Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Projeto e implementação do algoritmo semissupervisionado multidescrição CoAL

Ígor Assis Braga
Maria Carolina Monard

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Maio/2009

Projeto e implementação do algoritmo semissupervisionado multidescrição CoAL

Ígor Assis Braga
Maria Carolina Monard

[1]Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e de Computação – ICMC
Laboratório de Inteligência Artificial – LABIC
Av. Trabalhador São-carlense, 400 – Centro
Caixa Postal: 668 – CEP: 13560-970 – São Carlos – SP – Brasil

igorab@icmc.usp.br, mcmonard@icmc.usp.br

Resumo. *Algoritmos de aprendizado semissupervisionado podem ser aplicados em domínios em que poucos exemplos rotulados e uma vasta quantidade de exemplos não rotulados estão disponíveis. Uma poderosa abordagem ao aprendizado semissupervisionado, denominada aprendizado multidescrição, pode ser usada sempre que os exemplos de treinamento são descritos por dois ou mais conjuntos de atributos disjuntos. O* Co-Training *é o principal algoritmo semissupervisionado multidescrição disponível atualmente. No entanto, ele apresenta um problema que pode surgir quando os classificadores em cada descrição discordam com alta confiança na classificação de um exemplo. Em (Braga, 2010) foram propostas e avaliadas duas soluções para lidar com esse problema. Uma dessas soluções é o algoritmo* CoAL*, cuja implementação usando o ambiente Weka é descrita neste relatório.*

Sumário

1. Introdução

Algoritmos de aprendizado supervisionado geralmente necessitam que um número expressivo de exemplos rotulados estejam disponíveis para a obtenção de um classificador com boa capacidade preditiva. No entanto, em vários domínios de aplicação importantes, a aquisição de exemplos rotulados é uma tarefa de alto custo. Para o caso em que poucos exemplos rotulados e um grande número de exemplos não rotulados estão disponíveis, é indicado o uso de algoritmos de aprendizado semissupervisionado, pois estes algoritmos podem obter melhores resultados no aprendizado do que algoritmos supervisionados que usam somente esse pequeno número de exemplos rotulados.

Quando os exemplos podem ser descritos de maneiras distintas, algoritmos de aprendizado semissupervisionado multidescrição podem ser aplicados — Seção 2.1. Nesse cenário, os classificadores obtidos em cada descrição usando-se os poucos exemplos rotulados podem ser ruins, porém, um exemplo não rotulado que for mal classificado por um classificador em uma descrição não será necessariamente mal classificado pelos classificadores nas outras descrições. Algoritmos multidescrição como o Co-Training (Blum e Mitchell, 1998) exploram essa propriedade para rotular automaticamente exemplos não rotulados e obter melhores classificadores com um conjunto de exemplos rotulados expandido.

O Co-Training é o principal algoritmo semissupervisionado multidescrição disponível atualmente. No entanto, ele apresenta um problema que pode surgir quando os classificadores em cada descrição discordam com alta confiança na classificação de um exemplo — Seção 3. Um exemplo no qual isso acontece é chamado de ponto de contenção, e sua rotulação automática pelo algoritmo Co-Training não é desejável, pois um dos classificadores o está classificando erroneamente e com alta confiança.

Em (Braga, 2010) foram propostas e avaliadas duas soluções para lidar com o problema dos pontos de contenção em Co-Training. A solução mais simples não permite que os pontos de contenção sejam rotulados, e é incorporada ao Co-Training pela substituição da função $melhoresExemplos$ Original — responsável pela rotulação de exemplos no algoritmo Co-Training — pela função NoContention. A outra solução proposta é o algoritmo CoAL, o qual incorpora aprendizado ativo multidescrição ao algoritmo Co-Training. No CoAL, exemplos não rotulados

identificados como pontos de contenção podem ser rotulados por um oráculo. Assim, esse exemplo rotulado pelo oráculo é inserido com o rótulo correto no conjunto de exemplos rotulados, "corrigindo" o classificador que o estava classificando erroneamente e com alta confiança.

O projeto do algoritmo CoAL foi realizado em (Braga, 2010). Uma implementação baseada nesse projeto foi realizada para avaliar esse algoritmo usando tanto a versão Original quanto a versão NoContention da função $melhoresExemplos$. Neste relatório, o objetivo é descrever os componentes principais dessa implementação e como utilizá-la — Seção 4.

2. Aprendizado Semissupervisionado Multidescrição

O aprendizado semissupervisionado permite o aprendizado a partir de um conjunto de exemplos rotulados L e de um conjunto de exemplos não rotulados U. Isso é particularmente útil quando existem muitos exemplos não rotulados disponíveis, mas a rotulação de um número expressivo deles não é viável. Nesta seção são apresentadas as vantagens de se utilizar múltiplas descrições dos exemplos no aprendizado semissupervisionado. Entre os algoritmos de aprendizado semissupervisionado multidescrição está o Co-Training, o qual pode ser considerado como a versão multidescrição do algoritmo Self-Training (Abney, 2007).

2.1. Multidescrição de Exemplos

Existem tarefas de classificação nas quais os exemplos podem estar descritos de duas ou mais maneiras distintas. Eis alguns exemplos:

- Na filtragem de *spam* (Kiritchenko e Matwin, 2001; Koprinska et al., 2007), é possível construir um classificador para identificar *spam* usando o texto contido no campo assunto das mensagens ou usando o texto contido no corpo das mensagens;
- Para classificar imagens em uma página *web*, é possível construir um classificador usando as características visuais (baseadas nos *pixels* das imagens) ou usando o texto que acompanha as imagens (Gupta et al., 2008);
- Para identificar momentos importantes de uma partida de futebol, pode-se construir um classificador usando as características visuais obtidas da gravação da partida ou usando as características sonoras obtidas da narração da partida (Gupta et al., 2008);
- Para identificar artigos científicos de uma determinada área, é possível construir um classificador usando o texto dos artigos ou

usando uma rede de citações extraída dos artigos (Laguna e Lopes, 2009).

Considerando o formato atributo-valor como linguagem de descrição de exemplos, a existência de múltiplas descrições dos exemplos implica em haver dois ou mais conjuntos distintos de atributos para representar os mesmos exemplos. Implica, também, em cada um desses conjuntos de atributos ser suficiente para aprender um classificador caso fosse disponibilizada uma quantidade expressiva de exemplos rotulados.

No aprendizado semissupervisionado, no entanto, o número de exemplos rotulados é escasso. Assim, os classificadores obtidos usando somente os poucos exemplos rotulados disponíveis podem não ser eficazes. Entretanto, usando diferentes descrições dos exemplos, um exemplo mal classificado por um classificador não vai ser necessariamente mal classificado pelos outros classificadores. Essa é a grande vantagem de se utilizar múltiplas descrições dos exemplos no aprendizado semissupervisionado.

Considerando que duas descrições dos exemplos estão disponíveis e que essas descrições não são muito correlacionadas, um exemplo não rotulado que é classificado com alta confiança pelo classificador na primeira descrição torna-se um exemplo rotulado bastante informativo para o aprendizado do classificador na segunda descrição e vice-versa. Assim, é possível criar métodos de aprendizado semissupervisionado nos quais os classificadores construídos nas diferentes descrições fornecem exemplos rotulados com alta confiança uns aos outros, contornando o problema da escassez de exemplos rotulados. Essa ideia é explorada pelo algoritmo Co-Training, apresentado a seguir.

2.2. O Algoritmo Co-Training

O algoritmo Co-Training (Blum e Mitchell, 1998) consiste na obtenção de múltiplos classificadores a partir de diferentes descrições dos exemplos de treinamento. Esses classificadores rotulam exemplos com alta confiança, e esses novos exemplos rotulados são então adicionados ao conjunto de exemplos rotulados em todas as descrições. Assim, sempre que um dos classificadores rotular um exemplo, ele pode colaborar na melhoria dos outros classificadores por estar incrementando o número de exemplos do conjunto de treinamento.

Primeiramente, considere que o conjunto de M atributos $A = \{X_1, X_2, ..., X_M\}$, o qual é usado para descrever um conjunto de treinamento E, pode ser dividido em dois subconjuntos A_{D_1} e A_{D_2}, tal que $A = A_{D_1} \cup A_{D_2}$ e $A_{D_1} \cap A_{D_2} = \emptyset$. Como ilustrado na Figura 1, essa divisão provoca a criação de duas descrições E_{D_1} e E_{D_2} dos exemplos de treinamento. Nessa figura, por simplicidade, é considerado que $A_{D_1} = \{X_1, X_2, ..., X_j\}$ e $A_{D_2} = \{X_{j+1}, X_{j+2}, ..., X_M\}$. Exemplos com valor de Y igual a "?" representam exemplos não rotulados. Os vetores $(x_{i1}, x_{i2}, ..., x_{ij})$ e $(x_{i(j+1)}, x_{i(j+2)}, ..., x_{iM})$, que representam os valores dos atributos do exemplo $\mathbf{x}_i$ em cada descrição, serão indicados, respectivamente, pelas notações $\mathbf{x}_i^{D_1}$ e $\mathbf{x}_i^{D_2}$.

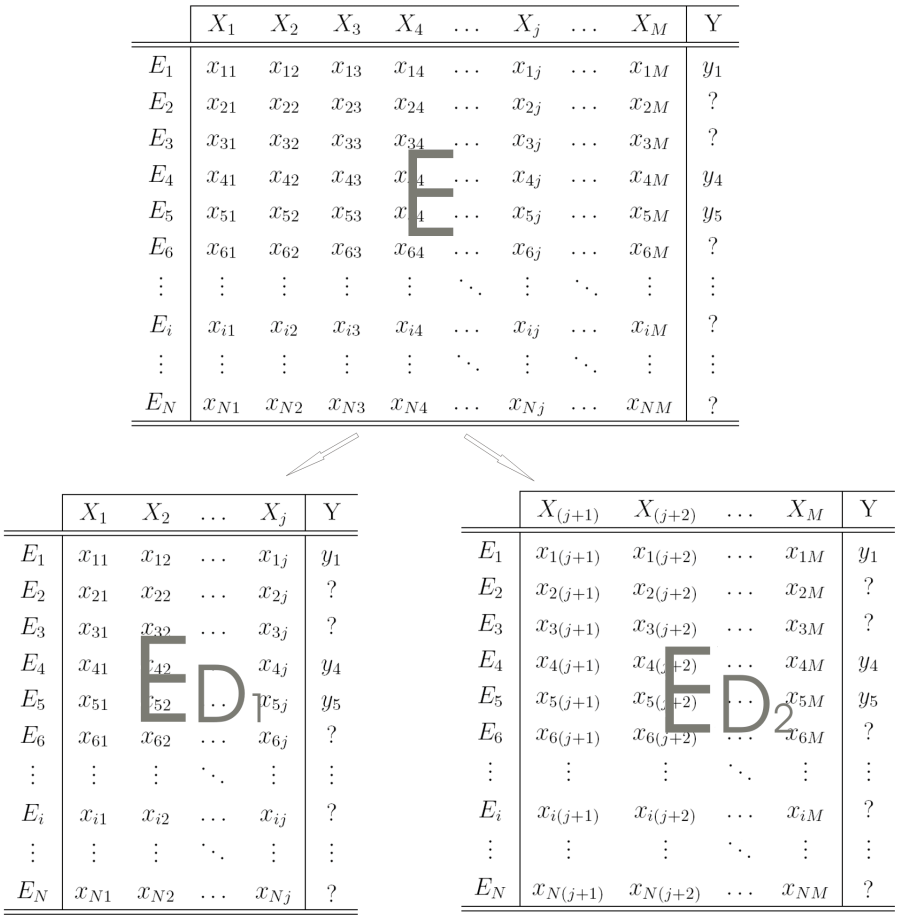

Figura 1. Duas descrições do conjunto de exemplos

Além da separação em duas descrições, os conjuntos E_{D_1} e E_{D_2} devem ser divididos, cada um, em dois subconjuntos L e U. Os exemplos em E_{D_1} que possuem o atributo classe conhecido formam o conjunto L_{D_1}, enquanto que os exemplos restantes, não rotulados, formam $U_{D_1} = E_{D_1} -$

L_{D_1}. Analogamente, E_{D_2} é dividido em L_{D_2} e U_{D_2}. Os dados de entrada do algoritmo CO-TRAINING consistem desses quatro conjuntos de exemplos L_{D_1}, L_{D_2}, U_{D_1} e U_{D_2}. Na Figura 2, encontram-se ilustrados esses quatro conjuntos.

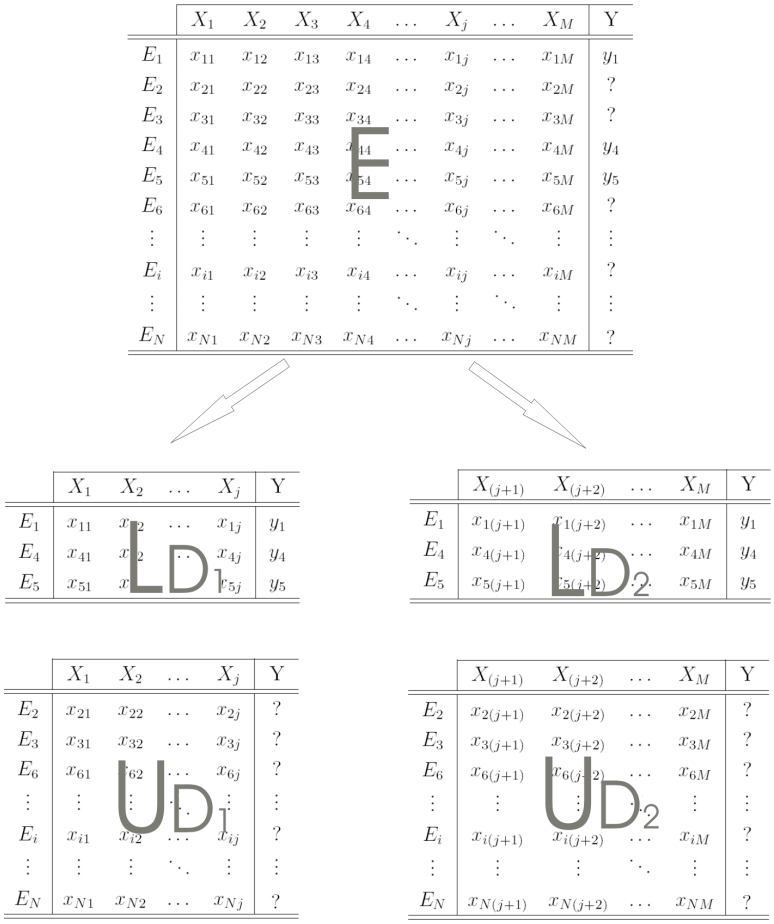

Figura 2. Conjunto de exemplos L_{D_1}, L_{D_2} U_{D_1} e U_{D_2}

No primeiro passo do CO-TRAINING — Figura 3 e Algoritmo 1 — são criados aleatoriamente dois subconjuntos de um número limitado de exemplos U'_{D_1} e U'_{D_2}, tal que $U'_{D_1} \subset U_{D_1}$ e $U'_{D_2} \subset U_{D_2}$. A construção desses dois conjuntos é feita de tal forma que U'_{D_1} e U'_{D_2} contêm os mesmos exemplos em descrições diferentes[1]. Formalmente, para cada $\mathbf{x}_i$ selecionado aleatoriamente, $(\mathbf{x}_i^{D_1}, ?) \in U'_{D_1} \Leftrightarrow (\mathbf{x}_i^{D_2}, ?) \in U'_{D_2}$. Os exemplos que compõem U'_{D_1} e U'_{D_2} são então removidos de U_{D_1} e U_{D_2} a fim de verificar as condições $U'_{D_1} \cap U_{D_1} = \emptyset$ e $U'_{D_2} \cap U_{D_2} = \emptyset$.

[1]O uso dos dois conjuntos auxiliares U'_{D_1} e U'_{D_2}, que constituem uma amostra dos conjuntos U_{D_1} e U_{D_2}, não é obrigatório. Contudo o número de exemplos não rotulados disponíveis em U_{D_1} e U_{D_2} pode ser muito grande. Assim, esses dois conjuntos auxiliares permitem diminuir o tempo de rotulação de exemplos em cada iteração do algoritmo.

Algoritmo 1: CO-TRAINING

Entrada: $L_{D_1}, L_{D_2}, U_{D_1}, U_{D_2}, k$
Retirar aleatoriamente alguns exemplos de U_{D_1} e U_{D_2} e adicioná-los em U'_{D_1} e U'_{D_2};
para $it = 1$ *até* k **faça**
  obter o classificador h_{D_1} a partir dos exemplos de treinamento em L_{D_1};
  obter o classificador h_{D_2} a partir dos exemplos de treinamento em 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};
  $(R_{D_1}, R_{D_2}) = melhoresExemplos(R'_{D_1}, R'_{D_2})$;
  $L_{D_1} = L_{D_1} \cup R_{D_1}$;
  $L_{D_2} = L_{D_2} \cup R_{D_2}$;
  para cada $(\boldsymbol{x}_i^{D_1}, y_i) \in R_{D_1}$ **faça**
    $U'_{D_1} = U'_{D_1} - \{(\mathbf{x}_i^{D_1}, ?)\}$;
    $U'_{D_2} = U'_{D_2} - \{(\mathbf{x}_i^{D_2}, ?)\}$;
  fim
  se $U_{D_1} = \emptyset$ **então Retorne** $h_{D_1}, h_{D_2}, L_{D_1}, L_{D_2}$ **senão**
    Recompor U'_{D_1} e U'_{D_2} a partir de U_{D_1} e U_{D_2};
  fim
fim
Retorne $h_{D_1}, h_{D_2}, L_{D_1}, L_{D_2}$;

No segundo passo, são obtidos dois classificadores h_{D_1} e h_{D_2} a partir dos conjuntos de exemplos rotulados L_{D_1} e L_{D_2}. Esses classificadores podem ser obtidos usando qualquer algoritmo de aprendizado que possa induzir um classificador e dar um valor de confiança (escore) para as suas classificações (na versão original, foi utilizado o *Naive* Bayes como algoritmo-base). No terceiro passo, h_{D_1} rotula todos os exemplos não rotulados de U'_{D_1}, e h_{D_2} faz o mesmo em U'_{D_2}. No fim desse processo, R'_{D_1} e R'_{D_2} contém, respectivamente, o conjunto de exemplos rotulados pelos classificadores h_{D_1} e h_{D_2}. O quarto passo consiste da seleção dos "melhores" exemplos rotulados $(\mathbf{x}_i, y_i)$ a partir de R'_{D_1} e R'_{D_2}. As partições $(\mathbf{x}_i^{D_1}, y_i)$ e $(\mathbf{x}_i^{D_2}, y_i)$ dos exemplos selecionados são inseridas em R_{D_1} e R_{D_2} e, em seguida, adicionadas a L_{D_1} e L_{D_2}. Desse modo, o número de exemplos rotulados pode ser incrementado a cada iteração. Finalmente, o processo para quando não existem mais exemplos a serem rotulados, ou o número máximo de iterações k seja atingido.

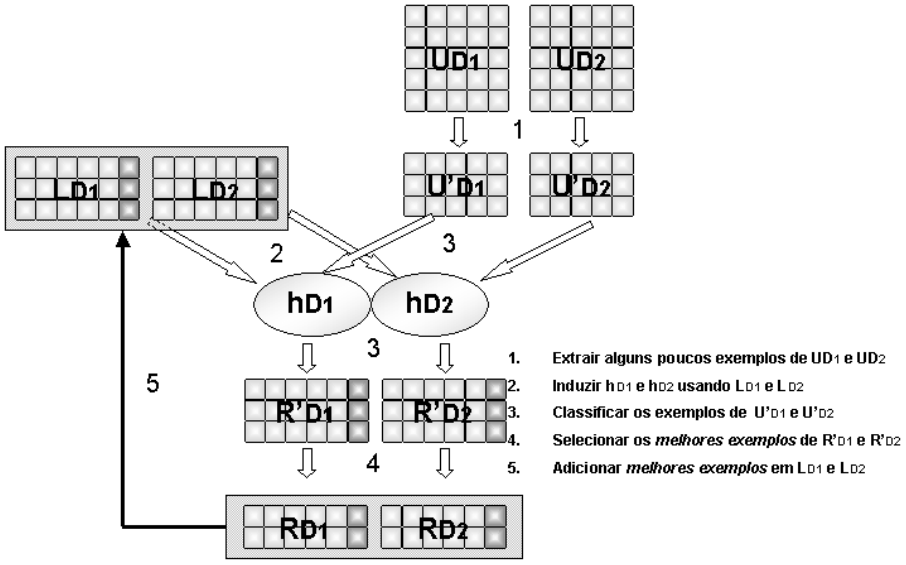

Figura 3. Ilustração do algoritmo CO-TRAINING (Matsubara, 2004)

Deve ser observado que o critério de seleção dos melhores exemplos, assim como o número de exemplos de cada classe a serem rotulados em cada iteração do algoritmo, são parâmetros importantes da função $melhoresExemplos$. Na proposta original de CO-TRAINING (Blum e Mitchell, 1998), o conjunto de exemplos utilizado possui somente duas classes, e os autores sugerem selecionar os p "melhores" exemplos positivos e os n "melhores" exemplos negativos em cada descrição para incrementar o conjunto de exemplos rotulados, sendo que "melhores" exemplos se referem aos exemplos rotulados com a mais alta confiança — Algoritmo 2. Nesse mesmo trabalho, os valores de p e n foram determinados considerando a distribuição das classes no conjunto de exemplos utilizado, o qual, inicialmente, estava todo rotulado. Entretanto, é importante ressaltar que em uma execução real de CO-TRAINING, somente é conhecido o rótulo de poucos exemplos, e inferir a distribuição das classes a partir de um conjunto de exemplos muito pequeno pode não ser válido. Em (Matsubara et al., 2006), é verificado experimentalmente que uma escolha de p e n em uma proporção muito diferente da distribuição natural das classes tende a degradar o desempenho de CO-TRAINING.

Como pode ser observado, a complexidade do Algoritmo 1 e de outros algoritmos baseados na ideia do CO-TRAINING depende fortemente

Algoritmo 2: Função $melhoresExemplos$ de Blum e Mitchell (1998) — ORIGINAL

Entrada: R'_{D_1}, R'_{D_2}
R_{D_1} = p melhores exemplos positivos e n melhores exemplos negativos de R'_{D_1};
R_{D_2} = p melhores exemplos positivos e n melhores exemplos negativos de R'_{D_2};
Para os exemplos que estiverem tanto em R_{D_1} quanto em R_{D_2}, mantenha somente o exemplo rotulado com maior confiança;
para cada $(\boldsymbol{x}_i^{D_1}, y_i) \in R_{D_1}$ **faça**
  $R_{D_2} = R_{D_2} \cup \{(\mathbf{x}_i^{D_2}, y_i)\}$;
fim
para cada $(\boldsymbol{x}_i^{D_2}, y_i) \in R_{D_2}$ **faça**
  $R_{D_1} = R_{D_1} \cup \{(\mathbf{x}_i^{D_1}, y_i)\}$;
fim
Retorne R_{D_1}, R_{D_2};

do algoritmo de aprendizado utilizado para obter, em cada iteração, os classificadores h_{D_1} e h_{D_2}. Dependendo do algoritmo-base utilizado, a complexidade da geração desses classificadores pode tipicamente atingir complexidade $O(|L|^2)$ em uma iteração, com $|L|$ igual ao número de exemplos rotulados disponíveis na iteração. Quanto à rotulação dos exemplos em U'_{D_1} e U'_{D_2}, ela pode ser realizada em cada iteração tipicamente em tempo linear no número de exemplos, o qual esta limitado pelo número de exemplos nesses subconjuntos. Finalmente, a complexidade da função $melhoresExemplos$ está relacionada com o algoritmo de ordenação utilizado para ranquear os exemplos já classificados de U'_{D_1} e U'_{D_2}. Assim, se a complexidade do algoritmo atingir $O(|L|^2)$, então o fator dominante em cada iteração será a obtenção dos classificadores. Ao longo das iterações, o valor de $|L|$ vai aumentando, enquanto que os conjuntos U'_{D_1} e U'_{D_2} só podem manter ou diminuir de tamanho. Pelo fato de um número grande de iterações poder ser realizado no CO-TRAINING, é mais viável utilizar um algoritmo de aprendizado incremental, o qual diminui consideravelmente a complexidade da obtenção dos classificadores em cada descrição.

Para finalizar esta seção, é interessante notar que CO-TRAINING é o algoritmo semissupervisionado multidescrição mais utilizado e mais citado na literatura, tendo sido amplamente estudado em diversos contextos, como a classificação de páginas da *web* (Blum e Mitchell, 1998), reconhecimento de entidades nominais (Collins e Singer, 1999), indução de *wrappers* (Muslea et al., 2002b), classificação de *emails* (Kiritchenko

e Matwin, 2001; Koprinska et al., 2007), classificação de diálogos (Maeireizo et al., 2004), classificação de textos (Matsubara et al., 2005) e classificação de imagens e vídeos (Gupta et al., 2008).

3. O Problema dos Pontos de Contenção em Co-Training

O algoritmo Co-Training proposto por Blum e Mitchell (1998) utiliza conjuntos de exemplos com duas classes $\{\oplus, \ominus\}$ e o algoritmo *Naive* Bayes (NB) como algoritmo-base escore para gerar h_1 e h_2. Usando a versão Original da função $melhoresExemplos$, o número máximo de exemplos que podem ser rotulados em cada iteração é $2p + 2n$.

A função Original — Algoritmo 2 — apresenta o seguinte problema: se um exemplo tiver sido rotulado positivamente com alta confiança em uma descrição e negativamente com alta confiança em uma outra descrição, então o rótulo desse exemplo é definido arbitrariamente, o que facilita a introdução de um erro de rotulação, já que somente uma das classificações é correta. Em outras palavras, essa função não verifica por rotulações conflitantes de $h_1(\mathbf{x}_i^{D_1})$ e $h_2(\mathbf{x}_i^{D_2})$ do mesmo exemplo $\mathbf{x}_i$.

Para exemplificar a rotulação realizada por Original, considere que existam 10 exemplos contidos nos conjuntos R'_{D_1} e R'_{D_2} — Tabela 1 — para os quais NB fornece uma aproximação da probabilidade do exemplo ser classificado como $\oplus$ ou $\ominus$ para $h_1(\mathbf{x}_i^{D_1})$ e $h_2(\mathbf{x}_i^{D_2})$, $i = 1..10$, dadas, respectivamente, pelos pares de valores (p_1, n_1) e (p_2, n_2). Para esses pares de valores, é indicada a ordem de prioridade considerando, respectivamente, p_1 e p_2 como valores de confiança e o valor $0,5$ como limiar de decisão. Considere que, em cada iteração, são rotulados por cada classificador os $p = 1$ exemplos $\oplus$ e $n = 3$ exemplos $\ominus$ rotulados com maior confiança. Para os exemplos na Tabela 1, em cada descrição, o exemplo com prioridade 1 é classificado como $\oplus$ e os exemplos com prioridade 10, 9 e 8 como $\ominus$.

Em outras palavras, $h_1(\mathbf{x}_i^{D_1})$ classifica E_5 $\oplus$ e E_3, E_7, E_9 $\ominus$, enquanto $h_2(\mathbf{x}_i^{D_2})$ classifica E_9 $\oplus$ e E_5, E_{10}, E_7 $\ominus$. Assim, pode ser observado que há conflito na rotulação dos exemplos E_5 e E_9. Caso os exemplos rotulados por $h_1(\mathbf{x}_i^{D_1})$ sejam incorporados primeiramente ao conjunto de exemplos rotulados, os exemplos rotulados nessa iteração de Co-Training com a função Original seriam os exemplos E_5 $\oplus$ e E_3, E_7, E_9 $\ominus$ de $h_1(\mathbf{x}_i^{D_1})$ e o exemplo E_{10} de $h_2(\mathbf{x}_i^{D_2})$. Com esse critério, dos 5 exemplos rotulados nessa iteração, dois deles são rotulados errados — coluna Original da

Tabela 1. Rotulação de exemplos pela função ORIGINAL

	Pri. $h_1(\mathbf{x}_i^{D_1})$	$h_1(\mathbf{x}_i^{D_1})$	Pri. $h_2(\mathbf{x}_i^{D_2})$	$h_2(\mathbf{x}_i^{D_2})$	ORIGINAL	Verdadeiro
E_1	2	(0.95, 0,05) $\oplus$	4	(0.70, 0.30) $\oplus$		$\oplus$
E_2	3	(0.70, 0.30) $\oplus$	3	(0.80, 0.20) $\oplus$		$\oplus$
E_3	10	(0.25, 0.75) $\ominus$	2	(0.90, 0.10) $\oplus$	$\ominus$	$\ominus$
E_4	7	(0.40. 0.60) $\ominus$	5	(0.51, 0.49) $\oplus$		$\ominus$
E_5	1	(1.00, 0.00) $\oplus$	10	(0.00, 1.00) $\ominus$	$\oplus$	$\ominus$
E_6	4	(0.65, 0.35) $\oplus$	7	(0.15, 0.85) $\ominus$		$\ominus$
E_7	9	(0.30, 0.70) $\ominus$	8	(0.10, 0.90) $\ominus$	$\ominus$	$\ominus$
E_8	6	(0.45, 0.55) $\ominus$	6	(0.45, 0.55) $\ominus$		$\ominus$
E_9	8	(0.35, 0.65) $\ominus$	1	(0.95, 0.05) $\oplus$	$\ominus$	$\oplus$
E_{10}	5	(0.55, 0.45) $\oplus$	9	(0.05, 0.95) $\ominus$	$\ominus$	$\ominus$

Tabela 1.

Exemplos que são classificados diferentemente por $h_1(\mathbf{x}_i^{D_1})$ e por $h_2(\mathbf{x}_i^{D_2})$ com alta confiança, tais como os exemplos E_5, E_3, E_6 e E_9 na Tabela 1, são chamados de *pontos de contenção*. Se um ponto de contenção tiver sido selecionado pela função ORIGINAL, então ele passa a integrar o conjunto de exemplos rotulados, o que não é desejável, pois não é possível decidir o rótulo de um ponto de contenção sem informação adicional. Deve ser observado que essa situação pode acontecer nas iterações iniciais de CO-TRAINING, pois o número de exemplos rotulados disponíveis para gerar h_1 e h_2 é pequeno. A seguir são apresentadas duas possíveis soluções para tratar pontos de contenção.

3.1. Solução I: Não Rotular Pontos de Contenção

A maneira mais direta de tratar pontos de contenção em CO-TRAINING é identificando-os e não os rotulando, o que pode ser realizado pela função $melhoresExemplos$. Em (Braga, 2010) é proposta uma função $melhoresExemplos$ para tratar pontos de contenção denominada NOCONTENTION[2] — Algoritmo 3.

A função NOCONTENTION apresentada no Algoritmo 3 foi projetada para trabalhar com duas descrições dos exemplos e com duas classes. Para obter uma solução nessa mesma linha para os casos em que existam mais de duas descrições e mais de duas classes, é necessário generalizar a definição de ponto de contenção.

A segunda solução proposta neste trabalho considera o uso de aprendizado ativo. Para entendê-la, é necessário compreender o algoritmo CO-TESTING, explicado a seguir.

[2]A função NOCONTENTION foi apresentada em (Matsubara, 2004; Braga et al., 2009) com uma ligeira diferença em relação à que é considerada neste trabalho.

Algoritmo 3: Função $melhoresExemplos$ NoContention

Entrada: R'_{D_1}, R'_{D_2}
Retirar de R'_{D_1} e R'_{D_2} todos os exemplos para os quais há uma discordância na classificação;
R_{D_1} = p melhores exemplos positivos e n melhores exemplos negativos de R'_{D_1};
R_{D_2} = p melhores exemplos positivos e n melhores exemplos negativos de R'_{D_2};
para cada $(\boldsymbol{x}_i^{D_1}, y_i) \in R_{D_1}$ **faça**
  $R_{D_2} = R_{D_2} \cup \{(\mathbf{x}_i^{D_2}, y_i)\}$;
fim
para cada $(\boldsymbol{x}_i^{D_2}, y_i) \in R_{D_2}$ **faça**
  $R_{D_1} = R_{D_1} \cup \{(\mathbf{x}_i^{D_1}, y_i)\}$;
fim
Retorne R_{D_1}, R_{D_2};

3.2. O Algoritmo Co-Testing

Co-Testing (Muslea et al., 2000, 2006) é um algoritmo multidescrição de aprendizado ativo (*active learning*). A ideia usada em Co-Testing consiste em selecionar pontos de contenção e fornecê-los a um *oráculo*[3] para que eles sejam rotulados. Então, os exemplos que o oráculo classifica são adicionados ao conjunto de exemplos rotulados e o processo é repetido. Com isso, o aprendizado passa a ser ativo, isto é, o próprio algoritmo de aprendizado requisita o rótulo de alguns exemplos.

Co-Testing é descrito pelo Algoritmo 4, o qual utiliza as notações apresentadas na seção anterior. Primeiramente, os classificadores h_{D_1} e h_{D_2} são induzidos utilizando os conjuntos L_{D_1} e L_{D_2} de exemplos rotulados. Em seguida, é construído o conjunto PC que contém os exemplos que foram rotulados com classes diferentes em cada descrição. Se $PC = \emptyset$ o processo termina, caso contrário, no próximo passo, a função $seleciona$ é responsável pela escolha do exemplo a ser submetido ao oráculo para ele fornecer o rótulo correspondente. Os exemplos $\mathbf{x}_i^{D_1}$ e $\mathbf{x}_i^{D_2}$ rotulados pelo oráculo são retirados de U_{D_1} e U_{D_2} e inseridos, juntamente com o rótulo dado, nos respectivos conjuntos L_{D_1} e L_{D_2} de exemplos rotulados. O processo é repetido até k vezes, caso não se verifique $PC \neq \emptyset$.

O Co-Testing foi proposto para ser usado com qualquer algoritmo-base de aprendizado, inclusive por algoritmos que não dão um valor de

[3]Um oráculo pode ser um especialista do domínio, um usuário do sistema ou um outro dispositivo que informe o rótulo correto de um exemplo.

Algoritmo 4: CO-TESTING

Entrada: $L_{D_1}, L_{D_2}, U_{D_1}, U_{D_2}, k$
para $it = 1$ *até* k **faça**
  obter o classificador h_{D_1} a partir dos exemplos de treinamento em L_{D_1};
  obter o classificador h_{D_2} a partir dos exemplos de treinamento em 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};
  $PC = \{(\mathbf{x}_i^{D_1}, \mathbf{x}_i^{D_2}) | h_{D_1}(\mathbf{x}_i^{D_1}) \neq h_{D_2}(\mathbf{x}_i^{D_2}) \wedge (\mathbf{x}_i^{D_1}, h_{D_1}(\mathbf{x}_i^{D_1})) \in R_{D_1} \wedge (\mathbf{x}_i^{D_2}, h_{D_2}(\mathbf{x}_i^{D_2})) \in R_{D_2}\}$;
  se $PC = \emptyset$ **então Retorne** $h_{D_1}, h_{D_2}, L_{D_1}, L_{D_2}$;
  $(\mathbf{x}_i^{D_1}, \mathbf{x}_i^{D_2}) = selecionar(PC)$;
  $y_i =$ rótulo, após interrogar o oráculo, atribuído ao par $(\mathbf{x}_i^{D_1}, \mathbf{x}_i^{D_2})$;
  $L_{D_1} = L_{D_1} \cup \{(\mathbf{x}_i^{D_1}, y_i)\}$;
  $L_{D_2} = L_{D_2} \cup \{(\mathbf{x}_i^{D_2}, y_i)\}$;
  $U_{D_1} = U_{D_1} - \{(\mathbf{x}_i^{D_1}, ?)\}$;
  $U_{D_2} = U_{D_2} - \{(\mathbf{x}_i^{D_2}, ?)\}$;
fim
Retorne $h_{D_1}, h_{D_2}, L_{D_1}, L_{D_2}$;

confiança para a classificação. Nesse caso, a função *selecionar* escolhe um exemplo aleatoriamente do conjunto PC. Caso seja utilizado um algoritmo-base que dá um valor de confiança (escore) para cada possível valor da classe, pode-se, então, escolher o exemplo que foi classificado com máximo escore em classes distintas. De qualquer maneira, o objetivo em cada iteração é selecionar o exemplo mais informativo para o oráculo rotular.

Observe que os exemplos rotulados com a mesma classe pelos classificadores não são adicionados em L_{D_1} e L_{D_2} durante as iterações do algoritmo. Somente os exemplos que foram rotulados pelo oráculo são adicionados a L_{D_1} e L_{D_2}.

3.3. Solução II: O Algoritmo CoAL

Nesta seção é apresentada a solução apresentada em (Braga, 2010) para tratar pontos de contenção no algoritmo CO-TRAINING utilizando aprendizado ativo multidescrição nos moldes do algoritmo CO-TESTING. O algoritmo resultante é denominado CoAL (CO-TRAINING with Active Learning) — Algoritmo 5. É importante notar que essa solução pode ser com-

binada à solução descrita na seção 3.1, isto é, o algoritmo CoAL pode usar a versão NoContention da função $melhoresExemplos$.

O algoritmo CoAL adiciona ao Co-Training somente o bloco **enquanto**, o qual é executado antes dos comandos de cada iteração do Co-Training. Esse bloco é responsável por selecionar pontos de contenção para o oráculo rotular e adicioná-los ao conjunto de exemplos rotulados. Em cada iteração de CoAL, os comandos dentro do bloco **enquanto** são executados até que não exista mais nenhum ponto de contenção ou o número máximo de consultas ao oráculo tenha sido atingido. Caso essa última condição se verifique, novas consultas não são realizadas nas iterações seguintes do algoritmo CoAL.

O algoritmo CoAL pode ser vantajoso no aprendizado semissupervisionado quando um usuário desempenha o papel do oráculo durante o processo de aprendizado. Quando um ponto de contenção aparece em uma execução de CoAL usando duas descrições e conjuntos de exemplos com duas classes, um dos classificadores está classificando erroneamente e com alta confiança o exemplo não rotulado. Assim, a rotulação do usuário contribui para que esse classificador seja corrigido em um exemplo na qual sua predição foi a pior possível. Se os pontos de contenção aparecem logo nas primeiras iterações, o algoritmo CoAL pode ser ainda mais benéfico, pois evita que os erros de rotulação que seriam causados por esses pontos se propaguem ao longo de suas iterações.

Outro ponto importante a ser ressaltado é que a combinação de aprendizado semissupervisionado e aprendizado ativo no contexto de multidescrição de exemplos já havia sido proposta anteriormente (Muslea et al., 2002a). No entanto, essa proposta embute um algoritmo de aprendizado semissupervisionado no algoritmo de aprendizado ativo, enquanto que, neste trabalho, é proposto o inverso. Em outras palavras, no trabalho de Muslea et al. (2002a), o objetivo é realizar aprendizado ativo, e, neste trabalho, o objetivo é realizar aprendizado semissupervisionado. A diferença é que um algoritmo de aprendizado ativo incrementa o conjunto de exemplos rotulados unicamente com os exemplos rotulados pelo oráculo. Já neste trabalho, são inseridos no conjunto de rotulados tanto os exemplos rotulados pelo usuário quanto os exemplos rotulados por Co-Training.

Algoritmo 5: CoAL

Entrada: $L_{D_1}, L_{D_2}, U_{D_1}, U_{D_2}, k, maxConsultas$
Retirar aleatoriamente alguns exemplos de U_{D_1} e U_{D_2} e adicioná-los em U'_{D_1} e U'_{D_2};
$consulta = 0$;
para $it = 1$ *até* k **faça**
    enquanto $consulta < maxConsultas$ **faça**
        obter o classificador h_{D_1} a partir dos exemplos em L_{D_1};
        obter o classificador h_{D_2} a partir dos exemplos em 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};
        $PC = \{(\mathbf{x}_i^{D_1}, \mathbf{x}_i^{D_2}) | h_{D_1}(\mathbf{x}_i^{D_1}) \neq h_{D_2}(\mathbf{x}_i^{D_2}) \wedge (\mathbf{x}_i^{D_1}, h_{D_1}(\mathbf{x}_i^{D_1})) \in R_{D_1} \wedge (\mathbf{x}_i^{D_2}, h_{D_2}(\mathbf{x}_i^{D_2})) \in R_{D_2}\}$;
        se $PC = \emptyset$ **então** Saia do laço **enquanto**;
        $(\mathbf{x}_i^{D_1}, \mathbf{x}_i^{D_2}) = selecionar(PC)$;
        y_i = rótulo, após interrogar o oráculo, atribuído ao par $(\mathbf{x}_i^{D_1}, \mathbf{x}_i^{D_2})$;
        $L_{D_1} = L_{D_1} \cup \{(\mathbf{x}_i^{D_1}, y_i)\}$;
        $L_{D_2} = L_{D_2} \cup \{(\mathbf{x}_i^{D_2}, y_i)\}$;
        $U_{D_1} = U_{D_1} - \{(\mathbf{x}_i^{D_1}, ?)\}$;
        $U_{D_2} = U_{D_2} - \{(\mathbf{x}_i^{D_2}, ?)\}$;
        $consulta++$;
    fim
    obter o classificador h_{D_1} a partir dos exemplos de treinamento em L_{D_1};
    obter o classificador h_{D_2} a partir dos exemplos de treinamento em 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};
    $(R_{D_1}, R_{D_2}) = melhoresExemplos(R'_{D_1}, R'_{D_2})$;
    $L_{D_1} = L_{D_1} \cup R_{D_1}$;
    $L_{D_2} = L_{D_2} \cup R_{D_2}$;
    para cada $(\boldsymbol{x}_i^{D_1}, y_i) \in R_{D_1}$ **faça**
        $U'_{D_1} = U'_{D_1} - \{(\mathbf{x}_i^{D_1}, ?)\}$;
        $U'_{D_2} = U'_{D_2} - \{(\mathbf{x}_i^{D_2}, ?)\}$;
    fim
    se $U_{D_1} = \emptyset$ **então Retorne** $h_{D_1}, h_{D_2}, L_{D_1}, L_{D_2}$ **senão**
        Recompor U'_{D_1} e U'_{D_2} a partir de U_{D_1} e U_{D_2};
    fim
fim
Retorne $h_{D_1}, h_{D_2}, L_{D_1}, L_{D_2}$;

4. Implementação do Algoritmo CoAL

Uma implementação do algoritmo CoAL foi realizada para avaliar esse algoritmo em (Braga, 2010). Como base para essa implementação, foi feito amplo uso do *Weka* (Witten e Frank, 2005), o qual é uma coleção de algoritmos de aprendizado de máquina direcionados para a realização de tarefas de mineração de dados. Além de algoritmos de aprendizado, o Weka conta com uma variedade de algoritmos que facilitam a manipulação das principais estruturas de dados utilizadas por algoritmos de aprendizado. Tais facilidades fazem do Weka um ambiente apropriado para o desenvolvimento de novos esquemas de aprendizado de máquina.

O algoritmo CoAL, assim como o algoritmo Co-Training, requer a utilização de um algoritmo-base supervisionado para a obtenção de classificadores em cada uma das suas iterações. A definição de qual algoritmo-base utilizar depende exclusivamente do domínio em que se queira aplicar esses algoritmos. Assim, uma implementação que não facilite o uso de diferentes algoritmos-base não é desejável. Isso faz com que o Weka seja bastante apropriado para a implementação do algoritmo CoAL, pois o Weka permite mudar o algoritmo-base sem ter que realizar modificações no código da implementação do CoAL.

Neste relatório é dada uma visão geral sobre a implementação do algoritmo CoAL utilizando o Weka, além da descrição de uso dessa implementação e de como estendê-la para incorporar melhorias a esses algoritmos. Para entender o funcionamento do Weka, o leitor pode se referir ao sítio `http://www.cs.waikato.ac.nz/ml/weka/`, o qual contém a documentação oficial da ferramenta e apontadores para diversos tutoriais.

4.1. Classes

O Weka foi desenvolvido na plataforma Java, a qual é orientada a objetos. Assim, existe uma coleção de classes que implementam os algoritmos de aprendizado, as estruturas de dados utilizadas por esses algoritmos e outras funcionalidades interessantes em um processo de mineração de dados. A documentação relativa a essa coleção de classes pode ser encontrada no endereço `http://weka.sourceforge.net/doc.stable/`.

Entre as classes do Weka utilizadas na implementação do CoAL, duas são bastante importantes:

- `weka.classifiers.Classifier`: classe abstrata representando um classificador no Weka. As classes representando os

algoritmos de aprendizado para classificação no Weka estendem essa classe abstrata. Por exemplo, a classe `weka.classifiers.bayes.NaiveBayes` implementa o método abstrato `buildClassifier` da classe `Classifier` usando o algoritmo de aprendizado supervisionado *Naive* Bayes.

- `weka.core.Instances`: classe para manipulação de um conjunto de exemplos. O método `buildClassifier`, implementado pelas classes que estendem a classe `Classifier`, recebe um objeto da classe `Instances` representando um conjunto de treinamento.

A implementação do algoritmo CoAL consiste de três classes, cujos códigos podem ser encontrados no endereço http://www.labic.icmc.usp.br/cotraining/:

- `AbstractCotraining`: classe abstrata que implementa um esquema geral para algoritmos que seguem o mesmo molde do CoAL e do Co-Training em geral. Para definir um esquema de aprendizado semissupervisionado multidescrição completo, é possível estender essa classe e implementar métodos relacionados à função *melhoresExemplos* do Co-Training e à função *selecionar* do CoAL.
- `CpCotraining`: classe que estende `AbstractCotraining` e que implementa o algoritmo CoAL, e também o algoritmo Co-Training, usando a função *melhoresExemplos* NoContention — seção 3.1.
- `OriginalCotraining`: classe que estende `AbstractCotraining` e que implementa o algoritmo CoAL, e também o algoritmo Co-Training, usando a função *melhoresExemplos* Original — seção 2.2.

4.2. Exemplo de Uso

Um exemplo de uso das classes que implementa o algoritmo CoAL é apresentado nesta seção. Na Listagem 1 é apresentado um trecho de código escrito em linguagem Java para executar o CoAL usando a função *melhoresExemplos* NoContention. Como será visto adiante, com pequenas mudanças nesse trecho de código, é possível executar o CoAL usando a versão Original da função *melhoresExemplos* e o Co-Training usando ambas as versões da função *melhoresExemplos*.

```
1  import br.usp.icmc.labic.cotraining.CpCotraining;
2  import weka.classifiers.Classifier;
3  import weka.core.Instances;
4  import weka.core.converters.ConverterUtils.DataSource;
5  import weka.core.Utils;
6
7  ...
8
9  Instances[] labeled = new Instances[2];
10 Instances[] unlabeled = new Instances[2];
11
12 labeled[0] = new DataSource("l1.arff").getDataSet();
13 labeled[1] = new DataSource("l2.arff").getDataSet();
14 unlabeled[0] = new DataSource("u1.arff").getDataSet();
15 unlabeled[1] = new DataSource("u2.arff").getDataSet();
16
17 labeled[0].setClassIndex(labeled[0].numAttributes() - 1);
18 labeled[1].setClassIndex(labeled[1].numAttributes() - 1);
19 unlabeled[0].setClassIndex(unlabeled[0].numAttributes() - 1);
20 unlabeled[1].setClassIndex(unlabeled[1].numAttributes() - 1);
21
22 String[] options = Utils.splitOptions("-C weka.classifiers.
      functions.SMO;weka.classifiers.bayes.NaiveBayes -T 0.8 -L 5,5
      -U -A 10 -S 187");
23 String[][] baseClassifierOptions = new String[2][];
24 baseClassifierOptions[0] = Utils.splitOptions("-M");
25 baseClassifierOptions[1] = null;
26
27 CpCotraining ct = new CpCotraining(labeled, unlabeled);
28 ct.setOptions(options, baseClassifierOptions);
29 ct.initialize();
30
31 Classifier[] initialClassifier = ct.getCurrentClassifier();
32 Classifier[] finalClassifier = ct.run(-1);
33
34 // Teste dos classificadores...
```

Listagem 1. Executando o CoAL usando a função *melhoresExemplos* **NoContention**

Linhas 1-5. Pacotes que devem ser importados para executar o trecho de código entre as linhas 9 e 32. As classes do Weka estão no arquivo `weka.jar`, que acompanha as classes da implementação do CoAL. O usuário deve colocar o arquivo `weka.jar` no *classpath* do Java quando for executar esse trecho de código.

Linhas 9-20. Carregamento do conjunto de exemplos rotulados e não rotulados. No exemplo, assume-se que há duas descrições dos exemplos. Assim, uma das descrições do conjunto de exemplos rotulados é o objeto `labeled[0]`, carregado do arquivo `l1.arff`, e a outra é o objeto `labeled[1]`, carregado do arquivo `l2.arff`. O mesmo raciocínio se aplica ao conjunto de exemplos não rotulados.

Detalhes sobre o formato de arquivos ARFF, o qual é o formato de arquivo padrão para conjuntos de dados no Weka, podem ser encontrados no endereço http://www.cs.waikato.ac.nz/~ml/weka/arff.html[4]. Após o carregamento dos conjuntos de exemplos, é necessário informar qual é o atributo-classe. No exemplo dado, o atributo-classe é definido como o último atributo listado no cabeçalho dos arquivos de dados.

Importante: Os exemplos, rotulados ou não rotulados, devem aparecer na mesma sequência nos arquivos `l1.arff` e `l2.arff` e nos arquivos `u1.arff` e `u2.arff`. Dessa maneira, por exemplo, a n-ésima linha no arquivo `l1.arff` e a n-ésima linha no arquivo `l2.arff` devem corresponder ao mesmo exemplo, porém em descrições distintas. Além disso, o conjunto de atributos em `l1.arff` e o conjunto de atributos em `u1.arff` devem ser iguais. O mesmo acontece em relação a `l2.arff` e a `u2.arff`.

Linhas 22-25. Parâmetros da execução do CoAL. Na linha 22, são definidos os parâmetros do CoAL. A seguir estão descritos os parâmetros que podem ser ajustados.

`-C` Lista separada por ponto-e-vírgula de classes que implementam os algoritmos-base. No exemplo, será utilizado o algoritmo SMO (para treinar SVMs) para obter classificadores na primeira descrição e o algoritmo *Naive* Bayes para obter classificadores na segunda descrição. Caso se queira que um mesmo algoritmo seja utilizado em todas as descrições, basta informar a classe uma única vez. A

[4]Em grandes conjuntos de dados esparsos, para que a implementação possa ser executada com maior rapidez e com menos gasto de memória, recomenda-se usar o formato ARFF esparso, descrito no mesmo endereço.

classes que podem ser utilizadas como algoritmos-base são aquelas que estendem a classe `Classifier` do Weka e implementam o método `distributionForInstance()` de maneira adequada. Todas as classes do Weka que implementam algoritmos supervisionados para classificação estendem a classe `Classifier`, mas nem todas elas implementam o método `distributionForInstance()` para emitir um escore que reflita a confiança na classificação. A documentação da classe de interesse do usuário deve ser consultada em http://weka.sourceforge.net/doc.stable/ para saber se o método `distributionForInstance()` foi implementado de forma adequada.

`-T` Define um limiar no intervalo $(0,5; 1.0]$ para impedir a rotulação de exemplos classificados com uma confiança menor que o limiar.

`-L` Lista separada por vírgulas definindo a quantidade máxima de exemplos de cada classe que podem ser rotulados por iteração em cada descrição. O primeiro número se refere à primeira classe e assim por diante. A ordem das classes é definida no arquivo de onde são lidos os conjuntos de exemplos. Por exemplo, se um arquivo em formato ARFF define o atributo-classe como "@attribute class {positivo,negativo}", então "-L 5,3" indica que serão rotulados, por iteração, no máximo 5 exemplos da classe positiva e no máximo 3 exemplos da classe negativa em cada descrição.

`-U` Quando esta opção está presente, informa que o algoritmo CoAL deve ser executado em modo de simulação. No modo de simulação, os exemplos no conjunto de exemplos não rotulados — lidos de `u1.arff` e `u2.arff` — estão todos rotulados. Isso permite simular um oráculo durante o aprendizado ativo, além de permitir a coleta de informações sobre o desempenho da rotulação realizada pelo CoAL.

`-A` Define a quantidade máxima de consultas que o algoritmo CoAL pode fazer ao usuário. Quando este parâmetro não está definido, o aprendizado ativo em CoAL é desabilitado, o que equivale a executar o algoritmo Co-Training. *Importante*: Por enquanto, a implementação do CoAL só suporta aprendizado ativo via modo de simulação. Assim, a opção `-U` deve estar ativada.

`-NT` A implementação do CoAL utiliza múltiplas *threads* de execução por padrão. Caso seja necessário desativar o uso de múltiplas *threads*, esta opção deve estar presente.

`-S` Define a semente para o gerador de números aleatórios utilizado em caso de empates.

Linhas 27-30. Definição da função *melhoresExemplos* a ser utilizada e iniciação. Na linha 27, um objeto da classe `CpCotraining` é criado. Isso implica que a versão NoContention da função *melhoresExemplos* será utilizada. Se, ao invés, fosse criado um objeto da classe `OriginalCotraining`, então a versão Original da função *melhoresExemplos* seria utilizada.

Na linha 28, as opções são finalmente definidas. O método `setOptions()` pode ser chamado quantas vezes for necessário antes da chamada ao método `initialize()`, que inicia as estruturas de dados necessárias para a execução do CoAL. Após a chamada ao método `initialize()`, não é mais possível alterar os parâmetros do CoAL.

Linhas 31-32. Execução do CoAL. Após a chamada ao método `initialize()` e antes da chamada ao método `run()`, é possível obter os classificadores iniciais de CoAL, isto é, aqueles classificadores obtidos em cada uma das descrições do conjunto inicial de exemplos rotulados. Isso é feito com uma chamada ao método `getCurrentClassifier()`.

Na linha 32, o método `run()` faz com que o CoAL seja executado de acordo com os parâmetros que foram definidos pelo usuário. Esse método recebe como parâmetro um inteiro que define o número máximo de iterações do CoAL que devem ser realizadas. O valor -1 indica que a execução do algoritmo só deve parar quando não for mais possível rotular exemplos no conjunto de exemplos não rotulados. De qualquer maneira, o método `run()` retorna os classificadores obtidos ao fim da sua execução.

O método `run()` pode ser chamado quantas vezes for necessário até que não haja mais exemplos a serem rotulados. Nesse caso, uma chamada ao método `run()` retorna o valor **`null`**. Quando se deseja realizar somente uma iteração do CoAL, pode-se utilizar o método `iterate()`, que nada mais é que um atalho para a chamada `run(1)`.

4.3. Estimação de Desempenho

Após a execução do CoAL e da obtenção dos classificadores, uma das maneiras de se estimar o desempenho desses classificadores é verificando a capacidade preditiva dos mesmos em um conjunto de teste. No Weka,

isso pode ser feito de maneira similar ao trecho de código na Listagem 2.

Após carregar o conjunto de teste e definir o atributo-classe, um objeto `eval` da classe `Evaluation` é criado, e seu método `evaluateModel()` é chamado no conjunto de teste e em um classificador `classifier` obtido previamente. Após essa chamada, diversos métodos de `eval` podem ser chamados para obter medidas de desempenho associadas à capacidade preditiva do classificador. A documentação referente à classe `Evaluation` pode ser encontrada no endereço http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation.html.

```
1  import weka.classifiers.Classifier;
2  import weka.classifiers.Evaluation;
3
4  ...
5
6  Instances test = new Instances;
7
8  // Carregar o conjunto de teste e definir atributo-classe
9
10 Evaluation eval = new Evaluation(test);
11 eval.evaluateModel(classifier, test);
12
13 System.out.println(eval.errorRate());
```

Listagem 2. Estimação de desempenho de um classificador no Weka

Como mencionado, a implementação do CoAL pode ser executada em um modo de simulação. Assim, os rótulos atribuídos pelo CoAL a exemplos do conjunto U podem ser confrontados com os rótulos verdadeiros desses exemplos. Dessa maneira, é possível verificar se há degradação na rotulação dos exemplos, o que resultaria também na degradação dos classificadores obtidos.

As classes `CpCotraining` e `OriginalCotraining` oferecem métodos para obter medidas de desempenho de rotulação do CoAL. Esses métodos podem ser chamados depois da chamada ao método `initialize()`. A seguir estão as descrições desses métodos.

- `getIterationsSoFar()`: Retorna o número de iterações do CoAL realizadas até o momento da chamada.

- `getLabeledSoFar()`: Retorna o número total de exemplos rotulados por CoAL até o momento da chamada.
- `getIncorrectlyLabeledSoFar()`: Retorna o número de exemplos erroneamente rotulados por CoAL até o momento da chamada.
- `getCurrentUnlabeledCount()`: Retorna o número de exemplos ainda não rotulados.
- `getIncorrectlyVersusLabeled()`: Retorna uma lista de pares (i, l), nos quais i é o número de exemplos rotulados erroneamente quando l exemplos já tenham sido rotulados. Em cada iteração do CoAL, um par desse tipo é adicionado ao fim da lista.

5. Considerações Finais

Neste relatório foram descritas duas soluções propostas em (Braga, 2010) para lidar com o problema dos pontos de contenção no algoritmo Co-Training, bem como a implementação dessas soluções no ambiente Weka. A solução mais simples não permite que os pontos de contenção sejam rotulados, e foi incorporada ao Co-Training pela substituição da função $melhoresExemplos$ Original — responsável pela rotulação de exemplos no algoritmo Co-Training — pela função NoContention. A outra solução proposta em (Braga, 2010) é o algoritmo CoAL, o qual incorpora aprendizado ativo multidescrição ao algoritmo Co-Training. No CoAL, exemplos não rotulados identificados como pontos de contenção podem ser rotulados por um oráculo. Assim, esse exemplo rotulado pelo oráculo é inserido com o rótulo correto no conjunto de exemplos rotulados, "corrigindo" o classificador que o estava classificando erroneamente e com alta confiança.

Em (Braga, 2010) foi realizada uma avaliação experimental dos algoritmos Co-Training e CoAL executados nas versões Original e NoContention da função $melhoresExemplos$, usando a implementação descrita neste relatório. Os resultados dessa avaliação mostraram que o algoritmo CoAL, executado nas duas versões da função $melhoresExemplos$, apresentou os melhores resultados.

Referências

Abney, S. (2007). *Semisupervised Learning for Computational Linguistics*. Chapman & Hall. Citado na página 2.

Blum, A. e Mitchell, T. (1998). Combining labeled and unlabeled data with Co-Training. Em *COLT '98: Proceedings of the 11th Annual Con-*

ference on Computational Learning Theory, páginas 92–100. Citado nas páginas 1, 3, 7, 8, e 9.

Braga, I. A. (2010). Aprendizado semissupervisionado multidescrição em classificação de textos. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo. Citado nas páginas i, 1, 2, 10, 12, 15, e 22.

Braga, I. A., Matsubara, E. T., e Monard, M. C. (2009). Um estudo sobre a rotulação de exemplos no aprendizado semissupervisionado multivisão. Em *ENIA '09: Anais do XXIX Congresso da Sociedade Brasileira de Computação — Encontro Nacional de Inteligência Artificial*, páginas 1059–1068. Citado na página 10.

Collins, M. e Singer, Y. (1999). Unsupervised models for named entity classification. Em *EMNLP/VLC '99: Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, páginas 100–110. Citado na página 8.

Gupta, S., Kim, J., Grauman, K., e Mooney, R. (2008). Watch, listen & learn: Co-Training on captioned images and videos. Em *ECML/PKDD '08: Proceedings of the 2008 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, páginas 457–472. Citado nas páginas 2 e 9.

Kiritchenko, S. e Matwin, S. (2001). Email classification with Co-Training. Em *CASCON '01: Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research*, páginas 192–201. Citado nas páginas 2 e 8.

Koprinska, I., Poon, J., Clark, J., e Chan, J. (2007). Learning to classify e-mail. *Information Sciences*, 177(10):2167–2187. Citado nas páginas 2 e 9.

Laguna, V. e Lopes, A. A. (2009). Mult-view approach for semi-supervised scientific paper classification. Em *WAAMD '09: Anais do V Workshop em Algoritmos e Aplicações de Mineração de Dados*, páginas 26–33. Citado na página 3.

Maeireizo, B., Litman, D., e Hwa, R. (2004). Co-Training for predicting emotions with spoken dialogue data. Em *ACL '04: Proceedings of 42nd Annual Meeting of the Association for Computational Linguistics*, páginas 202–205. Citado na página 9.

Matsubara, E. T. (2004). O algoritmo de aprendizado semi-supervisionado Co-Training e sua aplicação na rotulação de documentos. Dissertação de Mestrado, Instituto de Ciências Matemáticas e

de Computação, Universidade de São Paulo. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-19082004-092311/. Citado nas páginas 7 e 10.

Matsubara, E. T., Monard, M. C., e Batista, G. E. (2005). Utilizando algoritmos de aprendizado semi-supervisionados multi-visão como rotuladores de texto. Em *TIL '05: Anais do III Workshop em Tecnologia da Informação de da Linguagem Humana*, páginas 2108–2117. Citado na página 9.

Matsubara, E. T., Monard, M. C., e Prati, R. C. (2006). On the class distribution labelling step sensitivity of Co-Training. Em *IFIP AI '06: Artificial Intelligence in Theory and Practice*, páginas 199–208. Citado na página 7.

Muslea, I., Minton, S., e Knoblock, C. (2002a). Active + semi-supervised learning = robust multi-view learning. Em *ICML '02: Proceedings of the 19th International Conference on Machine Learning*, páginas 435–432. Citado na página 13.

Muslea, I., Minton, S., e Knoblock, C. (2002b). Adaptive view validation: A first step towards automatic view detection. Em *ICML '02: Proceedings of the 19th International Conference on Machine Learning*, páginas 443–450. Citado na página 8.

Muslea, I., Minton, S., e Knoblock, C. A. (2000). Selective sampling with redundant views. Em *AAAI/IAAI '00: Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, páginas 621–626. Citado na página 11.

Muslea, I., Minton, S., e Knoblock, C. A. (2006). Active learning with multiple views. *Journal of Artificial Intelligence Research*, 27:203–233. Citado na página 11.

Witten, I. H. e Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2ł edição. Citado na página 15.