dessa lista de exemplos indica a posição do exemplo no arquivo `.data`. Portanto, a lista `"0,2,4"` representa a primeira,

terceira e a quinta linha do arquivo `.data`.
Valor padrão: utiliza o parâmetro `-l` com valor 0.02;

Com os conjuntos de exemplos construídos a etapa preparatória para a execução do algoritmo termina, e o programa está pronto para executar o algoritmo de CO-TRAINING propriamente dito. Vários outros parâmetros podem ser utilizados na execução de CO-TRAINING, tanto no modo simulação quanto no modo real, os quais são descritos a seguir.

3.2.3 Descrição dos Parâmetros

Além dos parâmetros para a construção dos conjuntos de exemplos, `-l`, `-t` e `-seeds`, existem vários outros parâmetros que podem ser utilizados para realizar experimentos e analisar o comportamento de CO-TRAINING. A combinação desses parâmetros pode mudar consideravelmente o comportamento do algoritmo, o que fornece ao usuário um número expressivo de ajustes possíveis. Esses parâmetros são descritos a seguir:

`-i <número>`: define o número máximo de iterações que o algoritmo deve executar.
Valor padrão : 100;

`-at-least <porcentagem>`: para CO-TRAINING funcionar, os classificadores, além de rotular os exemplos, devem fornecer um valor de confiança para cada exemplo rotulado. O parâmetro `-at-least` permite definir um valor mínimo aceitável para esses valores de confiança. Dessa maneira, todos os exemplos rotulados que possuírem o valor de confiança menor que o definido por esse parâmetro são descartados.
Valor Padrão : 0.6;

`-lsm <tipo>`: *labeled selection method*, seleciona a função $melhores/2$ a ser utilizado. Como mencionado na descrição de CO-TRAINING — Algoritmo 1 na página 23 — a função $melhores/2$, responsável pela escolha dos "melhores" exemplos rotulados em cada iteração, é muito importante, e pode ser descrita de várias maneiras. Até agora, foram definidas duas funções $melhores/2$, `nbest` e `original`, mas outras funções podem ser facilmente implementadas e adicionadas ao programa. Em ambos os casos, a função $melhores/2$ ignora exemplos que foram rotulados diferentemente pelos classificadores h_{D_1} e h_{D_2}[5]. Após, cada uma das funções implementadas , `nbest` e `original`, utiliza critérios diferentes, como explicado a seguir:

[5]Por simplicidade, são considerados somente duas descrições no algoritmo apresentado, mas a implementação realizada permite utilizar mais de duas descrições

1. `nbest <n>`: utilizando o valor de confiança dado por cada classificador h_{D_1} e h_{D_2} para cada par de descrições de um mesmo exemplo classificadas com o mesmo rótulo, é calculado o que denominamos *ranking* dos exemplos, como o produto dos respectivos valores de confiança. Utilizando o valor desse *ranking* os exemplos são ordenados em forma decrescente e $melhores/2$ pega os `<n>` primeiros pares de exemplos desse *ranking* para construir (R_{D_1}, R_{D_2}).

2. `original`: neste caso, após calculado o *ranking* de maneira semelhante ao caso anterior, os exemplos são separados e ordenados por classe. Após, a função seleciona os melhores exemplos de cada classe. Para definir o número de exemplos a serem inseridos por esse método um dos seguintes parâmetros deve ser inicializado:

   `-a <número>`: este parâmetro indica que deve ser escolhida a mesma quantidade de exemplo de cada classe. Por exemplo, considerando uma base de exemplos com duas classes POSITIVO e NEGATIVO, caso `<número>` for definido como 10, então, a função escolhe, sempre que possível, os 10 melhores exemplos de cada classe (máximo 2x10 = 20) para compor R_{D_1} e R_{D_2}.
   Valor padrão: 2;

   `-c <lista de classes>`: este parâmetro indica que dever ser escolhida uma quantidade de exemplos diferente de cada classe. Considerando o exemplo anterior, para inserir 2 exemplos da classe POSITIVO e 6 da classe NEGATIVO, basta invocar a seguinte linha de comando

   ```
   cotraining.pl -c "positivo=2,negativo=6" baseLNAI
   ```

   Neste caso, em cada iteração do algoritmo poderão ser rotulado no máximo 8 exemplos.

   Esses parâmetros, `-a` e `-c`, apenas definem quantos exemplos de cada classe o usuário deseja que o algoritmo rotule em cada iteração. Desse maneira, o algoritmo tenta conseguir esse número de exemplos mas isso nem sempre é possível e portanto pode ser necessário utilizar o parâmetro `-force` explicado mais adiante.
   Valor padrão: 2 para cada classe;

   Esse dois parâmetros, `-a` e `-c`, não devem ser confundidos com os parâmetros de construção dos conjuntos L e U. Assim, vale ressaltar que ao definir esses dois parâmetros, esses valores não definem o número de exemplos no conjunto de exemplos L inicial, mas definem o número máximo de exemplos a ser rotulados e inseridos em L em cada iteração do algoritmo.

Valor Padrão : `original`;

`-force`: Este parâmetro força o algoritmo a completar os n exemplos antes de iniciar a próxima iteração. A função $melhores/2$ nem sempre consegue obter <n> "melhores" exemplos do conjunto de exemplos rotulados (ver parâmetro `-lsm`). Se isso acontecer e o parâmetro `-force` estiver ativo então nenhum exemplo é rotulado, e <n> exemplos não rotulados de U são adicionados a U'_{D_1} e U'_{D_2} respectivamente e o processo é repetido. Em outras palavras, são incrementados os conjuntos U'_{D_1} e U'_{D_2} a serem rotulados, até $melhores/2$ conseguir obter <n> exemplos rotulados. Se a situação persistir, o algoritmo tenta indefinidamente até conseguir ou o algoritmo parar por falta de exemplos em U.
Valor padrão: desabilitado;

`-v`: *verbose*, com esse parâmetro ativado, o algoritmo irá mostrar o valor de confiança da classificação de todos os exemplos de U'.
Valor padrão: desabilitado;

3.2.4 Arquivos de Saída

Na forma mais simples, executando CO-TRAINING com os parâmetros padrões, basta invocar a seguinte linha de comando, com a base `baseLNAI` da Figura 3.4 na página 27.

```
cotraining.pl baseLNAI
```

No final da execução os resultados são gravados em vários arquivos ilustrados na Figura 3.5. Como pode-se notar, a saída de CO-TRAINING consiste de 8 (oito) arquivos para cada visão, os quais ficam no subdiretório `rdata`, e 3 (três) arquivos que contém informações detalhadas das execuções. Esses arquivos são descritos em maiores detalhes a seguir.

`cotraining.log`: o arquivo de log contém todas as informações que são mostradas na tela. Essas informações mostram ao usuário os resultados da execução de cada iteração do algoritmo bem como os erros dos classificadores gerados, entre várias outras informações.

`gnuplot.dat`: após cada iteração, os classificadores gerados são testados sobre o conjunto de teste. No arquivo `gnuplot.dat` são mostrados os erros de cada classificador nesse conjunto de teste. O formato desse arquivo é similar ao formato utilizado pelo utilitário GnuPlot[6] o qual é utilizado para a geração de gráficos de desempenho.

[6] `http://www.gnuplot.info`.

```
baseLNAI/
         d1.nb/
               oneGram.data
               oneGram.names
         d2.svm/
               twoGram.data
               twoGram.names

         cotraining.log
         gnuplot.dat
         fullgnuplot.dat
         rdata/

             d1Labeled.data
             d1Labeled.names
             d1Unlabeled.data
             d1Unlabeled.names
             d1Test.data
             d1Test.names
             d1Uaux.data
             d1Uaux.names

             d2Labeled.data
             d2Labeled.names
             d2Unlabeled.data
             d2Unlabeled.names
             d2Test.data
             d2Test.names
             d2Uaux.data
             d2Uaux.names
```

Figura 3.5: Estrutura de diretório após a execução do CO-TRAINING

`fullgnuplot.dat`: fornece os erros sobre o conjunto de teste, os erros sobre o conjunto de exemplos temporários R'_{D_1} e R'_{D_2} e o número de exemplos que foram rotulados errados e foram inseridos em L_{D_1} e L_{D_2}. Essa informação é fornecida somente no caso de execução no modo simulação, no qual é conhecido o rótulo de todos os exemplos.

diretório `rdata`: o algoritmo pode ter a sua execução interrompida por várias razões: por um erro; por ter acabado os exemplos não rotulados; por ter atingido o número de iterações definidas pelo usuário; entre outros. Após parar, o CO-TRAINING salva nesse diretório todas as bases em memória. Assim, para cada visão ele cria 8 arquivos, que correspondem a 4 bases, a primeira base para os exemplos rotulados, a segunda para os exemplos não rotulados, a terceira para os exemplos de teste e finalmente a quarta base que correspondem a U'_{D_1} e U'_{D_2} ilustrados no Algoritmo 1 na página 23.

3.3 Considerações Finais

Encontrar bases de dados reais com um número significativo de exemplos rotulados, tal que é possível utilizar algoritmos de aprendizado supervisionado para a indução de bons classificadores é cada vez mais difícil, quando não impossível. Algoritmos de aprendizado semi-supervisionado, tais como CO-TRAINING descrito neste capítulo, tentam solucionar o problema rotulando exemplos a partir de poucos exemplos rotulados. O objetivo principal dessa família de algoritmo é tentar rotular, com uma certa certeza, um número maior de exemplos para, após, serem utilizados por qualquer algoritmo de aprendizado supervisionado. Para realizar essa rotulação prévia, o CO-TRAINING necessita de duas ou mais descrições dos mesmos exemplos.

Uma área na qual é simples obter duas ou mais descrições de um mesmo conjunto de de dados é a área de Mineração de Texto, na qual vários pesquisadores do Labic desenvolvem pesquisas. No próximo capítulo é descrita a ferramenta PRETEXT, que realiza pré-processamento de textos, e que foi desenvolvida como parte deste trabalho. Essa ferramenta foi utilizada para gerar duas descrições de diversos conjuntos de documentos textuais utilizados para avaliar o algoritmo CO-TRAINING como descrito no Capítulo 5.

CAPÍTULO

4

PRETEXT: Uma Ferramenta Para Pré-Processamento de Textos

O PRETEXT é uma ferramenta computacional que tem como objetivo realizar pré-processamento de textos e transformar esses textos em um formato estruturado que possa ser utilizado pela maioria dos algoritmos de Aprendizado de Máquina (AM). Deve ser observado que existem outras ferramentas disponíveis para esse fim (McCallum, 1996; Banerjee & Pedersen, 2003). Entretanto, foi decidido implementar o PRETEXT pois essas ferramentas não possuem algumas funcionalidades que consideramos necessárias. Um outro motivo é a facilidade de integração do PRETEXT no projeto DISCOVER, já que a sua implementação contempla os requisitos necessários para integrá-lo facilmente nesse ambiente, o que não acontece com as ferramentas disponíveis.

O pré-processamento de dados textuais (documentos) realizado pelo PRETEXT utiliza a abordagem *bag-of-words* que transforma esses textos, que são naturalmente não estruturados, em uma representação estruturada, mais especificamente em uma tabela atributo-valor na sintaxe padrão do DISCOVER. Várias funcionalidades foram implementadas no PRETEXT, as quais contemplam desde métodos que auxiliam o usuário a reduzir a dimensão (número de atributos utilizados) até a indução construtiva (Liu & Motoda, 1998; Lee, 2000). Entre essas funcionalidades vale destacar as diversas medidas para atribuição de valores aos atributos na representação dos textos.

4.1 Uma Visão Geral de Mineração de Texto

Nesta seção é mostrado onde se enquadra o pré-processamento que o PRETEXT realiza, entre as etapas de Mineração de Texto, bem como alguns conceitos que podem auxiliar a compreensão de algumas funcionalidades implementadas no PRETEXT.

O processo de descoberta de padrões úteis e interessantes em uma coleção de documentos (textos) é conhecido como Mineração de Textos. Devido a natureza textual não estruturada, os documentos necessitam de um pré-processamento para serem submetidos a algoritmos de aprendizado. A transformação dos documentos em uma representação mais adequada, como em uma tabela atributo-valor, é uma etapa de suma importância, visto que a representação desses documentos tem uma influência fundamental em quão bem um algoritmo de aprendizado poderá generalizar a partir dos exemplos (Sebastiani, 2002).

A representação de documentos pode ser feita usando diversas abordagens, entre elas, a abordagem *bag-of-words* utilizada no PRETEXT. Nessa abordagem cada documento é representado como um vetor das palavras que ocorrem no documento, ou em representações mais sofisticadas como frases ou sentenças. Por exemplo, este parágrafo pode ser representado por um vetor que representa as palavras e a freqüência dessas palavras deste parágrafo. Apesar de ser uma representação simples, a abordagem *bag-of-words* normalmente obtem resultados experimentais melhores que representações mais sofisticadas (Apté, Damerau, & Weiss, 1994; Dumais, Platt, Heckerman, & Sahami, 1998; Lewis, 1992). De acordo com Lewis (1992), a razão mais provável para explicar esses resultados é que, embora termos mais sofisticados tenham qualidade semântica superior, a qualidade estatística é inferior em relação a termos baseados em palavras simples. O processo de MT consiste nas seguintes etapas:

1. coleta de documentos;
2. pré-processamento;
3. extração de conhecimento;
4. avaliação e interpretação dos resultados.

A coleta de documentos é responsável pela recuperação de documentos relevantes ao domínio de aplicação do conhecimento a ser extraído. A etapa de pré-processamento é responsável por transformar os documentos em um formato adequado para serem submetidos a algoritmos de extração automática de conhecimento. A extração de conhecimento tem a finalidade de descobrir

padrões úteis e desconhecidos presentes nos documentos utilizando, entre outros, sistemas de aprendizado. Por fim, a etapa de avaliação é necessária para verificar se o objetivo foi alcançado ou se algumas das etapas necessitam ser refeitas.

O pré-processamento dos documentos é uma etapa não trivial e bastante custosa, devido à forma não estruturada dos documentos. Como mencionado, os documentos necessitam ser transformados em um formato adequado, tais como uma tabela atributo-valor, para serem submetidos a algoritmos de aprendizado. No entanto, a representação de documentos no formato atributo-valor é caracterizada pela alta dimensão e por dados bastante esparsos, sendo fundamental que essa etapa seja realizada com devido cuidado para obter um bom desempenho do processo de MT. A etapa de pré-processamento de documentos é descrita em maiores detalhes a seguir.

4.1.1 O Pré-Processamento de Textos

Considerando que a primeira etapa do processo tenha sido cumprida, ou seja, os documentos estejam disponíveis, é necessário realizar o pré-processamento desses documentos. Um dos procedimentos geralmente adotado, e utilizado neste trabalho, é a representação usando a abordagem *bag-of-words*, na qual cada documento é representado como um vetor de palavras que ocorrem no documento. A representação de documentos usando a abordagem *bag-of-words* pode utilizar o formato de uma tabela atributo-valor, como mostrado na Tabela 4.1. Essa tabela representa uma coleção de N documentos $D = \{d_1, d_2, ..., d_N\}$ e um conjunto de Ncl categorias $C = \{c_1, c_2, ..., c_{Ncl}\}$ associadas à coleção de documentos D. Cada documento d_i é um exemplo da tabela e cada termo (palavra) t_j é um elemento do conjunto de atributos.

	t_1	t_2	...	t_M	C
d_1	a_{11}	a_{12}	...	a_{1M}	c_1
d_2	a_{21}	a_{22}	...	a_{2M}	c_2
...	...	...	...	...	...
d_N	a_{N1}	a_{N2}	...	a_{NM}	c_N

Tabela 4.1: Representação de documentos

Mais especificamente, na Tabela 4.1 estão representados N documentos (exemplos) compostos por M termos (atributos). Cada documento d_i é um vetor $d_i = (a_{i1}, a_{i2}, ...a_{iM})$, no qual o valor a_{ij} refere-se ao valor associado ao j-ésimo termo do documento i. O valor a_{ij}, do termo t_j no documento d_i, pode ser calculado utilizando diferentes medidas, tais como:

***boolean*:** A medida *boolean* usa a representação binária para os termos. Quando o termo está presente no documento o valor de a_{ij} é 1, caso contrário 0.

***tf*:** A medida *tf* (*term frequency*) considera o valor de a_{ij} como a freqüência absoluta que o termo aparece no documento. Essa medida é definida pela Equação 4.1, na qual $freq(t_j, d_i)$ é o número de vezes que o termo t_j ocorre no documento d_i.

$$tf(t_j, d_i) = freq(t_j, d_i) \tag{4.1}$$

***tfidf*:** Esta medida, definida pela Equação 4.2, é bastante citada na literatura. A freqüência do termo no documento é multiplicada por um fator de ponderação. Esse fator varia entre 0 e $\log N$ e é calculado pela Equação 4.3, na qual N é o número de documentos da coleção e $d(t_j)$ é o número de documentos nos quais o termo t_j ocorre pelo menos uma vez. Assim, quando o termo aparece em todos os documentos o fator de ponderação é igual a 0, quando aparece em apenas 1 documento ele é $\log N$.

$$tfidf(t_j, d_i) = freq(t_i, d_j)\ .\ idf(t_j) \tag{4.2}$$

$$idf(t_j) = \log \frac{N}{d(t_j)} \tag{4.3}$$

***tflinear*:** Esta medida é uma variação da medida *tfidf*. Sua principal diferença está no fator de ponderação, definido pela Equação 4.5, que é linear. Desse modo, o fator de ponderação de *tflinear* varia entre 0 e 1. De modo similar a medida *tfidf*, quando o termo aparece em todos os documentos o fator de ponderação será zero. Quando o termo aparece em apenas um documento o termo será multiplicado por um valor muito próximo de 1. Existe uma diferença significativa em relação a medida *tfidf* pois o valor máximo da *tfidf* é $\log N$, enquanto que a medida *tflinear* é 1.

$$tflinear(t_j, d_i) = freq(t_i, d_j)\ .\ linear(t_j) \tag{4.4}$$

$$linear(t_j) = 1 - \frac{d(t_j)}{N} \tag{4.5}$$

Um outro aspecto que também deve ser levado em conta é o tamanho dos documentos na coleção. Freqüentemente, o tamanho desses documentos é muito diferente e essa diferença de tamanho deveria estar melhor refletida nas medidas utilizadas. Por exemplo, considere dois documentos que pertencem a mesma categoria com tamanhos de 1 Kbyte e 100 Kbytes respectivamente. Nesse caso, existirá uma grande diferença na freqüência dos termos em ambos os documentos. Uma possível solução para esse problema é normalizar

os valores da tabela atributo-valor — Tabela 4.1 na página 37. Essa normalização é freqüentemente realizada usando a normalização linear, definida pela Equação 4.6, ou a normalização quadrática, definida pela Equação 4.7.

$$NormLinear(t_j, d_i) = \frac{a_{ij}}{MAX_{i=1..N}(a_{ij})} \tag{4.6}$$

$$NormQuadratic(t_j, d_i) = \frac{a_{ij}}{\sqrt{\sum_{i=1}^{N}(a_{ij})^2}} \tag{4.7}$$

4.1.2 Redução da Dimensionalidade dos Atributos

Representar uma coleção de documentos utilizando a abordagem *bag-of-words* pode ser muito custoso pois o vetor que representa cada documento deve conter todos os termos de toda a coleção de documentos. Facilmente um conjunto de somente 10 documentos de tamanho médio pode necessitar de mais de 2000 atributos (termos) para ser representado, ou seja, cada documento é representado por um vetor de alta dimensão. Assim, quanto maior o número de documentos na coleção, o tamanho desses documentos bem como os termos tratados, provavelmente, menor será a porcentagem de valores não nulos a_{ij} dos termos utilizados na representação da tabela atributo-valor — Tabela 4.1 na página 37. Em outras palavras, essa tabela é uma tabela esparsa.

Vários métodos podem ser utilizados a fim de reduzir a quantidade de termos (atributos) necessário para representar uma coleção de documentos, visando uma melhor representação e melhor desempenho do processo de MT. Entre outros, a transformação de cada termo para o radical que o originou, por meio de algoritmos de *stemming*, é um método amplamente utilizado e difundido. Uma outra forma para reduzir a dimensionalidade dos atributos é buscando os termos que são mais representativos na discriminação dos documentos usando a Lei de Zipf e os cortes de Luhn.

Basicamente, algoritmos de *stemming* consistem em uma normalização lingüística, na qual as formas variantes de um termo são reduzidas a uma forma comum denominada *stem*. A conseqüência da aplicação de algoritmos de *stemming* consiste na remoção de prefixos ou sufixos de um termo, ou mesmo na transformação de um verbo para sua forma no infinitivo. Por exemplo, as palavras `trabalham`, `trabalhando`, `trabalhar`, `trabalho` e `trabalhos` podem ser transformados para um mesmo *stem* `trabalh`. Desse modo, algoritmos de *stemming* podem ser utilizados para reduzir a dimensão da tabela atributo-valor.

Percebe-se que algoritmos de *stemming* são fortemente dependentes do idioma no qual os documentos estão escritos. Um dos algoritmos de *stemming*

mais conhecidos é o algoritmo do Porter, que remove sufixos de termos em inglês (Porter, 1980). O algoritmo tem sido amplamente usado, referenciado e adaptado nos últimos 20 anos. Diversas implementações do algoritmo estão disponibilizadas na *Web*, entre elas a página oficial escrita e mantida pelo autor para distribuição do seu algoritmo (`http://www.tartarus.org/~martin/PorterStemmer`).

No PRETEXT, além do algoritmo de *stemming* do Porter para o inglês, ele foi adaptado e implementado para a lingua portuguesa e o espanhol (Matsubara, Martins, & Monard, 2003). No algoritmo de Porter, os sufixos que estão ao final de um *stem* com um comprimento mínimo estabelecido são eliminados considerando algumas regras pré-estabelecidas. Caso não seja possível eliminar nenhum sufixo de acordo com essas regras, são analisadas as terminações verbais da palavra. Essa é a principal diferença entre o algoritmo de *stemming* para palavras em inglês e palavras em português ou espanhol. Ou seja, o fato das linguagens provenientes do latim terem formas verbais altamente conjugadas em sete tempos, cada uma com seis terminações diferentes, faz necessário ter um tratamento diferenciado para essas terminações.

É pouco provável que o algoritmo de *stemming* retorne o mesmo *stem* para todas as palavras que tenham a mesma origem ou radical morfológico, devido a própria natureza intrínseca dos sistemas relacionados à Recuperação de Informações (RI) e Processamento de Linguagem Natural (PLN). Já foi observado que a medida que o algoritmo se torna mais específico na tentativa de minimizar a quantidade de *stems* diferentes para palavras com um mesmo radical, a eficiência do algoritmo degrada (Porter, 1980).

Um outro modo para reduzir a dimensionalidade dos atributos é buscar os termos que são mais representativos na discriminação dos documentos. A Lei de Zipf descreve uma maneira de encontrar termos considerados pouco representativos em uma determinada coleção de documentos. A lei, formulada por George Kingsley Zipf, professor de lingüística de Harvard (1902-1950), declara que a freqüência de ocorrência de algum evento está relacionada a uma função de ordenação. Zipf mostrou que uma das características das linguagens humanas, populações das cidades e muitos outros fenômenos humanos e naturais, seguem uma distribuição similar, a qual denominou de "*Principle of Least Effort*" (Zipf, 1949).

Existem diversas maneiras de aplicar a Lei de Zipf para uma coleção de documentos. A mais simples é procedimental: pegar todos os termos na coleção e contar o número de vezes que cada termo aparece. Se o histograma resultante for ordenado de forma decrescente, ou seja, o termo que ocorre mais freqüentemente aparece primeiro, então, a forma da curva é a "curva de Zipf" para aquela coleção de documentos. Se a curva de Zipf for plotada em

uma escala logarítmica, ela aparece como uma reta com inclinação -1, embora Gelbukh & Sidorov (2001) mostram que a curva de Zipf é dependente da linguagem e do estilo. A Lei de Zipf em documentos de linguagem natural pode ser aplicada não apenas aos termos mas, também, a frases e sentenças da linguagem. Na realidade, a lei de Zipf é uma observação empírica que se aplica em diversos domínios, e segue a seguinte distribuição:

$$p_1 = \frac{c}{1}, p_2 = \frac{c}{2}, ..., p_n = \frac{c}{n} \tag{4.8}$$

na qual $c = \frac{1}{H_n}$ e $H_n = \sum_{j=1}^{n} \frac{1}{j}$ (Knuth, 1973). Ou seja, considerando uma coleção de documentos escritos em linguagem natural, pode ser observado que o j-ésimo termo mais comum ocorre com freqüência inversamente proporcional a j. A Lei de Zipf não é restrita apenas aos termos mas também a *stem* ou sentenças dos documentos para a maiorias dos idiomas.

Enquanto Zipf verificou sua lei utilizando jornais escritos em inglês, Luhn usou a lei como uma hipótese nula para especificar dois pontos de corte, os quais denominou de superior e inferior, para excluir termos não relevantes (Luhn, 1958). Os termos que excedem o corte superior são os mais freqüentes e são considerados comuns por aparecer em qualquer tipo de documento, como as preposições, conjunções e artigos. Já os termos abaixo do corte inferior são considerados raros e, portanto, não contribuem significativamente na discriminação dos documentos. A Figura 4.1 mostra a curva da Lei de Zipf (I) e os cortes de Luhn aplicados a Lei de Zipf (II), no qual o eixo cartesiano f representa a freqüência das palavras e o eixo cartesiano r as palavras correspondentes ordenadas segundo essa freqüência.

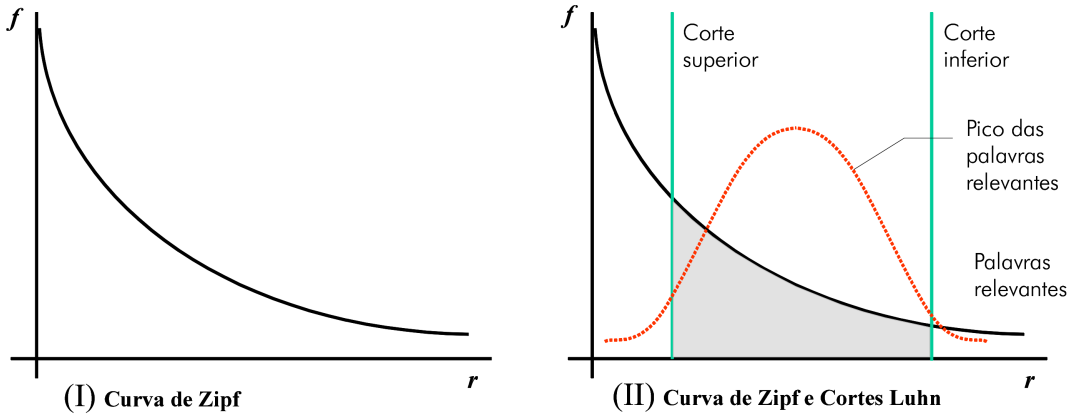


Figura 4.1: A curva de Zipf e os cortes de Luhn

Assim, Luhn propôs uma técnica para encontrar termos relevantes, assumindo que os termos mais significativos para discriminar o conteúdo de documentos estão em um pico imaginário posicionado no meio dos dois pontos de corte. Porém, uma certa arbitrariedade está envolvida na determinação dos

pontos de corte, bem como na curva imaginária, os quais são estabelecidos por tentativa e erro (Van Rijsbergen, 1979). A seguir, é apresentada a ferramenta computacional PRETEXT que implementa os conceitos apresentados.

4.2 A Ferramenta Computacional PRETEXT

O PRETEXT é uma ferramenta computacional desenvolvida com o objetivo de realizar o pré-processamento de uma coleção de documentos utilizando a abordagem *bag-of-words*. Uma de suas principais funcionalidades é transformar palavras escritas em português, espanhol ou inglês, em *stems*. O algoritmo de *stemming* implementado na ferramenta é baseado no algoritmo do Porter para a língua inglesa e adaptado para o português e o espanhol. A implementação foi realizada utilizando o paradigma de orientação a objetos em Perl (Wall, Christiansen, & Schwartz, 1996).

Como mencionado, o pré-processamento de documentos é uma etapa árdua e composta por uma seqüência de passos, os quais muitas vezes precisam ser refeitos. Considerando essas características, PRETEXT, ilustrado na Figura 4.2, foi implementado em dois módulos: **stem.pl** e **report.pl**. Desse modo, é possível tornar esse processo menos dispendioso criando arquivos intermediários para que, caso necessário, o processo seja refeito apenas a partir do passo necessário.

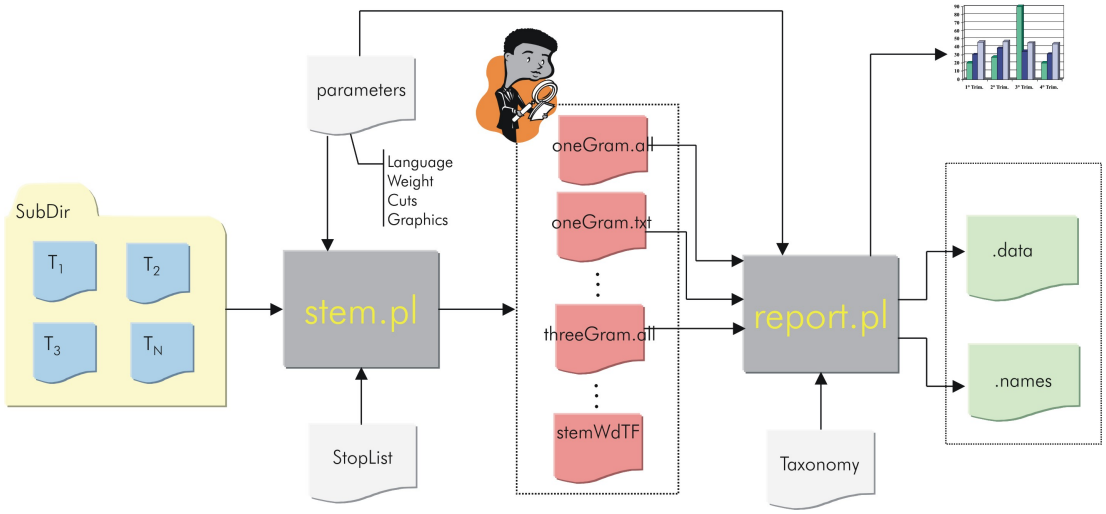


Figura 4.2: A ferramenta PRETEXT

O módulo **stem.pl** da ferramenta é responsável pela transformação das palavras em *stems*. Os documentos a serem processados devem estar em um diretório, como ilustrado na Figura 4.2, ou em uma hierarquia de subdiretórios. Para execução do módulo **stem.pl** é necessário um arquivo contendo algumas especificações dos parâmetros de execução, bem como um diretório contendo uma ou mais listas de *stopwords*, que são palavras pouco significativas como

artigos, preposições e conjunções que pouco caracterizam os documentos. A saída desse módulo consiste em diversos arquivos intermediários, denominados de **stembase**, que contém os *stems* correspondentes às palavras contidas nos documentos e informações úteis relacionadas a cada um dos *stems*.

É importante ressaltar que, para uma mesma coleção de documentos e uma mesma lista de *stopwords*, o módulo **stem.pl** gera a mesma **stembase**. Assim, uma vez transformados os documentos em um conjunto de *stems*, esse módulo não precisa ser executado novamente, e o usuário pode gerar diferentes tabelas atributo-valor executando apenas o módulo **report.pl** com diferentes parâmetros.

O módulo **report.pl**, a partir de alguns arquivos gerados pelo módulo **stem.pl** e os parâmetros especificados pelo usuário, retorna informações para geração de gráficos e criação da tabela atributo-valor. A tabela é gerada na sintaxe DSX representados pelo arquivos de dados (`.data`) e pelo arquivo de atributos (`.names`). Para calcular os valores dos atributos na tabela, a ferramenta utiliza qualquer uma das medidas implementadas, descritas na Seção 4.1.1 na página 37, de acordo com o parâmetro especificado pelo usuário. Além disso, a ferramenta apresenta facilidades para reduzir a dimensionalidade do conjunto de atributos usando a lei de Zipf e os cortes de Luhn.

Uma das características da ferramenta é a construção de *stems* usando mais de um *gram*. No PRETEXT[1], 1-*gram* se refere a um *stem* simples, enquanto 2 e 3-*gram* referem-se a 2 ou 3 *stems*, cujas palavras ocorrem seqüencialmente no documento. A ferramenta permite a concatenação de até 3 *stems*, ou seja, 3-*gram*. Porém, os *gram* são formados a partir dos *stems* gerados e, portanto, termos que são *stopwords*, como por exemplo o termo 'de', não comparecem na concatenação de *stems*. Desse modo, é possível trabalhar com a combinação de *gram* a fim de obter uma melhor representação dos documentos. A utilização de mais de um *gram* permite que palavras que aparecem seqüencialmente no documento como "inteligência artificial", "aprendizado de máquina" e "mineração de texto", que são mais representativas conceitualmente quando utilizadas juntas, possam ser utilizadas no PRETEXT.

Uma outra característica da ferramenta é a facilidade de usar indução construtiva na criação de novos atributos na tabela atributo-valor. A indução construtiva é realizada utilizando um arquivo contendo um conjunto de taxonomias definidas pelo usuário, o qual, quando disponível, é interpretado pelo módulo **report.pl**.

A ferramenta PRETEXT está implementada usando idéias aceitas e difun-

[1]É importante ressaltar que existem outras formas para representação de *gram*. Neste trabalho, n-*gram* significa que n *stems* são concatenados formando um único *stem*. Ainda a palavra *gram* deve ser usada sempre no singular.

didas na comunidade científica. No entanto, PRETEXT possui diversas facilidades implementadas e, quando executada usando uma coleção de documentos, retorna uma grande quantidade de informações relacionadas a esses documentos. O grande diferencial da ferramenta consiste nessa diversidade de informações geradas, uso de taxonomias e o uso de várias medidas que podem ser utilizadas para auxiliar no processo de Mineração de Textos.

Nas próximas seções são apresentados resumidamente cada um dos módulos **stem.pl** e **report.pl**. Maiores detalhes encontram-se em (Matsubara, Martins, & Monard, 2003).

4.2.1 Módulo *stem.pl*

O módulo **stem.pl** é responsável por transformar as palavras dos documentos em *stems* e armazenar essas informações no diretório **stembase**. As principais características deste módulo são:

- implementação do algoritmo de *stemming* para três idiomas: inglês, português e espanhol. Como já mencionado, para o inglês foi utilizado a implementação do *stemming* mantida pelo Porter. Os dois últimos, português e espanhol, foram implementados e adicionados no PRETEXT;

- utilização de *stoplists*. A **stoplist** é um diretório contendo um ou mais arquivos com as *stopwords*. Cada um desses arquivos são denominados *stopfiles*. Esses *stopfiles* contém palavras como artigos, preposições, conjunções, entre outras, que são pouco significativas para caracterizar o documento. O PRETEXT aloca em memória apenas as *stopfiles* que tiverem a extensão `.enabled`. Desse modo, é possível habilitar e desabilitar várias *stopfiles* apenas alterando a sua extensão;

- geração da **stembase**. A **stembase** fornece informações referentes a cada palavra, o que permite saber em qual *stem* cada palavra foi transformada e em quais documentos se encontram cada um desses *stems*. Na **stembase** é possível observar, entre outras coisas, quais são os *stems* que podem ser *stopwords* em potencial, observando a freqüência e em quantos documentos o *stem* aparece. Além das *stopwords*, é possível verificar quais são os pontos de cortes máximo e mínimo mais razoáveis verificando a freqüência com que as palavras mais relevantes aparecem.

- construção dos 2-*gram* e 3-*gram*. Após a transformação das palavras em *stems*, podem ser gerados 2-*gram* e 3-*gram* desses *stems*. Neste trabalho, os 2-*gram* são utilizados para compor a segunda descrição necessária para o CO-TRAINING.

4.2.2 Módulo ***report.pl***

A base de *stems* (**stembase**) gerada pelo módulo **stem.pl** contém, entre outros, os arquivos com informações sobre as freqüências dos *stems*. A partir desses arquivos, o PRETEXT cria a tabela atributo-valor, na qual cada documento é um exemplo, e os *stems* são os atributos dessa tabela no formato padrão do sistema DISCOVER. As principais características deste módulo são:

- implementação das 4 medidas: *bool*, *tf*, *tflinear* e *tfidf*. As duas primeiras, como já explicado, são medidas muito simples que consideram se o termo aparece ou não no documento ou a freqüência do termo neste documento. As duas últimas utilizam fatores de ponderação.

- dois tipos de normalizações: linear e quadrática. Com as normalizações a tabela atributo-valor gerada contém exemplos com valores mais uniformes descrevendo melhor os exemplos. Essas normalizações são indicadas para bases em que o tamanho dos documentos apresenta grande variabilidade.

- método *smooth*: as medidas *tflinear* e *tfidf* utilizam um fator de ponderação. Quando o fator de ponderação se torna zero, ou seja, nos casos em que o termo aparece em todos os documentos, esse fator pode causar alguns problemas pois o valor zero é multiplicado por todos os valores da coluna que representa esse termo. Assim, a coluna inteira que representa o termo é zero. Pode parecer razoável ignorar esse termo já que ele aparece em todos os documentos e provavelmente pode não possuir nenhum valor que possa descriminar as classes. Por outro lado, por mais que o termo apareça em todos os documentos, podem ocorrer casos em que essa informação é útil. Desse modo, para evitar que o fator de ponderação seja zero, o PRETEXT utiliza o *smooth* que, quando ativo, atribui um valor muito pequeno ao fator de ponderação.

- os cortes de Luhn: com os cortes é possível definir um máximo e um mínimo na freqüência dos termos em toda a coleção de documentos. Desse modo, o usuário pode definir os limites superiores e inferiores apresentados na Figura 4.1 na página 41.

- uso de taxonomias: o usuário pode fornecer uma taxonomia, o que permite realizar indução construtiva nos atributos. A indução construtiva é caracterizada pela criação de novos atributos a partir de alguns atributos já existentes. Esses novos atributos, geralmente, são generalizações de dois ou mais atributos.

O PRETEXT foi utilizado para desenvolver diversos trabalhos relacionados com Mineração de Textos (Melo, 2004; Brasil, 2004; Martins, 2003; Martins, Monard, & Matsubara, 2003b,a, 2004; Martins, Godoy, Monard, Matsubara, & Amandi, 2003; Matsubara & Monard, 2004b,a).

4.3 Considerações Finais

Uma restrição do algoritmo CO-TRAINING é a necessidade de duas descrições de exemplos as quais nem sempre encontram-se disponíveis para a execução do algoritmo. Entretanto, é simples obter duas, ou mais, descrições de um conjunto de textos utilizando, por exemplo, palavras simples (1-*gram*) e compostas (2-*gram*) para representar cada uma dessas descrições. Assim, o desenvolvimento do PRETEXT foi muito importante para este trabalho de mestrado.

Além disso, o PRETEXT foi desenvolvido com o intuito de oferecer uma ferramenta bastante completa de pré-processamento de textos e não somente para gerar as descrições necessárias para o CO-TRAINING. Boa parte das idéias difundidas na comunidade científica foram implementadas no PRETEXT, o que possibilita ao usuário da ferramenta realizar pré-processamento de textos utilizando uma ampla diversidade de parâmetros e modos de execução.

CAPÍTULO

5

Análise Experimental do Algoritmo CO-TRAINING

Neste capítulo são descritos vários experimentos realizados com o objetivo de avaliar CO-TRAINING e a influência dos diversos parâmetros implementados para sua execução. Os experimentos foram conduzidos utilizando três bases de documentos (textos), as quais foram pré-processadas com a ferramenta PRETEXT. Esses experimentos visam analisar os erros cometidos por CO-TRAINING na rotulação de novos exemplos em cada iteração do algoritmo, bem como o comportamento de CO-TRAINING como um algoritmo de classificação, *i.e.*, analisando o modelo final induzido.

5.1 Pré-Processamento das Bases de Documentos

Como já mencionado, a etapa de pré-processamento de textos, *i.e.* a transformação dos textos em uma tabela atributo-valor, não é trivial. Nesta seção é descrito em detalhes como foram feitas essas transformações para as três bases de textos utilizadas nos experimentos, denominadas NEWS, LNAI e COURSE descritas a seguir:

NEWS a base NEWS foi criada a partir da base *mini-news* (Blake & Merz, 1998) que contém textos relacionados a *news-groups*. Essa base foi criada com o objetivo de testar o CO-TRAINING com um conjunto balanceado de classes. A base original *mini-news* consiste de documentos classificados em 20 classes diferentes com 100 documentos de cada classe. A nova base NEWS foi construída agrupando as 4 classes da base *mini-news*

relacionadas com `sci.crypt`, `sci.electronics`, `sci.med` e `sci.space` em uma única classe denominada `sci`, e as 4 classes relacionadas com `talk.politics.guns`, `talk.politics.mideast`, `talk.politics.misc` e `talk.religion.misc` em uma única classe denominada `talk`. Assim, a base NEWS consiste de 800 documentos classificados em duas classes, `sci` e `talk`, cada uma delas com 400 documentos. A primeira descrição é dada por 1-*gram* e a outra por 2-*gram* do conjunto de documentos.

LNAI Este conjunto de exemplos consiste dos títulos, resumos e referências de artigos de *Case-Based Reasoning* (CBR) e *Inductive Logic Programming* (ILP) retirados de *Lecture Notes in Artificial Intelligence* (LNAI). A base LNAI tem um total de 396 artigos dos quais 277 (70%) são da classe CBR e 119 (30%) são da classe ILP. Além das duas descrições, uma descrição referente a 1-*gram* e a outra referente a 2-*gram*, que podem ser extraídas desse conjunto de documentos, é possível extrair outras descrições considerando, por exemplo, o *bag-of-words* dos títulos, dos resumos e/ou das referências.

Esse córpus foi produzido para auxiliar no desenvolvimento e avaliação do projeto Uma Ferramenta Inteligente de Apoio à Pesquisa (FIP) que está sendo realizado no Labic coordenado pelo professor Alneu de Andrade Lopes, tendo participação de três alunos de mestrado e um de iniciação científica. A FIP tem como objetivo a recuperação automática de artigos científicos sobre áreas pré-selecionadas na web; análise desses artigos por meio de técnicas de Aprendizado de Máquina e Mineração de Dados; a construção de um mapa representativo do conhecimento das sub-áreas, tópicos e temas de pesquisas; importância dos artigos e autores, entre outros (Melo, Secato, & Lopes, 2003; Melo, 2004; Brasil, 2004).

COURSE Este conjunto de exemplos foi construído a partir das bases de textos e links utilizado no artigo de Blum & Mitchell (1998) no qual é proposto o algoritmo CO-TRAINING [1]. Essa base contém 1051 exemplos referentes a páginas de internet coletadas de quatro universidades: Universidade de Cornell, Universidade de Washington, Universidade de Wisconsin e Universidade do Texas. As páginas estão em formato html e estão dividas em duas classes: páginas referentes aos cursos oferecidos nessas universidade (`course`) e páginas que não se referem a cursos (`non-course`). As duas descrições são formadas pelos textos dessas páginas e pelos links que apontam para as mesmas. Neste trabalho a primeira descrição, composta por textos, é sempre referida como TEXTO, e a segunda descrição

[1] `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-51/www/co-training/data`

como LINKS. Foi observado que nem todos os exemplos (páginas de internet) na descrição LINKS continham algum conteúdo, portanto, ao realizar o pré-processamento dessa base com o PRETEXT, esses exemplos foram retirados do conjunto, o que reduziu o número de exemplos da base original para 1037 exemplos. Desses 1037 exemplos, 223 (21%) são da classe `course` e 817 (79%) da classe `non-course`.

Na Tabela 5.1 é mostrado um resumo das características desses conjuntos de documentos.

Conj. Doc.	#Doc.	Classes	%Classe	Erro da Classe Majoritária
NEWS	800	(talk,sci)	(50%,50%)	50%
LNAI	396	(ILP,CBR)	(30%,70%)	70%
COURSE	1037	(course,non-course)	(21%,79%)	79%

Tabela 5.1: Conjuntos de documentos

Os três conjuntos de documentos foram pré-processados utilizando a ferramenta PRETEXT e transformados em uma tabela atributo-valor. Inicialmente, utilizando a lista de *stopwords* padrão do PRETEXT, os textos foram transformados em *stems*[2]. Depois foram aplicados os cortes de Luhn nos quais é possível especificar a freqüência mínima e máxima dos termos da coleção. Por exemplo, ao especificar mínimo igual a 3 em uma coleção com 10.000 documentos, o PRETEXT ignora todos os termos que apareceram uma ou duas vezes em todos os 10.000 documentos. Note que os termos que aparecem três vezes são mantidos.

Na Tabela 5.2 são apresentados os cortes de Luhn utilizados em cada descrição de cada conjunto de documentos. Por exemplo, a descrição TEXTO do conjunto COURSE foi "cortado" em mínimo = 2, ou seja, todos os termos que aparecem apenas 1 vez em toda a coleção desses documentos foram retirados da tabela atributo-valor que representa esses documentos. Antes de efetuar esse corte a descrição TEXTO continha 13198 termos, após o corte esse número foi reduzido para 6870 termos.

Conj. Doc.	Descrição	Min	Max	#Atr Antes dos Cortes	#Atr Após os Cortes
NEWS	1-*gram*	2	-	15711	8668
	2-*gram*	3	-	71039	4521
LNAI	1-*gram*	2	400	5627	2914
	2-*gram*	2	-	21969	3245
COURSE	TEXTO	2	-	13198	6870
	LINKS	2	-	1604	1067

Tabela 5.2: Cortes de Lunh

[2]Como valor associado a cada atributo (*stem*) dessa tabela, foi utilizado a medida tf, Equação 4.1 na página 38

Como pode ser observado na Tabela 5.2, nos três conjuntos de documentos e para todas as descrições foi definido o corte mínimo de Luhn igual a 2 (dois), com exceção da descrição 2-*gram* da base NEWS. Dentre as três bases, a base NEWS é a que apresenta o maior número de *stems* (termos) diferentes e, quando aplicado o corte mínimo igual a 2 (dois), foi observado que os termos com freqüência igual a 2 da descrição 2-*gram* eram irrelevantes. Assim, foi decidido desconsiderar esses termos aplicando um corte mínimo igual a 3.

CO-TRAINING foi avaliado utilizando 10-*fold cross-validation*. Entretanto, deve ser lembrado que CO-TRAINING utiliza duas descrições do conjunto de exemplos, os quais devem se corresponder um a um. Assim, o método de reamostragem 10-*fold cross-validation* foi adaptado para essa particularidade do algoritmo, como ilustrado na Figura 5.1. Após o pré-processamento das bases de documentos, cada uma das descrições do conjunto de exemplos foi particionado em 10 subconjuntos, sendo que cada subconjunto contém exemplos correspondentes a cada uma das descrições. Essas partições são sempre pareadas uma a uma, tal que as mesmas partições de ambas as descrições, em cada iteração, são utilizadas para treinamento (90%) e teste (10%). Todos os experimentos com CO-TRAINING, descrito a seguir, foram avaliados utilizando 10-*fold cross-validation*, no qual os 10 *folds* foram construídos como ilustrado na Figura 5.1.

Um ponto importante a ser ressaltado é a validade do método de amostragem utilizado. Como o nosso objetivo é avaliar o comportamento de CO-TRAINING e não a qualidade do pré-processamento dos textos, o pré-processamento realizado, *i.e.*, a medida utilizada tf bem como os cortes de Luhn, são muito simples e não utilizam informações adicionais dos conjuntos de teste. Caso for realizado um pré-processamento utilizando medidas mais sofisticadas para determinar o valor de cada termo (*stem*) nos documentos, bem como uma redução mais apurada da dimensão dos atributos, então, cada um do 10 (dez) conjuntos de treinamento e teste deveriam ser pré-processados independentemente (Batista, 2003).

5.2 Experimentos

Como descrito no Capítulo 3, o CO-TRAINING pode ser executado em dois modos diferentes: modo simulação e modo real. A principal diferença entre esses modos de execução refere-se ao conjunto de exemplos não rotulados U. No modo simulação, o conjunto U é construído artificialmente a partir de um conjunto de exemplos rotulados. Dessa maneira, é possível verificar, a cada iteração do algoritmo, se os rótulos atribuídos por CO-TRAINING aos exemplos "não rotulados" coincidem com os verdadeiros rótulos desses exemplos. Todos

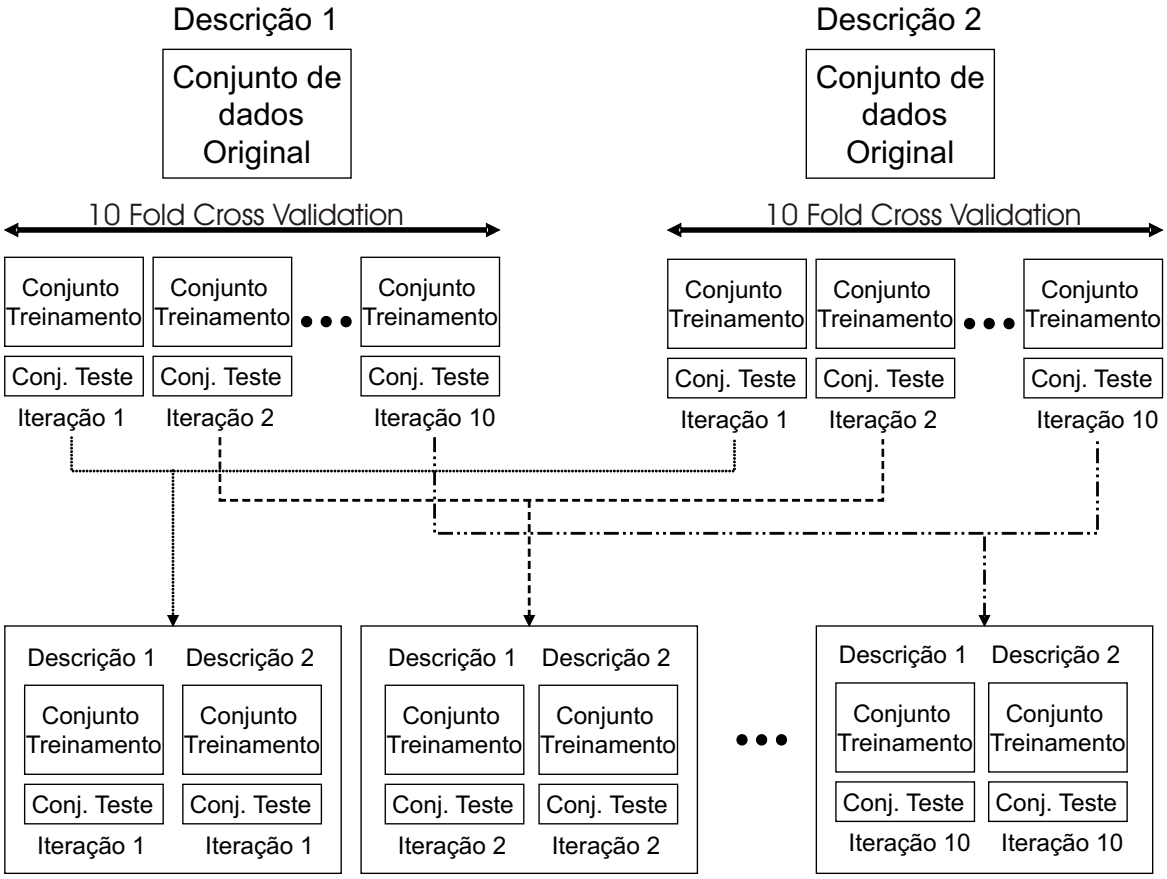


Figura 5.1: Representação gráfica da construção dos 10-*folds* para CO-TRAINING

os experimentos com CO-TRAINING realizados neste trabalho foram executados no modo simulação utilizando os conjuntos de documentos e as duas descrições desses conjuntos, descritos na Seção 5.1 na página 47. Foi utilizado o algoritmo *Naive Bayes* para construir, em cada iteração do algoritmo, os classificadores h_{D_1}, h_{D_2} e o classificador combinado h — Algoritmo 2 na página 23. Além disso, o CO-TRAINING foi executado diversas vezes variando os valores dos parâmetros de execução.

Antes de executar CO-TRAINING, e com o objetivo de verificar o erro dos classificadores induzidos por algoritmos de AM supervisionado que tem *bias* bem diferentes, foram executados os algoritmos $\mathcal{C}4.5$, que induz árvores de decisão, e *Naive Bayes*. O erro desses classificadores foi medido usando 10-*fold cross validation* estratificado. Na Tabela 5.3 são mostrados os resultados obtidos.

Nas últimas duas colunas dessa tabela são mostrados, respectivamente, o erro dos classificadores induzidos por $\mathcal{C}4.5$ e *Naive Bayes* (*Overall*) e o erro desses classificadores em cada uma das classes, junto com o desvio padrão desses erros. Como pode ser observado na Tabela 5.3 na próxima página, as bases COURSE e LNAI são desbalanceadas, já que possuem classes com

Conj. Doc.	#Doc	Descrição	#Atributos	Classe	%Classe	Erro C4.5	Erro NB
NEWS	800	1-*gram*	8668	`sci`	50%	0.5 (0.3)	2.5 (1.7)
				`talk`	50%	1.0 (0.4)	0.8 (1.2)
				Overall		0.8 (0.2)	1.6 (1.0)
		2-*gram*	4521	`sci`	50%	0.0 (0.0)	2.0 (2.0)
				`talk`	50%	0.0 (0.0)	0.5 (1.1)
				Overall		0.0 (0.0)	1.3 (1.2)
LNAI	396	1-*gram*	2914	ILP	30%	18.3 (2.4)	1.7 (3.7)
				CBR	70%	3.6 (1.2)	1.4 (1.9)
				Overall		8.1 (1.2)	1.5 (1.8)
		2-*gram*	3245	ILP	30%	27.9 (5.4)	1.8 (1.7)
				CBR	70%	3.6 (0.8)	1.5 (1.9
				Overall		10.9 (1.6)	1.8 (1.7)
COURSE	1038	TEXTO	6870	`course`	21%	33.0 (3.1)	16.3 (5.4)
				`non-course`	79%	4.9 (0.5)	3.8 (2.0)
				Overall		10.9 (0.8)	6.5 (2.3)
		LINKS	1067	`course`	21%	28.9 (3.1)	9.6 (7.6)
				`non-course`	79%	2.8 (0.4)	16.0 (4.7)
				Overall		8.4 (0.7)	14.6 (3.5)

Tabela 5.3: Erros dos algoritmos $\mathcal{C}4.5$ e *Naive Bayes* nos conjuntos de exemplos

proporção de exemplos bastante diferente. Nesses casos, o erro na classe minoritária é usualmente maior que na classe majoritária (Batista, Prati, & Monard, 2004; Prati, Batista, & Monard, 2004). Entretanto, o resultado obtido por *Naive Bayes* na descrição LINKS da base COURSE não segue esse padrão. Nesse caso, o erro da classe minoritária (`course` com 21% dos exemplos) é 9.55 com 7.56 de desvio padrão, foi menor que o erro da classe majoritária (`non-course` com 79% dos exemplos) que é 16.03 com 4.69 de desvio padrão. Ainda que dentre os algoritmos de AM supervisionado já tem sido reportado na literatura que a *Naive Bayes* é um dos menos sensíveis à distribuição das classes, esse resultado chama a atenção.

A seguir são descritos os diversos experimentos realizados com CO-TRAINING.

5.2.1 Experimento 1: Execução com os Valores Padrão dos Parâmetros

Inicialmente, para avaliar o algoritmo CO-TRAINING, foram executados experimento com os valores padrão dos parâmetros, ilustrados a seguir:

número de iterações	:	100
conjunto de exemplos inicial	:	5%
valor mínimo de confiança	:	0.6
função $melhores/2$	:	`original`
parâmetro `-force`	:	desabilitado
conjunto de teste	:	arquivo `.test`

Deve ser observado que, em algumas execuções, o algoritmo pode interromper sua execução antes de chegar as 100 iterações, isso acontece caso o conjunto de exemplos não rotulados estiver vazio ou não for possível rotular mais exemplos com o valor mínimo de confiança. Com relação ao conjunto de teste, ele consiste, em cada uma das execuções realizadas usando a adaptação de 10-*fold cross validation* ilustrado na Figura 5.1 na página 51, de um arquivo `.test` que contém os 10% dos exemplos de teste para os respectivos 90% de exemplos de treinamento.

Uma das principais medidas de desempenho do CO-TRAINING é o número de exemplos rotulados errados em cada iteração e no fim da execução. A seguir, o conjunto inicial de exemplos com rótulo fornecidos a CO-TRAINING é denominado L_{ini} e os que não possuem rótulo é denomina U_{ini}. No fim da iteração, o conjunto total de exemplos com rótulo é denominado L_{fim}.

A Tabela 5.4 mostra, para cada conjunto de documentos, se o critério de parada (CP) de 100 iterações foi atingido (s) ou se a execução terminou por não ser possível rotular mais exemplos (n); o número de exemplos com rótulo $|L_{ini}|$ e sem rótulo $|U_{ini}|$ no início da execução; o número médio de exemplos no conjunto de exemplos rotulados $|L_{fim}|$; o número médio de exemplos rotulados errados ($\#Errados$); a proporção de exemplos rotulados errados ($\%Errados$), a qual é calculada pela Equação 5.1;

$$\%Errados = \frac{\#Errados}{|L_{fim}| - |L_{ini}|} \tag{5.1}$$

os números entre parênteses indicam o desvio padrão dessas médias. Finalmente, a coluna Figuras indica, respectivamente, os gráficos que mostram, para cada conjunto de documentos, a média da proporção do número de exemplos rotulados errados com relação ao número total de exemplos rotulados, até essa iteração[3]. Note que essa proporção é calculada sobre o conjunto de exemplos rotulados L que, é incrementada a cada iteração.

Conj. Doc.	CP	$\|L_{ini}\|$	$\|U_{ini}\|$	$\|L_{fim}\|$	$\#Errados$	$\%Errados$	Figuras
NEWS	s	36	684	432.0 (0.0)	1.9 (1.6)	0.5%(0.4)	5.3
LNAI	n	17	339	276.8 (4.6)	9.5 (3.3)	3.7% (1.2)	5.4
COURSE	s	45	888	327.6 (23.2)	21.3 (13.7)	7.5% (4.5)	5.5

Tabela 5.4: Experimento 1; resumo do número de exemplo rotulados errados inseridos em L

Considerando a porcentagem de exemplos rotulados errados no fim da execução de CO-TRAINING ($\%Errados$ na Tabela 5.4), o melhor resultado foi atingido com o conjunto NEWS (0.5%), e o pior resultado com o conjunto COURSE (7.5%). Na Figura 5.2 é mostrado o número médio de exemplos rotulados

[3]Não são considerados nessa proporção o número inicial $|L_{ini}|$ de exemplos com rótulo fornecidos inicialmente ao algoritmo, mas somente os exemplos por ele rotulados.

errados e inseridos em L em cada iteração, para cada um dos três conjuntos.

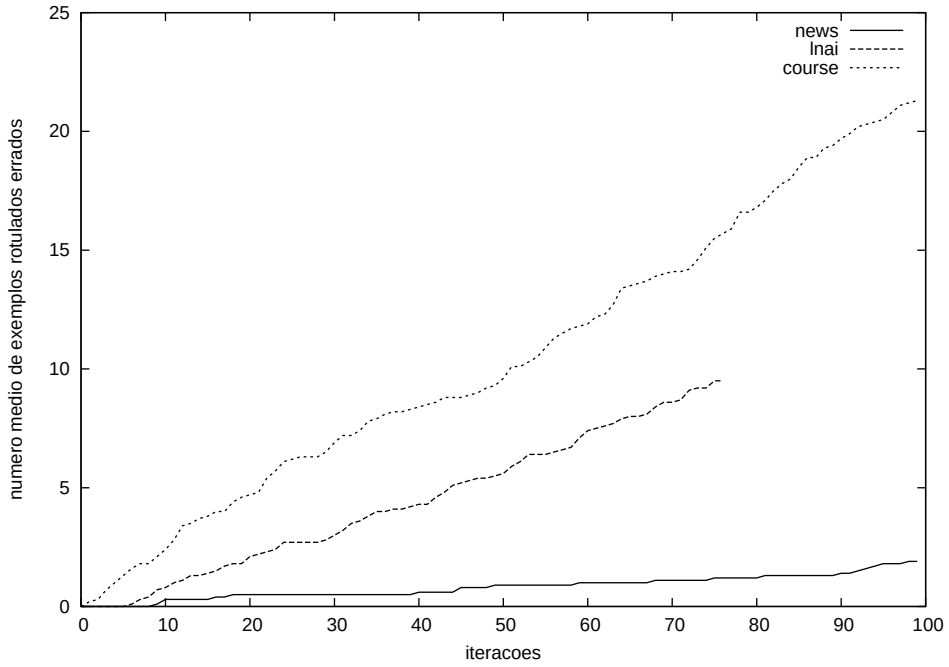


Figura 5.2: Experimento 1; número médio de exemplos rotulados errados e inseridos em L

No caso do conjunto NEWS, esse número mantém-se constantemente muito baixo, iniciando aproximadamente na 8^a iteração, sendo que na maioria das iterações nenhum exemplo é rotulado errado. Já no conjunto LNAI, há exemplos rotulados errados a partir da 5^a iteração aproximadamente, enquanto que para o conjunto COURSE isso acontece a partir da primeira iteração.

Com relação ao critério de parada de 100 iterações, somente não foi atingido pelo conjunto LNAI. Considerando o valor médio de $|L_{fim}|$ para esse conjunto, é possível concluir que CO-TRAINING finalizou a execução pela impossibilidade de rotular mais exemplos com o valor mínimo de confiança 0.6.

Na Figura 5.3 é possível observar que a porcentagem de exemplos rotulados errados inseridos em L tende a se estabilizar em 0.4% no caso do conjunto NEWS.

Para o conjunto LNAI — Figura 5.4 — após a iteração 15, aproximadamente, essa porcentagem cresce lentamente até chegar a 3.7% quando a execução pára por não ser possível rotular mais exemplos.

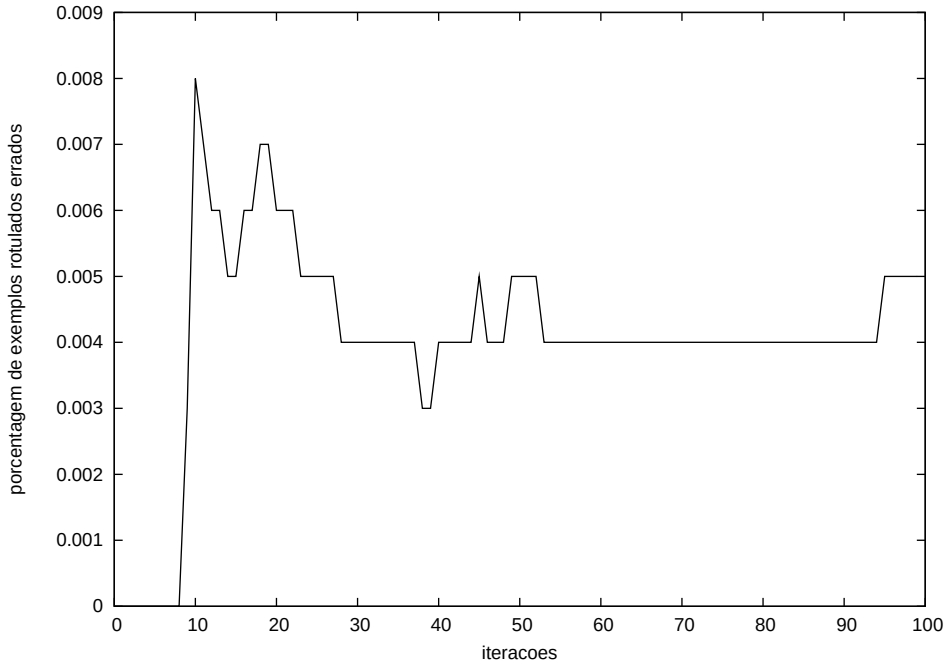


Figura 5.3: Experimento 1 – conjunto NEWS; porcentagem de exemplos rotulados errados inseridos em L

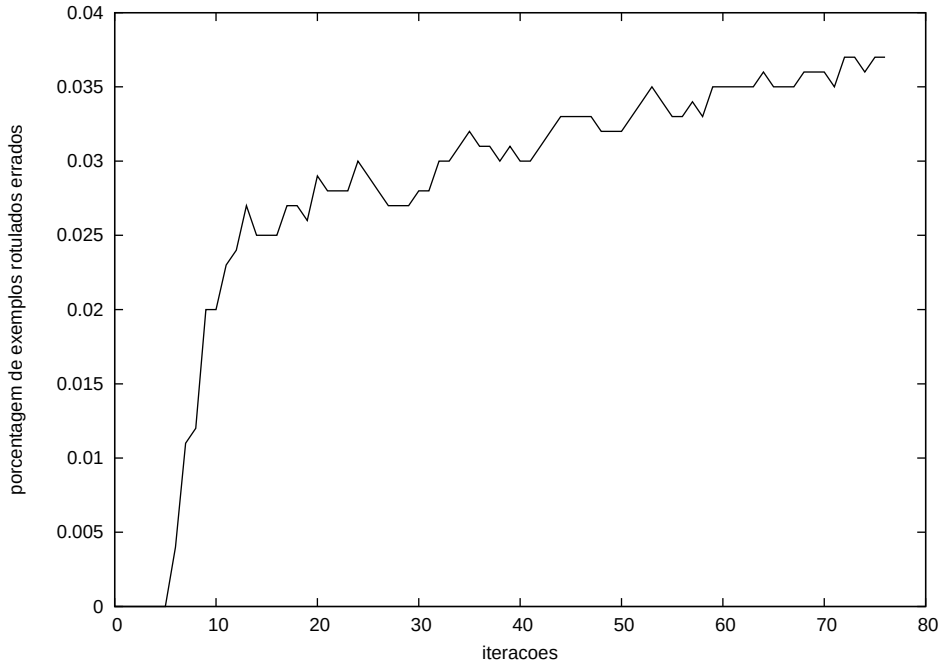


Figura 5.4: Experimento 1 – conjunto LNAI; porcentagem de exemplos rotulados errados inseridos em L

No caso do conjunto COURSE — Figura 5.5 — a porcentagem de exemplos rotulados errados inseridos em L é alta desde o início, sendo maior que 6% antes da 10^a iteração. Após a iteração 30, aproximadamente, essa porcentagem parece se estabilizar em 7.5%.

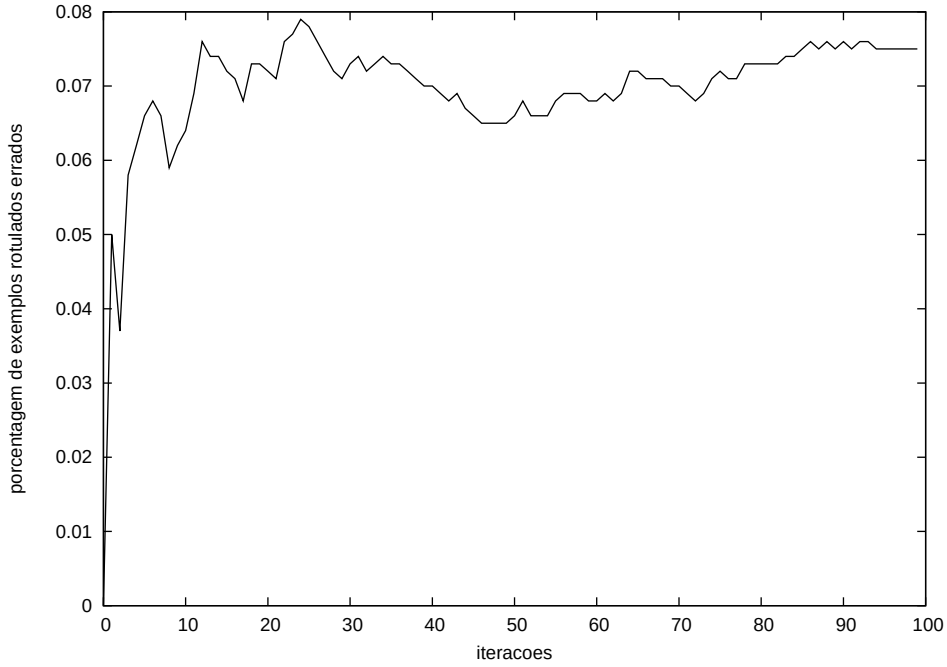


Figura 5.5: Experimento 1 – conjunto COURSE; porcentagem de exemplos rotulados errados inseridos em L

Considerando os valores médios de $|L_{fim}|$ na Tabela 5.4 na página 53, é possível estimar a proporção de exemplos que CO-TRAINING não consegue rotular

antes de atingir o critério de parada. Observe que neste experimento o número máximo de exemplos a ser rotulados em k iterações é dado por $4 \times k$, já que são rotulados, em cada iteração, dois exemplos de cada classe. Para o conjunto NEWS, não foram rotulados 4 exemplos, *i.e.*, (400 - $|L_{fim}|$ + $|L_{ini}| = 400-432+36$), ou seja 1% dos exemplos. Para o conjunto LNAI ($k = 75$), não foram rotulados, em média, 40 exemplos (300 - 277 + 17), ou seja, mais de 13% dos exemplos, enquanto que esse número sobe para 117 exemplos (400 - 328 + 45) para o conjunto COURSE, o que representa mais de 29% dos exemplos.

Assim, considerando os exemplos rotulados errados e a capacidade para rotular mais exemplos, pode-se concluir que CO-TRAINING atingiu excelentes resultados com o conjunto NEWS, resultados bons com o conjunto LNAI, enquanto que com o conjunto COURSE os resultados deixam a desejar. A seguir é analisado o comportamento do CO-TRAINING como um algoritmo de classificação.

A Tabela 5.5 mostra, para cada conjunto de documentos, o erro médio na primeira e última iteração dos classificadores induzidos, h_{D_1} e h_{D_2}, e do classificador combinado h; o número entre parênteses indica o desvio padrão. Observe que na primeira iteração do CO-TRAINING os classificadores são construídos utilizando os exemplos dos quais é conhecido o rótulo. O número desses exemplos é dado por $|L_{ini}|$ na Tabela 5.4 na página 53. A última coluna, Figura, da Tabela 5.5, indica os gráficos que mostram, para cada conjunto de documentos, o erro médio dos classificadores h_{D_1}, h_{D_2} e h em cada iteração.

Conj. Doc.	h_{D_1}	h_{D_2}	h	Figura
NEWS	1-*gram*	2-*gram*		
primeira it.	8.5 (4.6)	3.9 (2.7)	6.0 (3.8)	5.6
última it.	2.2 (1.3)	1.9 (1.2)	1.4 (1.2)	
LNAI	1-*gram*	2-*gram*		
primeira it.	8.3 (4.9)	10.4 (4.3)	7.6 (4.6)	5.7
última it.	4.3 (3.4)	4.3 (2.9)	3.0 (2.3)	
COURSE	TEXTO	LINKS		
primeira it.	12.8 (3.5)	16.0 (5.4)	11.4 (4.6)	5.8
última it.	12.9 (6.6)	18.1 (5.8)	12.6 (6.2)	

Tabela 5.5: Experimento 1; erro médio dos classificadores induzidos na primeira e última iterações

Para o conjunto NEWS — Figura 5.6 — que teve o maior número de exemplos rotulados e o menor número de exemplos rotulados errados, há um decréscimo acentuado do erro dos classificadores, se estagnando a partir da iteração 60 aproximadamente. É interessante comparar esses resultados com os obtidos pelos classificadores induzidos utilizando todos os exemplos rotulados — Tabela 5.3 na página 52. Uma primeira observação é que o erro dos classificadores final combinado h induzido por CO-TRAINING (1.4 na Tabela 5.5) é semelhante aos classificadores induzidos por *Naive Bayes* (1.6 para a des-

crição 1-*gram* e 1.3 para a descrição 2-*gram* na Tabela 5.3). Também, pode ser observado na Figura 5.6 que o erro de h_{D_1} (descrição 1-*gram*) é maior que o erro de h_{D_2} (descrição 2-*gram*), o que também acontece com o erro dos classificadores induzidos tanto por $\mathcal{C}4.5$ quanto por *Naive Bayes* utilizando todos os exemplos rotulados — 5.3.

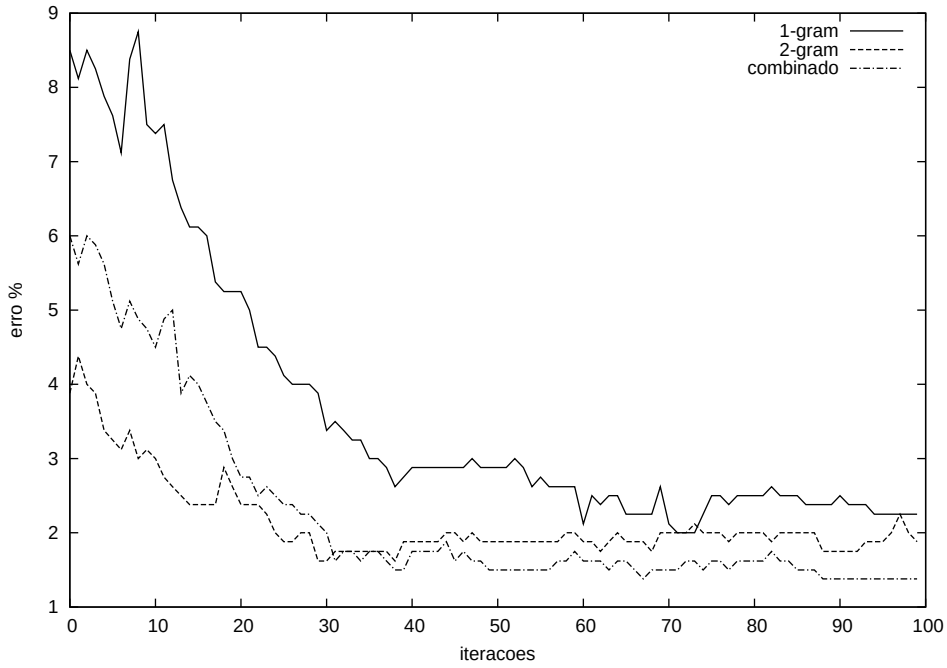


Figura 5.6: Experimento 1 – conjunto NEWS; erro dos classificadores

No caso do conjunto LNAI — Figura 5.7 — o decréscimo no erro dos classificadores não é tão acentuado. Após a iteração 30 há uma tendência do erro de h se manter dentro de um intervalo, mas esse intervalo é bem maior que no caso anterior, com variância 2.3 na última iteração. Com relação ao erro de h na última iteração (3.0 na Tabela 5.5) ele é maior que o erro de *Naive Bayes* utilizando todos os exemplos rotulados (1.5 para a descrição 1-*gram* e 1.8 para a descrição 2-*gram* na Tabela 5.3). Ainda que o erro de h é menor que o dos classificadores induzidos por $\mathcal{C}4.5$ (8.1 para a descrição 1-*gram* e 10.9 para a descrição 2-*gram* na Tabela 5.3), o que pode ser observado é que o *bias* utilizado por $\mathcal{C}4.5$ para induzir árvores de decisão não é apropriado para o conjunto LNAI.

O pior resultado é obtido com o conjunto COURSE — Figura 5.8, para o qual foi rotulado o menor número de exemplos e que teve o maior número de exemplos rotulados errados. É possível observar que neste caso praticamente não houve melhoria na precisão dos classificadores. Além disso, o erro dos classificadores induzidos utilizando a descrição LINKS está muito perto do erro majoritário (21%). Observando os resultados obtidos por $\mathcal{C}4.5$ e *Naive Bayes* utilizando todos os exemplos — Tabela 5.3 — é possível notar que a informação fornecida pelos exemplos desse conjunto não é suficiente para

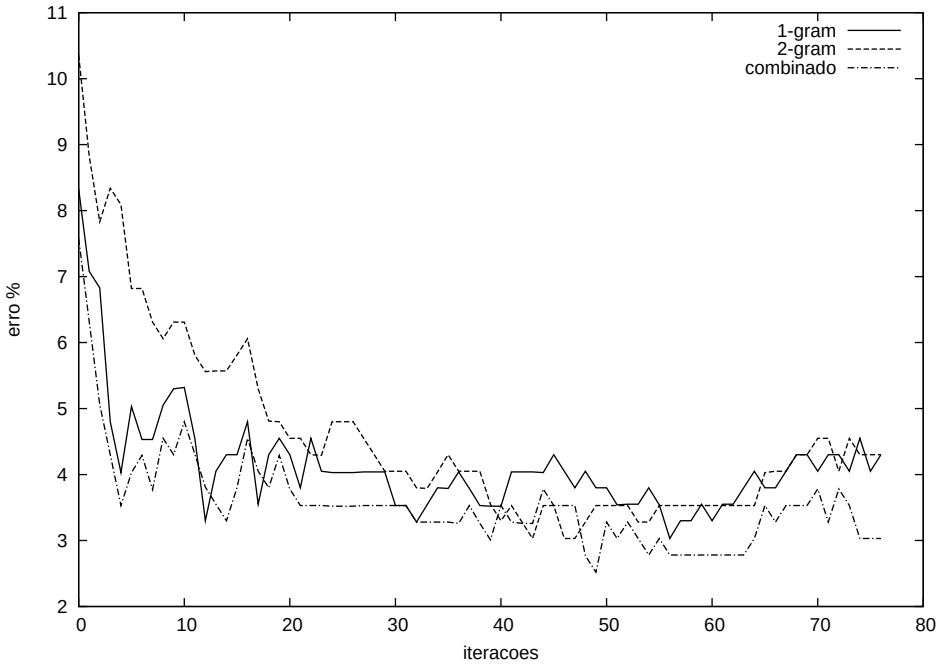


Figura 5.7: Experimento 1 – conjunto LNAI; erro dos classificadores

CO-TRAINING induzir um bom classificador. Observe que nesse caso, o *bias* de $\mathcal{C}4.5$ parece mais apropriado para a descrição LINKS, já que o erro é menor que o de $\mathcal{C}4.5$ para a descrição TEXTO (6.5 e 10.9 na Tabela 5.3). Em outras palavras, caso o *bias* de *Naive Bayes* não for apropriado para ambas visões do conjunto de exemplos, CO-TRAINING não terá um bom comportamento.

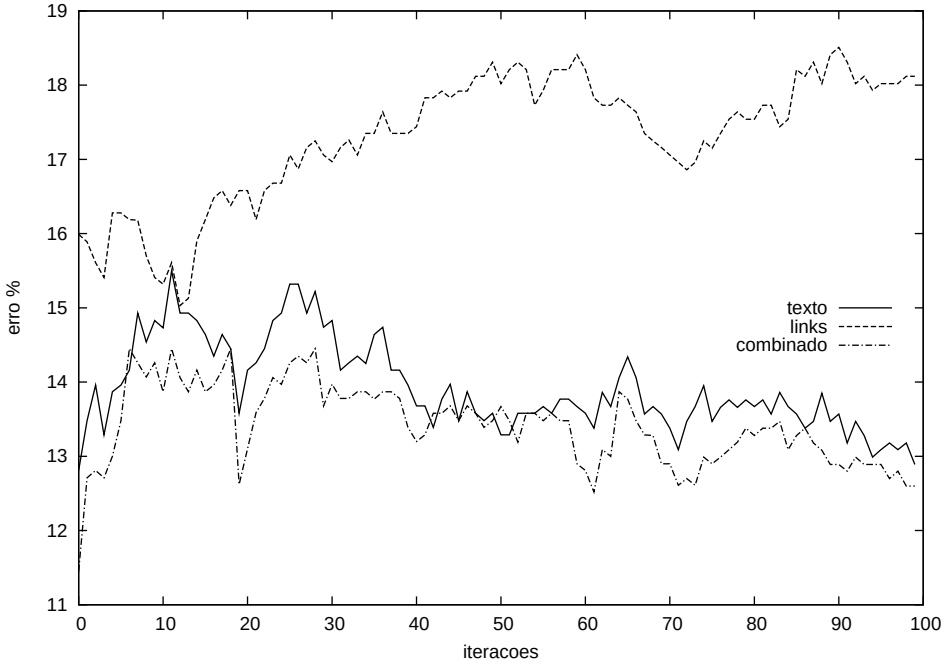


Figura 5.8: Experimento 1 – conjunto COURSE; erro dos classificadores

Finalmente, pode também ser observado que nos três casos, o número de exemplos rotulados errados está refletido nos classificadores induzidos.

5.2.2 Experimento 2: Variando o Tamanho de L_{ini}

É a partir do pequeno conjunto de exemplos inicial com rótulo L_{ini} que, iterativamente, e com o auxílio de duas descrições, CO-TRAINING rotula mais exemplos. Assim, a qualidade e a quantidade desses exemplos são importantes para o algoritmo. O objetivo deste experimento é verificar o comportamento de CO-TRAINING considerando a quantidade de exemplos iniciais com rótulo fornecida a ele. O algoritmo foi executado variando a porcentagem do conjunto de exemplos inicial, mas mantendo os valores padrão dos outros parâmetros. Os resultados obtidos referentes à rotulação de exemplos encontram-se na Tabela 5.6

C.Doc.	CP	% de $\|L_{ini}\|$	$\|U_{ini}\|$	$\|L_{fim}\|$	$\#Errados$	$\%Errados$
NEWS	s	1% 6	714	402.0 (0.00)	2.4 (0.9)	0.6% (0.2)
		2% 14	706	409.3 (2.21)	1.9 (0.9)	0.5% (0.2)
		5% 36	684	432.0 (0.00)	1.9 (1.5)	0.5% (0.4)
		7% 50	670	446.0 (0.00)	2.4 (0.9)	0.6% (0.2)
		10% 72	648	468.0 (0.00)	2.6 (0.6)	0.7% (0.2)
LNAI	n	2% 6	350	275.7 (2.9)	11.0 (2.5)	4.1% (0.9)
		5% 17	339	276.8 (4.6)	9.5 (3.3)	3.7% (1.2)
		7% 24	332	276.0 (3.4)	3.5 (7.1)	2.8% (1.1)
		10% 34	322	279.8 (1.7)	7.4 (1.8)	3.0% (0.8)
COURSE	s	2% 17	917	273.5 (29.5)	57.3 (26.0)	23.1% (11.9)
		5% 45	888	327.6 (23.2)	21.3 (13.7)	7.5% (4.5)
		7% 64	869	338.6 (18.1)	21.9 (21.0)	7.9% (7.4)
		10% 92	841	372.2 (9.5)	11.1 (3.5)	4.0% (1.3)

Tabela 5.6: Experimento 2; variando o tamanho do conjunto L_{ini}

Para o conjunto NEWS, a porcentagem de exemplos rotulados errados ($\%Errados$) é praticamente a mesma independentemente do número de exemplos em L_{ini}. Esse resultado é obtido também com o conjunto LNAI, mas, considerando a variância, os resultados são menos estáveis do que os obtidos para o conjunto NEWS. Já para o conjunto COURSE, há uma acentuada diferença na porcentagem de exemplos rotulados errados quando é incrementado o número de exemplos em L_{ini}.

Isso fica mais claro nas Figuras 5.9, 5.10 e 5.11, nas quais são mostrados os erros dos classificadores combinados h para as diferentes proporções de exemplos em L_{ini} em cada iteração. A Tabela 5.7 mostra o erro médio na primeira e última iteração dos classificadores h_{D_1}, h_{D_2} e h.

Conj. Doc.	% de $\|L_{ini}\|$	h_{D_1}	h_{D_2}	h
NEWS		1-*gram*	2-*gram*	
primeira it.	1%	23.6 (9.8)	19.4 (9.2)	20.1 (11.1)
última it.		1.9 (1.9)	1.5 (1.0)	1.6 (1.4)
primeira it.	2%	13.9 (10.0)	15.0 (5.0)	12.4 (9.3)
última it.		2.0 (1.20)	1.5 (1.1)	1.4 (0.9)
primeira it.	5%	8.5 (4.6)	3.9 (2.7)	6.0 (3.8)
última it.		2.2 (1.3)	1.9 (1.2)	1.4 (1.2)
primeira it.	7%	5.4 (2.3)	2.7 (1.8)	2.7 (1.6)
última it.		2.0 (1.7)	1.9 (1.6)	1.4 (1.1)
primeira it.	10%	6.7 (4.0)	2.0 (2.0)	4.7 (2.7)
última it.		1.9 (1.2)	1.5 (1.3)	1.1 (0.9)
LNAI		1-*gram*	2-*gram*	
primeira it.	2%	13.4 (7.7)	20.0 (8.0)	11.1 (6.4)
última it.		5.3 (4.4)	4.0 (3.0)	3.0 (3.3)
primeira it.	5%	8.3 (4.9)	10.3 (4.3)	7.6 (4.6)
última it.		4.3 (3.4)	4.3 (2.9)	3.0 (2.3)
primeira it.	7%	6.6 (4.6)	8.3 (4.0)	5.8 (4.1)
última it.		4.0 (3.8)	3.5 (2.5)	3.0 (2.6)
primeira it.	10%	5.0 (3.9)	7.6 (2.9)	4.5 (3.7)
última it.		3.3 (4.5)	4.0 (3.3)	3.0 (3.3)
COURSE		TEXTO	LINKS	
primeira it	2%	18.7 (4.5)	19.4 (6.5)	18.7 (4.6)
última it		22.0 (8.6)	24.5 (5.8)	22.1 (7.9)
primeira it.	5%	12.8 (3.5)	16.0 (5.4)	11.4 (4.6)
última it.		12.9 (6.6)	18.0 (5.8)	12.6 (6.2)
primeira it.	7%	11.6 (2.3)	11.0 (3.6)	10.7 (2.3)
última it.		12.0 (7.5)	19.0 (5.8)	11.3 (7.2)
primeira it.	10%	10.3 (3.3)	14.7 (6.8)	8.4 (3.3)
última it.		9.8 (3.2)	18.0 (6.4)	8.1 (3.3)

Tabela 5.7: Experimento 2; erro médio dos classificadores na primeira e última iteração dos classificadores com tamanhos diferentes de L_{ini}

Para o conjunto NEWS (Figura 5.9) a partir de 40^a iteração os erros dos classificadores combinados se tornam praticamente iguais para qualquer tamanho de $|L_{ini}|$.

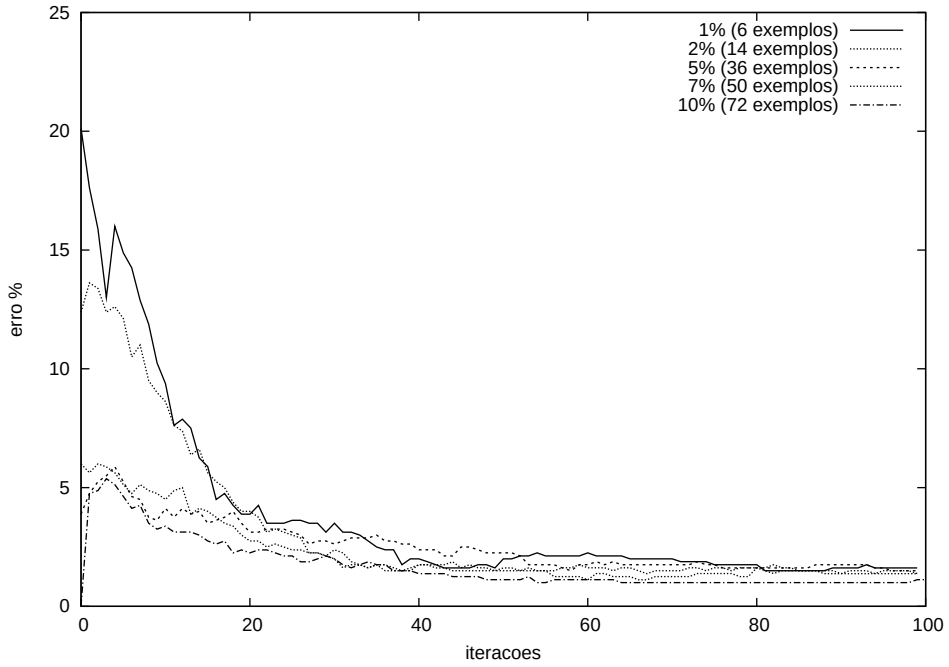


Figura 5.9: Experimento 2 – conjunto NEWS; erro do classificador combinado variando o tamanho do conjunto L_{ini}

O mesmo pode ser observado para o conjunto LNAI (Figura 5.10), no qual a partir da 8^a iteração os resultados se tornam muito semelhantes. Provavelmente deva existir para cada conjunto de exemplos um valor mínimo aceitável para esse comportamento.

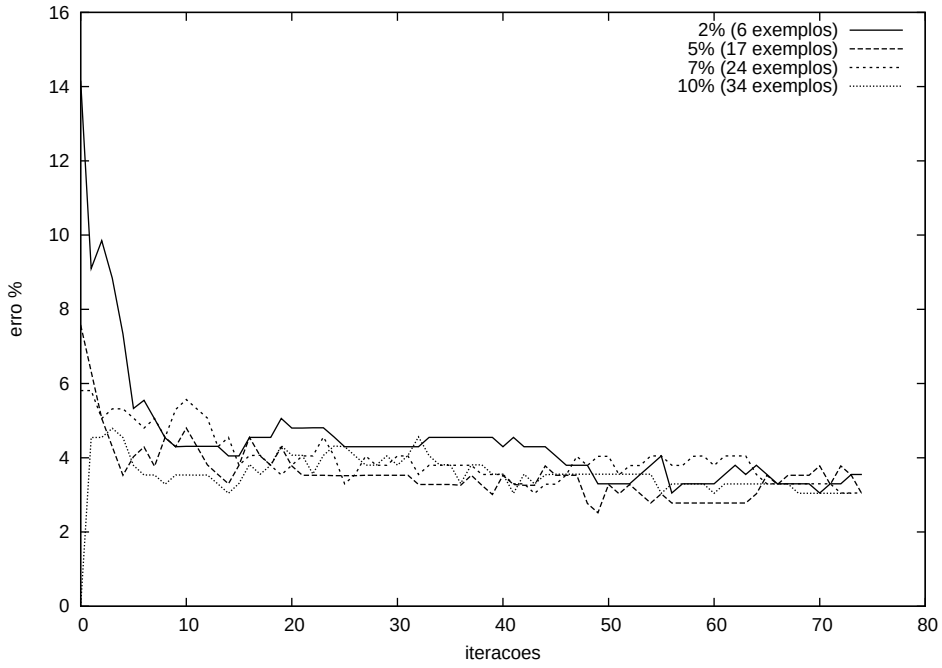


Figura 5.10: Experimento 2 – conjunto LNAI; erro do classificador combinado variando o tamanho do conjunto L_{ini}

Novamente, o conjunto COURSE teve comportamento diferente das outras duas bases. Como pode-se observar na Figura 5.11, ainda que o erro dos classificadores diminui quanto mais exemplos rotulados encontram-se disponíveis, o erro de cada classificador individual praticamente não se altera durante as iterações, não diminuindo, nem aumentando. Entretanto, isso pode ser considerado um bom resultado, já que foram rotulados uma quantidade expressiva de exemplos, o erro dos classificadores não aumentou e o valor de $\%Errados$ diminui quase na mesma proporção que o conjunto de exemplos em L_{ini} aumenta. Aparentemente, em um exemplo real de uso, poderiam ser rotulados mais exemplos até atingir uma precisão razoável.

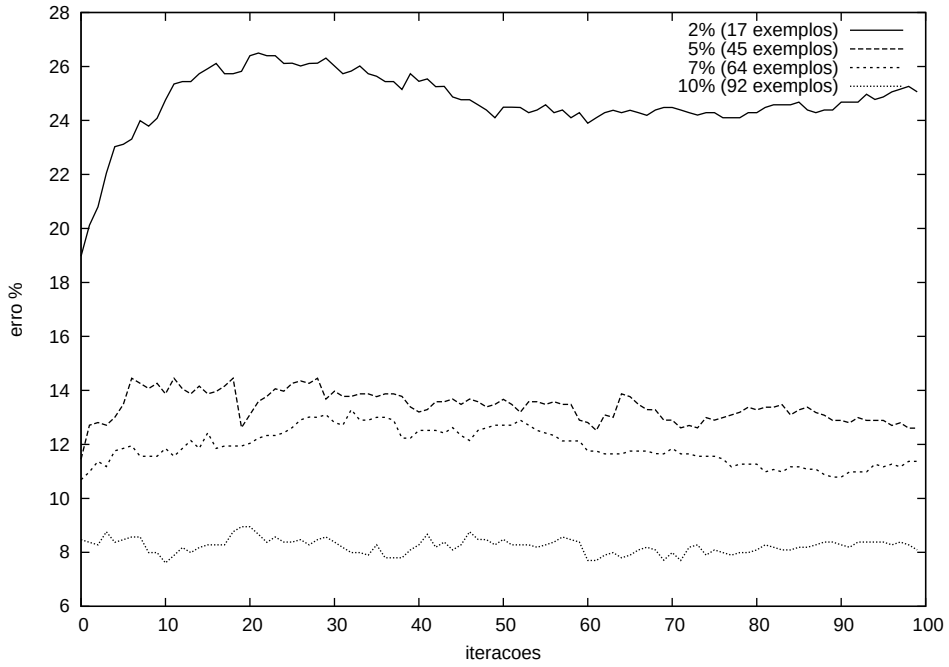


Figura 5.11: Experimento 2 – conjunto COURSE; erro do classificador combinado variando o tamanho do conjunto L_{ini}

5.2.3 Experimento 3: Parâmetros `original` e `nbest`

Como descrito na Seção 3.2.3 na página 30, é possível escolher a função $melhores/2$ que seleciona os exemplos a serem rotulados em cada iteração: `original` e `nbest`. Ao utilizar o parâmetro `original` são rotulados os melhores exemplos de cada classe, segundo o número de exemplos por classe definidos pelo usuário — valor padrão de 2 (dois) exemplos por classe —, enquanto que ao utilizar o parâmetro `nbest` não é levado em conta a classe desses exemplos — valor padrão de 4 (quatro) exemplos. Vale ressaltar que nem sempre o algoritmo consegue rotular esse número de exemplos em cada iteração (ver parâmetro `-force` na Seção 3.2.3). Os resultados obtidos encontram-se nas Tabelas 5.8 e 5.9

Conj. Doc.	CP	$\|L_{ini}\|$	$\|U_{ini}\|$	$melhores/2$	$\|L_{fim}\|$	$\#Errados$	$\%Errados$
NEWS	s	36	684	`original`	432.0 (0.0)	1.9 (1.5)	0.5% (0.4)
				`nbest`	432.0 (0.0)	36.8 (58.2)	9.3% (14.7)
LNAI	n	17	339	`original`	276.8 (4.6)	9.5 (3.3)	3.7% (1.2)
				`nbest`	321.0 (0.0)	1.5 (1.1)	0.5% (0.4)
COURSE	s	45	888	`original`	327.6 (23.2)	21.3 (13.7)	7.5% (4.5)
				`nbest`	418.2 (55.1)	32.2 (22.4)	9.3% (7.4)

Tabela 5.8: Experimento 3; parâmetros `nbest` e `original`

Conj. Doc.	$melhores/2$	h_{D_1}	h_{D_2}	h
NEWS		1-*gram*	2-*gram*	
primeira it.	`original`	8.5 (4.6)	3.8 (2.7)	6.0 (3.8)
última it.		2.2 (1.3)	1.8 (1.2)	1.4 (1.2)
primeira it.	`nbest`	10.5 (6.3)	4.0 (2.5)	6.1 (4.1)
última it.		10.1 (15.1)	10.2 (15.0)	9.9 (15.0)
LNAI		1-*gram*	2-*gram*	
primeira it.	`original`	8.3 (4.9)	10.3 (4.3)	7.6 (4.6)
última it.		4.3 (3.4)	4.3 (2.9)	3.0 (2.3)
primeira it.	`nbest`	8.3 (3.7)	11.3 (4.4)	7.1 (4.2)
última it.		2.5 (2.9)	2.5 (2.7)	2.3 (2.6)
COURSE		TEXTO	LINKS	
primeira it.	`original`	12.8 (3.5)	16.0 (5.4)	11.4 (4.6)
última it.		12.9 (6.6)	18.1 (5.8)	12.6 (6.2)
primeira it.	`nbest`	14.0 (5.7)	13.4 (5.1)	13.0 (5.5)
última it.		14.9 (10.0)	20.5 (12.5)	13.7 (9.6)

Tabela 5.9: Experimento 3; erro médio na primeira e última iteração dos classificadores com os parâmetros `nbest` e `original`

Apesar do conjunto NEWS, que tem as classes balanceadas, ter tido os melhores resultados nos experimentos anteriores, com o parametro `nbest`. Esse conjunto teve o pior resultado (Figura 5.12). Observando as classes dos exemplos rotulados em cada iteração de CO-TRAINING, foi possível verificar que `nbest` rotula mais exemplos da classe `talk` que da classe `sci`. Como pode ser observado na Tabela 5.3 na página 52, o erro na classe `talk` é menor que na classe `sci`, o que parece justificar esse comportamento de CO-TRAINING. Em

outras palavras, ainda que o classificador h_{D_2} induzido usando a descrição `talk` tenha um bom desempenho, o classificador h_{D_1} sofre uma degradação, o que se reflete no erro do classificador combinado h.

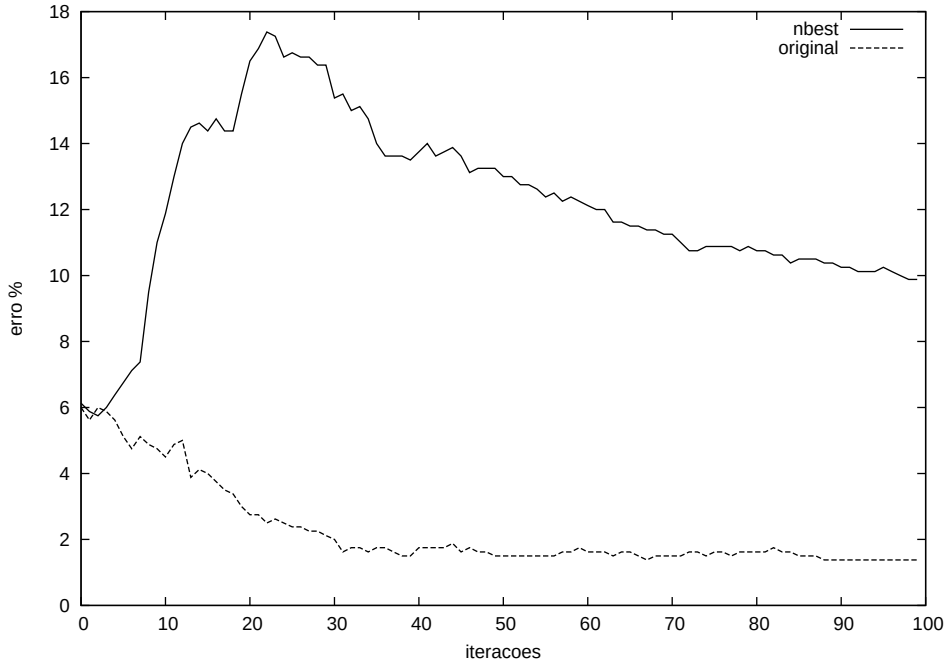


Figura 5.12: Experimento 3 – conjunto NEWS; erro do classificador combinado com o parâmetro `nbest`

Com o conjunto LNAI, que tem classes desbalanceadas na proporção 30-70, foi observado que, ainda que as classes dos exemplos rotulados em cada iteração de CO-TRAINING varia, há uma tendência a rotular mais exemplos da classe majoritária (CBR). Com `nbest` CO-TRAINING consegue rotular mais exemplos do conjunto LNAI com poucos exemplos rotulados errados — Tabela 5.8 — e melhora a precisão do classificador combinados — Tabela 5.9 na página anterior e Figura 5.13 na próxima página.

O conjunto COURSE também tem classes desbalanceadas na proporção 21-79. No entanto, o resultado é diferente que para o conjunto LNAI, ainda que ao verificar a mesma tendência que no caso anterior, de CO-TRAINING rotular mais exemplos da classe majoritária — Figura 5.14 — o que confirma mais uma vez que esta base não é apropriada para CO-TRAINING.

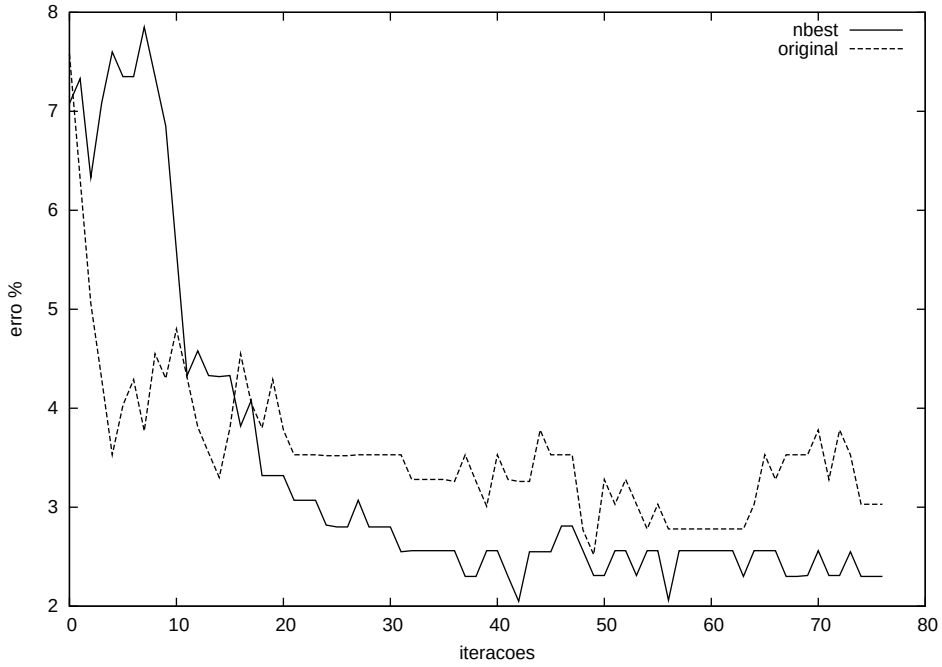


Figura 5.13: Experimento 3 – conjunto LNAI; erro do classificador combinado com o parâmetro `nbest`

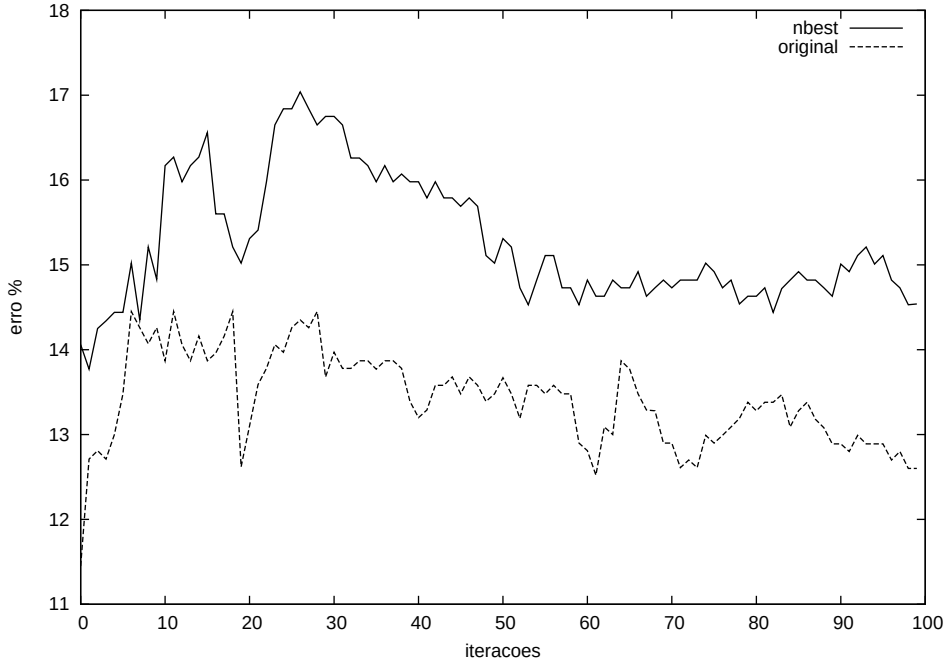


Figura 5.14: Experimento 3 – conjunto COURSE; erro do classificador combinado com o parâmetro `nbest`

5.2.4 Experimento 4: Parâmetro `original` Variando o Número de Exemplos a Serem Rotulados em Cada Classe

Considerando que a proporção de exemplos rotulados de cada classe influi nos resultados, e que o número de exemplos rotulados em cada iteração de CO-TRAINING pode ser fixado usando `original`, foi decidido avaliar o desempenho de CO-TRAINING no melhor caso, rotulando em cada classe um número de exemplos proporcional a proporção das classes no conjunto de dados, e no pior caso, utilizando a proporção inversa das classes no conjunto de dados.

Na Tabela 5.10 são apresentados os resultados obtidos. Como pode ser observado, o número de exemplos de cada classe a ser rotulado em cada iteração é um parâmetro importante na rotulação de exemplos. Em todos os casos, ao rotular um número de exemplos cuja proporção é semelhante à proporção das classes dos conjuntos de documentos, foram obtidos os melhores resultados — em negrito na Tabela 5.10 e 5.11. Entretanto, para realizar este tipo de ajuste, é necessário conhecer essa proporção, o que raramente é possível no caso real de execução de CO-TRAINING pois, nesse caso, somente é conhecido o rótulo de um conjunto pequeno de exemplos. Calcular a proporção das classes com base nesse número pequeno de exemplos, forneceria um resultado pouco confiável.

Conj. Doc.	CP	$\|L_{ini}\|$	$\|U_{ini}\|$	(maj,min)	$\|L_{fim}\|$	$\#Errados$	$\%Errados$
NEWS	s	36	684	(4,2)	558.0 (5.2)	4.9 (3.7)	0.9% (0.7)
				(2,2)	432.0 (0.0)	1.9 (1.5)	**0.5% (0.4)**
				(2,4)	587.9 (9.9)	28.30 (11.0)	5.1% (1.9)
LNAI	n	17	339	**(4,2)**	312.4 (2.0)	3.0 (0.9)	**1.0% (0.3)**
				(2,2)	276.8 (4.6)	9.5 (3.3)	3.7% (1.2)
				(2,4)	225.6 (5.2)	12.4 (4.7)	5.9% (2.2)
COURSE	s	45	888	(4,1)	498.6 (24.3)	26.8 (24.9)	6.2% (6.3)
				(2,2)	327.6 (23.2)	21.3 (13.7)	7.5% (4.5)
				(1,4)	299.1 (73.3)	73.3 (76.9)	26.0%(18.4)
				(6,2)	753.1 (26.7)	29.7 (10.5)	**4.2% (1.6)**

Tabela 5.10: Experimento 4; parâmetro `original` variando a proporção de exemplos a serem rotulados em cada classe

Com esses experimentos pode-se observar que a proporção das classes é de fundamental importância para a execução do algoritmo CO-TRAINING. Ao utilizar as proporções semelhantes as das bases originais o CO-TRAINING teve melhor desempenho em todas elas.

Finalmente, nas Figuras 5.15 e 5.16 são mostrados para o conjunto LNAI e COURSE, respectivamente, o erro médio dos classificadores h_{D_1}, h_{D_2} e h, para o melhor caso, marcado em negrito nas Tabelas 5.10 e 5.11.

Conj. Doc.	(maj,min)	h_{D_1}	h_{D_2}	h
NEWS		1-*gram*	2-*gram*	
primeira it.	(4,2)	12.6 (10.5)	7.9 (6.9)	9.7 (9.0)
última it.		2.2 (1.4)	2.2 (1.5)	1.7 (1.0)
primeira it.	**(2,2)**	8.5 (4.6)	3.9 (2.7)	6.0 (3.8)
última it.		2.2 (1.3)	1.9 (1.2)	**1.4 (1.2)**
primeira it.	(2,4)	4.4 (1.8)	3.7 (2.8)	3.3 (1.9)
última it.		5.1 (2.1)	6.5 (3.4)	4.7 (2.5)
LNAI		1-*gram*	2-*gram*	
primeira it.	**(4,2)**	7.8 (2.2)	11.1 (3.2)	7.3 (2.8)
última it.		2.3 (3.0)	2.5 (2.1)	**1.8 (2.1)**
primeira it.	(2,2)	8.3 (4.9)	10.4 (4.3)	7.6 (4.6)
última it.		4.3 (3.4)	4.3 (2.9)	3.0 (2.3)
primeira it.	(2,4)	7.6 (3.0)	11.4 (7.9)	7.8 (3.1)
última it.		5.8 (4.8)	5.3 (6.5)	4.5 (4.5)
COURSE		TEXTO	LINKS	
primeira it.	(4,1)	14.8 (3.1)	15.9 (3.8)	13.4 (3.0)
última it.		11.4 (4.5)	19.0 (4.5)	9.6 (5.1)
primeira it.	(2,2)	12.8 (3.5)	16.0 (5.4)	11.4 (4.6)
última it.		12.9 (6.6)	18.1 (5.8)	12.6 (6.2)
primeira it.	(1,4)	13.5 (2.7)	14.7 (3.8)	13.2 (2.9)
última it.		28.2 (17.6)	39.1 (23.6)	28.6 (18.0)
primeira it.	**(6,2)**	11.2 (2.6)	13.5 (3.4)	10.2 (2.0)
última it.		8.7 (3.6)	18.3 (3.9)	**8.8 (4.9)**

Tabela 5.11: Experimento 4; erro médio dos classificadores na primeira e última iteração

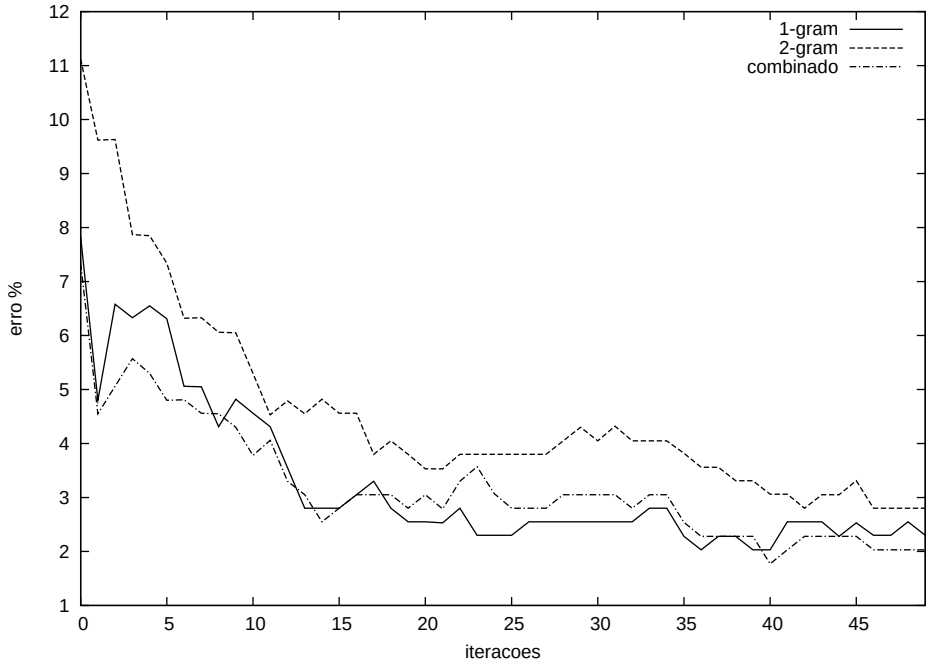


Figura 5.15: Experimento 4 – conjunto LNAI; erro dos classificadores com parâmetro `original` (CBR =4 ILP =2)

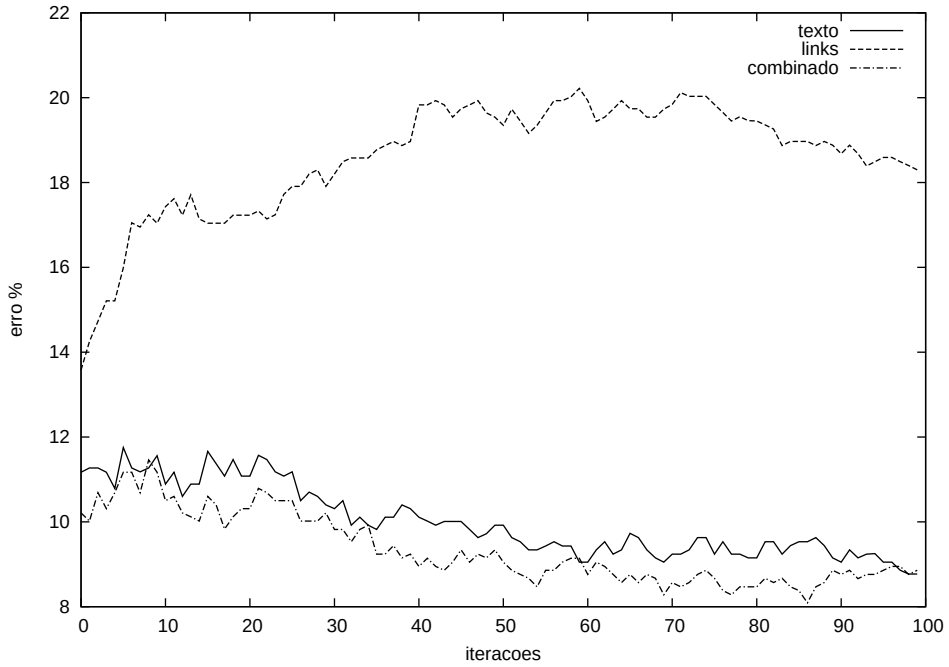


Figura 5.16: Experimento 4 – conjunto COURSE; erro dos classificadores com parâmetro `original` (`non-course` =6 `course` =2)

5.3 Considerações Finais

Neste capítulo foram descritos vários dos experimentos realizados com o objetivo de avaliar CO-TRAINING, utilizando três bases de documentos. Estes e outros experimentos encontram-se descritas em maiores detalhes em (Matsubara & Monard, 2004a).

Nas bases NEWS e LNAI para os quais foi utilizado 1-*gram* e 2-*gram* para construir as duas descrições, os resultados obtidos por CO-TRAINING foram muito bons. Já que a base COURSE, que foi utilizada pelos autores de CO-TRAINING com as descrições TEXTO e LINKS, os resultados não são tão bons.

CAPÍTULO

6

Conclusões e Trabalhos Futuros

Neste trabalho apresentamos o algoritmo de aprendizado semi-supervisionado CO-TRAINING e a sua implementação como parte integrante de um projeto maior em desenvolvimento por pesquisadores do grupo de pesquisa de Inteligência Computacional do ICMC-USP, o projeto DISCOVER, juntamente com a ferramenta de pré-processamento de textos PRETEXT. O algoritmo CO-TRAINING necessita de um pequeno conjunto de dados rotulados e um conjunto maior de dados não rotulados para a sua execução. Ambos os conjuntos de dados devem estar representados por, pelo menos, duas descrições diferentes desses dados. Com as duas descrições iniciais do pequeno conjunto de exemplos rotulados, CO-TRAINING induz dois classificadores que são utilizados para rotular uns poucos exemplos de sua respectiva descrição. Após, os exemplos correspondentes que foram rotulados com a mesma classe por cada um desses classificadores, e que verificam várias condições consideradas importantes para diminuir o erro da classificação desses exemplos, são adicionados ao conjunto de exemplos rotulados e o processo se repete até não ser possível rotular mais exemplos ou outro critério de parada ser atingido.

Assim, CO-TRAINING, como a maioria dos algoritmos de aprendizado semi-supervisionados, pode ser avaliado considerando o erro na rotulação de novos exemplos em cada iteração, como também considerando a precisão dos dois classificadores induzidos e/ou o classificador combinado, como é o caso de CO-TRAINING. Neste trabalho, CO-TRAINING foi avaliado com esses dois pontos de vista.

Para avaliar o algoritmo foram utilizadas três bases de textos. Uma delas, a base COURSE, é a base utilizada por Blum & Mitchell (1998), autores do CO-TRAINING, com a qual foram obtidos os piores resultados. As outras duas

bases, NEWS e LNAI, foram inicialmente pré-processadas utilizando o PRETEXT para obter, entre outros, as duas descrições requeridas por CO-TRAINING. Os resultados obtidos considerando o número de exemplos rotulados errados em cada iteração e a precisão do classificador combinado induzido são muito animadores. Assim, consideramos que CO-TRAINING é adequado para incrementar o número de documentos rotulados quando poucos deles estão disponíveis.

Entretanto, também consideramos que o uso de CO-TRAINING, ou qualquer outro algoritmo de aprendizado semi-supervisionado, que têm como principal objetivo rotular exemplos quando há poucos exemplos rotulados disponíveis, deve ser usado com precaução. O motivo principal é que esses algoritmos podem rotular erroneamente alguns exemplos; como esses exemplos são incorporados ao conjunto de exemplos rotulados e o processo é repetido, esses erros serão propagados nas próximas iterações do algoritmo o qual rotulará com alta probabilidade, mais exemplos errados. Dessa maneira, o classificador resultante será induzido observando exemplos que descrevem erroneamente instâncias do conceito a ser aprendido.

6.1 Principais Contribuições

Consideramos que as principais contribuições deste trabalho são o projeto e implementação de CO-TRAINING e da ferramenta PRETEXT, ambos integrados no projeto DISCOVER e descritos em maiores detalhes em (Matsubara & Monard, 2004b) e (Matsubara, Martins, & Monard, 2003) respectivamente. A ferramenta PRETEXT foi utilizada no desenvolvimento de diversos trabalhos de pesquisa cujos resultados foram publicados em anais de congressos e relatórios técnicos (Martins, Monard, & Matsubara, 2003b,a, 2004; Martins, Godoy, Monard, Matsubara, & Amandi, 2003). Também, está em fase de finalização a publicação de um relatório técnico que descreve em detalhes os experimentos realizados com CO-TRAINING, descritos neste trabalho, e resultados obtidos com vários outros experimentos realizados (Matsubara & Monard, 2004a).

6.2 Trabalhos Futuros

O algoritmo CO-TRAINING é freqüentemente citado na literatura conjuntamente com diversas propostas de variações e melhorias; muito ainda precisa ser investigado para torná-lo melhor. A seguir são apresentadas algumas das nossas propostas de trabalhos futuros a serem realizados com CO-TRAINING.

- a implementação de CO-TRAINING realizada neste trabalho permite a utilização de mais de duas descrições dos dados. Consideramos que utilizar

mais de duas descrições pode resultar em uma melhoria na rotulação de exemplos;

- o projeto da nossa implementação contempla requisitos para a fácil inserção de outros algoritmos de AM além do *Naive Bayes*. Entretanto, o CO-TRAINING exige do classificador um valor que represente a "força" de classificação de novos exemplos. O algoritmo *Support Vector Machines* fornece essa força de classificação pelo mapeamento de suas saídas que se apresentam como distâncias dos exemplos em relação à fronteira de decisão;

- o projeto da implementação realizada também prevê a utilização de CO-TRAINING com algoritmos diferentes. Assim, após inserir o algoritmo SVM, será possível verificar o comportamento de CO-TRAINING usando ambos os algoritmos conjuntamente. Uma outra possibilidade é utilizar ambos algoritmos, *i.e. Naive Bayes* e SVM, mas somente uma descrição dos dados. Em outras palavras, considerando duas descrições idênticas dos dados;

- como já mencionado, consideramos que o uso de CO-TRAINING, ou outro algoritmo de aprendizado semi-supervisionado, deve ser utilizado com cuidado devido a possibilidade de rotular exemplos errados em cada iteração, os quais são utilizados para induzir os classificadores que rotularão mais exemplos, e assim por diante. Em alguns domínios, tais como medicina, a possibilidade desse tipo de erro é inadmissível. Entretanto, o algoritmo CO-TRAINING poderia ser executado interativamente, tal que antes de inserir um novo exemplo no conjunto de exemplos rotulados solicitaria a confirmação de um especialista no domínio;

- a implementação realizada permite fixar a proporção de exemplos de cada classe a ser rotulados em cada iteração de CO-TRAINING. Com essa funcionalidade, é possível fazer um estudo mais aprofundado, utilizando curvas ROC (Provost, Fawcett, & Kohavi, 1998), para encontrar qual é a melhor proporção;

- finalmente, consideramos também interessante implementar uma outra funcionalidade que permita trocar dinamicamente a proporção de exemplos de cada classe rotulados em cada iteração levando em consideração a precisão dos classificadores induzidos em cada iteração de CO-TRAINING.

Referências Bibliográficas

Apté, C., F. Damerau, & S. M. Weiss (1994). Automated learning of decision rules for text categorization. *Information Systems 12*(3), 233–251. http://citeseer.nj.nec.com/apte94automated.html. 36

Banerjee, S. & T. Pedersen (2003). The design, implementation and use of the ngram statistics package. In *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 370–381. 35

Baranauskas, J. A. (2001). Extração Automática de Conhecimento por Múltiplos Indutores. Tese de Doutorado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08102001-112806. 24

Baranauskas, J. A. & G. E. A. P. A. Batista (2000). O Projeto DISCOVER: Idéias Iniciais. Comunicação pessoal. 23

Basu, S., A. Banerjee, & R. Mooney (2002). Semi-supervised clustering by seeding. In *Proceedings of the 19th International Conference on Machine Learning*, pp. 19–26. Morgan Kaufmann. 15

Batista, G. E. A. P. A. (2003). Pré-processamento de Dados em Aprendizado de Máquina Supervisionado. Tese de Doutorado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-06102003-160219/publico/TeseDoutorado.pdf. 50

Batista, G. E. A. P. A. & M. C. Monard (2003). Descrição da Arquitetura e do Projeto do Ambiente Computacional DISCOVER LEARNING ENVIRONMENT — DLE. Technical Report 187, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_187.pdf. 23, 24

Batista, G. E. A. P. A., R. C. Prati, & M. C. Monard (2004). A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data.

SIGKDD Explorations 6(1). Special issue on Learning from Imbalanced Datasets (in print). 52

Bennett, K. & A. Demiriz (1998). Semi-supervised support vector machines. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), *Advances in Neural Information Processing Systems*, Volume 11, pp. 368–374. MIT Press. 17

Bennett, K. P. & A. Demiriz (2002). Exploiting unlabeled data in ensemble methods. In F. Provost & R. Srikant (Eds.), *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining (ACM-SIGKDD)*, pp. 289–296. ACM Press. `http://www.rpi.edu/~demira/assemble.pdf`. 17

Blake, C. & C. Merz (1998). UCI Repository of Machine Learning Databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`. 47

Blum, A. & S. Chawla (2001). Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 19–26. Morgan Kaufmann. 17

Blum, A. & T. Mitchell (1998). Combining labeled and unlabeled data with co-training. In *Proc. 11th Annu. Conf. on Comput. Learning Theory*, pp. 92–100. ACM Press, New York, NY. 4, 19, 22, 48, 73

Brasil, C. (2004). Uma Ferramenta Inteligente de Apoio à Pesquisa: Mineração de Artigos Científicos na WEB. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP. 46, 48

Bruce, R. (2001). A bayesian approach to semi-supervised learning. In *Natural Language Processing Pacific Rim Symposium*, Tokyo, Japan, pp. 57–64. `www.afnlp.org/nlprs2001/pdf/0100-01.pdf`. 17

Cheeseman, P. & J. Stutz (1996). Bayesian Classification (autoclass): Theory and Results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 153–180. Menlo Park, CA: American Association for Artificial Intelligence. 9

Cortes, C. & V. Vapnik (1995). Support-vector networks. *Machine Learning 20*(3), 273–297. 17, 26

Dempster, A. P., N. M. Laird, & D. B. Rubin (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm (with Discussion). *Journal of Royal Statistical Society B39*, 1–38. 17

Dumais, S., J. Platt, D. Heckerman, & M. Sahami (1998). Inductive learning algorithms and representations for text categorization. In *Proceedings of the 7th International Conference on Information and Knowledge Management*

(CIKM 98), pp. 148–155. http://research.microsoft.com/~sdumais/cikm98.pdf. 36

Fung, G. & O. L. Mangasarian (1999). Semi-supervised support vector machines unlabeled data classification. Technical Report 99, Data Mining Institute. www-personal.engin.umich.edu/~hjia/resource/SVM_DataMinging/99-05.pdf. 17

Gelbukh, A. & G. Sidorov (2001). Zipf and heaps laws coefficients depend on language. In *Proceedings CICLing2001, Conference on Intelligent Text Processing and Computational Linguistics, Lecture Notes in Computer Science*, Number 2004, pp. 332–335. Springer-Verlag. http://www.gelbukh.com/CV/Publications/2001/CICLing-2001-Zipf.htm. 41

Geromini, M. R. (2002). Projeto e Desenvolvimento da Interface Gráfica do Sistema DISCOVER. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP. 24

Gomes, A. K. (2002). Análise do Conhecimento Extraído de Classificadores Simbólicos utilizando Medidas de Avaliação e Interessabilidade. Dissertação de Mestrado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-04072002-144610. 12

Ivanov, Y., B. Blumberg, & A. PentLand (2001). Expectation maximization for weakly labeled data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 218–225. Morgan Kaufmann. www.media.mit.edu/characters/additionalresources/icml2001.pdf. 17

Knuth, D. E. (1973). *The Art of Computer Programming*, Volume 3. Addison-Wesley. 41

Kremer, S. C. & D. A. Stacey (2001). Competition: Unlabeled data for supervised learning. http://www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/nips2001.html. 17

Kubat, M., R. Holte, & S. Matwin (1998). Machine Learning for the Detection of Oil Spills in Satellite Radar Images. *Machine Learning 30*, 195–215. 11

Lee, H. D. (2000). Seleção e Construção de Features Relavantes para o Aprendizado de Máquina. Dissertação de Mestrado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-15032002-113112. 35

Lewis, D. D. (1992, June). An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th International*

ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 37–50. 36

Liu, H. & H. Motoda (1998). *Feature Selection for Knowledge and Data Mining*. Massachusetts: Kluwer Academic Publishers. 35

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of Research and Development 2*(2), 159–165. 41

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observation. In *Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Volume 1, Berkeley, pp. 281–297. University of California Press. 15

Martins, C. A. (2003). Uma Abordagem para Pré-processamento de Dados Textuais em Algoritmos de Aprendizado. Tese de Doutorado, ICMC-USP. 13, 24, 46

Martins, C. A., D. Godoy, M. C. Monard, E. T. Matsubara, & A. Amandi (2003). Uma experiência em mineração de textos utilizando clustering probabilístico e clustering conceitual. Technical Report 205, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_205.zip`. 46, 74

Martins, C. A., M. C. Monard, & E. T. Matsubara (2003a). Reducing the dimensionality of bag-of-words text representation used by learning algorithms. In *Proceedings of The Third IASTED International Conference on Artificial Intelligence and Applications (AIA 2003)*, Benalmádena, Espanha, pp. 228–233. Acta Press. 46, 74

Martins, C. A., M. C. Monard, & E. T. Matsubara (2003b). Uma ferramenta computacional para auxiliar no pré-processamento de textos. In R. de Oliveira Anido & P. C. Masiero (Eds.), *Anais do XXIII Congresso da Sociedade Brasileira de Computação - IV Encontro Nacional de Inteligência Artificial (ENIA)*, Campinas, SP. 6 pgs. (CD-ROM). 46, 74

Martins, C. A., M. C. Monard, & E. T. Matsubara (2004). Uma metodologia para auxiliar na seleção de atributos relevantes usados por algoritmos de aprendizado no processo de classificação de textos. In *Proceedings of XXX Conferencia LatinoAmericana de Informatica - CLEI*, La Paz, Bolívia. (in print). 46, 74

Matsubara, E. T., C. A. Martins, & M. C. Monard (2003). Pretext: Uma ferramenta para pré-processamento de textos utilizando a abordagem *bag-of-words*. Technical Report 209, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_209.zip`. 40, 44, 74

Matsubara, E. T. & M. C. Monard (2004a). Análise experimental do algoritmo de aprendizado de máquina semi-supervisionado CO-TRAINING. Technical report, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_235.pdf. 46, 71, 74

Matsubara, E. T. & M. C. Monard (2004b). Projeto e implementação do algoritmo de aprendizado de máquina semi-supervisionado CO-TRAINING. Technical Report 229, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/rel_tec229.zip. 27, 46, 74

McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/~mccallum/bow. 35

Melanda, E. & S. Rezende (2003). Sintaxe Padrão Para Representar Regras de Associação. Technical Report 206, ICMC-USP. 24

Melo, V. (2004). Uma Ferramenta Inteligente de Apoio à Pesquisa: Construção Automática de Clusters de Artigos Científicos. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP. 46, 48

Melo, V., M. Secato, & A. A. Lopes (2003). Extração e Identificação Automática de Informações Bibliográficas de Artigos Científicos. In *IV Workshop on Advances and Trend in AI*, Chile, pp. 1–10. 48

Michalski, R. S., J. G. Carbonell, & T. M. Mitchell (1983). *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company. 11

Milaré, C. R. (2003). Extração de Conhecimento de Redes Neurais Artificiais Utilizando Sistemas de Aprendizado Simbólico e Algoritmos Genéticos. Tese Doutorado, ICMC-USP. 24

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. 7, 9, 17

Monard, M. C. & J. A. Baranauskas (2003). *Conceitos sobre Aprendizado de Máquina* (1 ed.), Chapter 4, pp. 89–114. Volume 1 of *1* Rezende (2003). 11

Nigam, K., A. K. McCallum, S. Thrun, & T. M. Mitchell (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning 39*(2/3), 103–134. 17

Porter, M. (1980). An algorithm for suffix stripping. *Program 14*(3), 130–137. 40

Prati, R. C. (2003). O *Framework* de Integração do Sistema DISCOVER. Dissertação de Mestrado, ICMC-USP. 24

Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001a). Extração de Informações Padronizadas para a Avaliação de Regras Induzidas por Algoritmos de Aprendizado de Máquina Simbólico. Technical Report 145, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip. 24

Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001b). Uma Proposta de Unificação da Linguagem de Representação de Conceitos de Algoritmos de Aprendizado de Máquina Simbólicos. Technical Report 137, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip. 24

Prati, R. C., G. E. A. P. A. Batista, & M. C. Monard (2004). Class Imbalances *versus* Class Overlapping: an Analysis of a Learning System Behavior. In *Mexican International Conference on Artificial Intelligence, LNAI 2972*, pp. 312–321. Springer-Verlag. 52

Provost, F., T. Fawcett, & R. Kohavi (1998). The Case Against Accuracy Estimation for Comparing Induction Algorithms. In *15th International Conference on Machine Learning*, pp. 445–453. Morgan Kaufmann, San Francisco, CA. 75

Pugliesi, J. B. (2004). Pós-Processamento de Regras de Regressão. Tese de Doutorado, ICMC-USP (a ser defendida). 24

Rezende, S. O. (2003). *Sistemas Inteligentes: Fundamentos e Aplicações*. Barueri, SP, Brasil: Editora Manole. 81

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psycological Review 65*, 386–40. 2

Sanches, M. K. (2003). Aprendizado de máquina semi-supervisionado: proposta de um algoritmo para rotular exemplos a partir de poucos exemplos rotulados. Dissertação de Mestrado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-12102003-140536. 16

Sebastiani, F. (2002, March). Machine learning in automated text categorisation. *ACM Computing Surveys 34*(1), 1–47. http://faure.iei.pi.cnr.it/~fabrizio/Publications/ACMCS02.pdf. 36

Seeger, M. (2002). Covariance kernels from bayesian generative models. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems 14*, pp. 905–912. MIT Press. 17

Van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow. http://citeseer.nj.nec.com/vanrijsbergen79information.html. 42

Wagstaff, K., C. Cardie, S. Rogers, & S. Schroedl (2001). Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 577–584. 15

Wall, L., T. Christiansen, & R. L. Schwartz (1996). *Programming Perl* (2 ed.). O'Reilly & Associates. 23, 42

Zipf, G. (1949). *Human Behaviour and the Principle of Least Effort*. Addison-Wesley. 40