UMA ESTRATÉGIA HÍBRIDA PARA O PROBLEMA DE CLASSIFICAÇÃO MULTIRRÓTULO

TIAGO AMADOR COELHO

UMA ESTRATÉGIA HÍBRIDA PARA O PROBLEMA DE CLASSIFICAÇÃO MULTIRRÓTULO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Wagner Meira Júnior
Co-orientador: Ahmed Ali Abdalla Esmin

Belo Horizonte
Março de 2011

Coelho, Tiago Amador
C672e Uma estratégia híbrida para o problema de classificação multirrótulo / Tiago Amador Coelho. — Belo Horizonte, 2011
xvi, 59 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Wagner Meira Júnior
Co-orientador: Ahmed Ali Abdalla Esmin

1. Computação - Teses. 2. Mineração de dados (computação) - Teses. 3. Sistemas de Recuperação da informação(Computação) - Teses. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*72 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Uma estratégia híbrida para o problema de classificação multi-rótulo

TIAGO AMADOR COELHO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. WAGNER MEIRA JUNIOR - Orientador
Departamento de Ciência da Computação - UFMG

PROF. AHMED ALI ABDALLA ESMIN - Co-orientador
Departamento de Ciência da Computação - UFLA

PROF. THIAGO DE SOUZA RODRIGUES
CEFET-MG

PROF. ADRIANO ALONSO VELOSO
Departamento de Ciência da Computação - UFMG

PROFA. GISELE LOBO PAPPA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 29 de março de 2011.

Resumo

Este trabalho apresenta um novo método para resolver o problema de classificação multirrótulo, baseado no método de enxame de partículas, chamado *Multi Label K-Nearest Michigam Particle Swarm Optimization* (ML-KMPSO), que foi avaliado utilizando-se duas bases de dados reais. A aprendizagem multirrótulo se originou na categorização de textos, onde cada documento pode pertencer a várias classes simultaneamente. Neste trabalho é proposta uma nova abordagem híbrida, na qual o ML-KMPSO se enquadra. Esta abordagem é baseada em duas estratégias. A primeira estratégia é a divisão do problema multirrótulo em diversos problemas binários, sendo que para tal foi utilizado o *Michigan Particle Swarm Optimization* (MPSO) para resolvê-los, porém, esta estratégia não leva em consideração as correlações existentes entre as classes. Já a segunda estratégia tem como objetivo considerar as correlações existentes entre as classes utilizando o *Multi Label K-Nearest Neighbor* (ML-KNN). Avaliamos a performance do ML-KMPSO utilizando a base *Yeast* (classificação funcional de genes) e a base *Scene* (classificação semântica de cenas). Os resultados obtidos pelo ML-KMPSO se igualam ou superam algoritmos de classificação multirrótulo do estado da arte.

Palavras-chave: Classificação Multirrótulo, Método de Enxame de Partículas, Mineração de Dados.

Abstract

This work presents a new method for multi-label classification based on Particle Swarm Optimization, called Multi Label K-Nearest Michigan Particle Swarm Optimization (ML-KMPSO) and evaluates it experimentally using two real-world datasets. Multi-label learning first arose in the context of text categorization, where each document may belong to several classes simultaneously. In this work, we propose a new hybrid approach, ML-KMPSO. It is based on two strategies. The first strategy is the Michigan Particle Swarm Optimization (MPSO), which breaks the multi-label classification task into several binary classification problems, but it does not take into account the correlations among the various classes. The second strategy is Multi Label K-Nearest Neighbor (ML-KNN), which is complementary and takes into account the correlations among classes. We evaluated the performance of ML-KMPSO using two real-world datasets: Yeast gene functional analysis and natural scene classification. The experimental results show that ML-KMPSO produced results that match or outperform well-established multi-label learning algorithms.

Keywords: Multi-Label Classication, Particle Swarm Optimization, Data Mining.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Nos últimos anos houve um demasiado aumento na quantidade de dados disponíveis. Estes dados contêm valiosos conhecimentos ocultos que poderiam ser utilizados para análise, diagnóstico, simulação e/ou prognóstico do processo que gerou a base de dados associada (Han & Kamber [2001] Ultsch [1999]). Segundo Jain & Ghosh [2005], a utilização de um grande volume de dados de uma maneira eficiente é um grande desafio, o que leva a uma necessidade de se utilizar métodos semiautomáticos de extrair conhecimento destes dados. Esta necessidade levou ao surgimento da área de mineração de dados ou *data mining* (DM).

DM utiliza métodos de várias outras áreas, especialmente aprendizado de máquina (*Machine Learning*) e Estatística, para extrair conhecimentos e/ou padrões a partir de conjuntos de dados (Han & Kamber [2001]). As técnicas de DM estão agrupadas nas seguintes subáreas: regras de associação, agrupamento, classificação e previsão.

As regras de associação identificam grupos de dados que apresentam co-ocorrência entre si. Também encontram itens que determinam a presença de outros em uma mesma transação e estabelecem regras que correlacionam a presença de um conjunto de itens com um outro intervalo de valores para um outro conjunto de variáveis. Uma associação é normalmente representada por uma regra de associação do tipo $\mathcal{X} \Rightarrow \mathcal{Y}$, que implica numa relação de dependência entre os conjuntos de dados $\mathcal{X}$ e $\mathcal{Y}$. Assim, se $\mathcal{X}$ ocorre na base de dados, então $\mathcal{Y}$ também ocorre (com alguma relação a $\mathcal{X}$) (C.Gonçalves [2005]; Pierrakos et al. [2003]; Hipp et al. [2000]).

A técnica de agrupamento consiste em identificar classes de itens em uma base de dados de acordo com alguma medida de similaridade. Cada grupo, chamado *cluster*, consiste de objetos que são similares entre eles e diferentes dos objetos dos outros grupos. Sendo assim, a diferença entre os grupos é dada por um valor, como medidas de distância ou similaridade/dissimilaridade. Diferentemente da classificação e predição,

em que os dados estão previamente classificados, a análise de *clusters* trabalha sobre dados nos quais as classes não estão definidas (Pierrakos et al. [2003]).

Ao contrário da clusterização, o objetivo da classificação e previsão é identificar características distintas de classes pré-definidas, baseadas num conjunto de instâncias. Essa informação pode ser usada tanto para entender a existência dos dados, quanto para predizer como novas instâncias se comportarão (Phyu [2009]). Em outras palavras, em uma tarefa de classificação deve ser identificado o conjunto mínimo das características conhecidas de um determinado objeto que sejam suficientes para prever uma característica desconhecida.

Sendo assim, criam-se modelos (funções) que descrevem e distinguem classes ou conceitos, baseados em dados conhecidos, com o propósito de utilizar estes modelos para predizer a classe de objetos que ainda não foram classificados. Como a classificação é um processo de aprendizado supervisionado, o modelo construído baseia-se na análise prévia de um conjunto de dados de amostragem ou dados de treinamento, contendo objetos corretamente classificados (Han & Kamber [2001]).

Neste contexto, percebe-se que na grande maioria dos problemas de classificação encontrados os dados são mutuamente exclusivos, ou seja, pertencentes a apenas uma classe. Alguns problemas de classificação podem conter dados ambíguos, significando que algumas instâncias dos dados podem pertencer a mais de uma classe (rótulo) simultaneamente, motivando assim o surgimento da linha de pesquisa denominada classificação multirrótulo (Tsoumakas & Katakis [2007]).

Desta forma, observa-se recentemente um aumento significativo, da comunidade científica, no interesse pelo uso e aplicação da computação evolutiva em problemas de classificação, algoritmo genético (Ishibuchi et al. [1995]), redes neurais (Hansen & Salamon [1990]), colônia de formigas (Shelokar et al. [2004]), método de enxame de partículas (*Particle Swarm Optimization* - PSO) (Cervantes et al. [2009a]), entre outros.

Dentre esses, destaca o Método de Enxame de Partícula ou *Particle Swarm Optimization* (PSO), que é um método de otimização baseado em comportamento social de um bando pássaros proposto por Kennedy & Eberhart [1995]. O método foi descoberto através da simulação de um modelo social simplificado. O PSO pode ser usado para resolver uma vasta gama de diferentes problemas de otimização, incluindo a maioria dos problemas que podem ser resolvidos usando o Algoritmo Genético den Bergh [2001].

1.1 Motivação

Atualmente, problemas reais vêm requerendo cada vez mais classificação multirrótulo. Sendo a classificação multirrótulo um problema relativamente recente, deste modo não são encontrados muitos trabalhos na literatura atualmente.

O PSO, por sua vez, tem sido utilizado com sucesso para resolver diversos problemas (Pugh et al. [2005]; Omran et al. [2006]; Reddy & Kumar [2007]). Este método demostrou ser bastante promissor e se mostrou comparável em desempenho a *Simulated Annealing* e algoritmos genéticos (Nebti & Meshoul [2009], Esmin et al. [2008]).

Por serem recentes na literatura não se tem conhecimento do uso do método PSO em problema multirrótulo, o que confere um caráter original a esta dissertação.

1.2 Objetivo Geral

O objetivo geral deste trabalho é estudar o Método de Enxame de Partículas (PSO), o desenvolvimento e sua adaptação para o problema de classificação multirrótulo, demonstrando que o método se aplica ao problema, com resultados comparados experimentalmente aos métodos existentes na literatura.

1.2.1 Objetivos Específicos

Especificamente, pretende-se:

- estudar a classificação multirrótulo;
- estudar o método de enxame de partículas (PSO) e seu uso para classificação;
- adaptar o PSO para resolução de problemas de classificação multirrótulo;
- realizar experimentos e compará-los com outros métodos de classificação multirrótulo encontrados na literatura.

1.3 Organização

O trabalho está organizado da seguinte forma: o Capítulo 2 introduz os conceitos de classificação, classificação multirrótulo, métodos para abordagem do problema de classificação multirrótulo e os principais algoritmos encontrados na literatura para a resolução do problema. No Capítulo 3 é explicado o Método de Enxame de Partículas (*Particle Swarm Optimization* - PSO), abordando os métodos *Michigan* (MPSO) e

Pittsburg, como o PSO tem sido utilizado para a classificação (*Adaptive Michigan PSO* - AMPSO). O Capítulo 4 aborda o método proposto para o problema de classificação multirrótulo, explicando como foi a sua elaboração e o seu funcionamento. O Capítulo 5 apresenta os experimentos realizados e os resultados obtidos. Por fim, no Capítulo 6 é apresentada a conclusão do trabalho, bem como são sugeridos trabalhos futuros.

Capítulo 2

Classificação Multirrótulo

2.1 Considerações Iniciais

Nesse capítulo são introduzidos os conceitos de mineração de dados, classificação e classificação multirrótulo para melhor entendimento do problema abordado neste trabalho. São apresentadas as estratégias encontradas na literatura para resolver o problema, bem como a apresentação e discussão das métricas utilizadas na avaliação dos métodos de classificação multirrótulo.

2.2 *Mineração de Dados*

A partir da década de 1960, a tecnologia de armazenamento de dados tem passado por uma grande evolução, dos primitivos armazenamentos em arquivos para poderosos e sofisticados sistemas de banco de dados, materializados através do desenvolvimento dos Sistemas Gerenciadores de Bancos de Dados (SGBD's) Jain & Ghosh [2005].

A quantidade de dados armazenados vem crescendo cada vez mais nos últimos anos. Essa grande quantidade de dados armazenados contém valiosos conhecimentos ocultos, que poderiam ser utilizados para análise, diagnóstico, simulação e/ou prognóstico do processo que gerou a base de dados Han & Kamber [2001] Ultsch [1999].

Segundo Jain & Ghosh [2005], a utilização direta e eficiente de um um grande volume de dados é um grande desafio. Existe assim, uma necessidade de se utilizar métodos semiautomáticos para extrair conhecimento destes dados. Esta necessidade levou ao surgimento da área de mineração de dados ou *Data Mining* (DM).

Em seu trabalho Han & Kamber [2001] descrevem *Data Mining* (DM) como extração ou "mineração"de conhecimento de uma grande quantidade de dados. Sendo uma

área multidisciplinar, que utiliza de métodos de várias outras áreas, especialmente de aprendizado de máquina e Estatística, para extrair conhecimentos a partir de conjuntos de dados.

De acordo com Jain & Ghosh [2005], *Data Mining* (DM) é o núcleo de um processo mais amplo, conhecido como *Knowledge Discovery in Databases* (KDD). Além da etapa de DM, que efetivamente extrai o conhecimento a partir de dados, o processo de KDD inclui outras etapas como pré-processamento (ou preparação de dados) e pós-processamento (ou refinamento de conhecimento), como pode ser observado na Figura 2.1 . O objetivo do pré-processamento de dados é transformar os dados para facilitar a aplicação de uma ou várias técnicas de DM, enquanto que o objetivo dos métodos de refinamento do conhecimento é validar e aperfeiçoar o conhecimento descoberto. Idealmente, é necessário que o conhecimento deva ser não apenas preciso, mas também compreensível e interessante para o usuário.

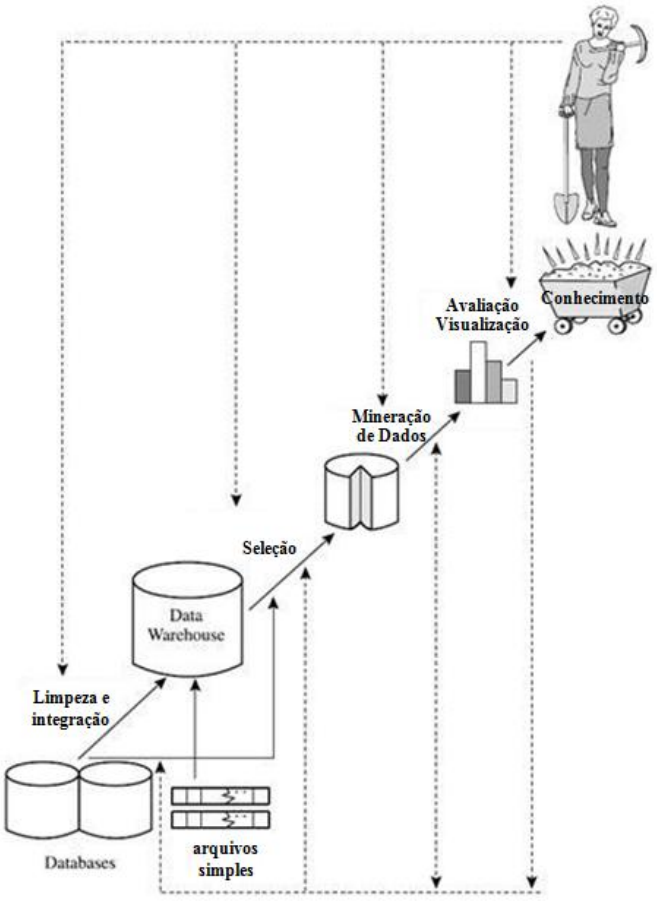

Figura 2.1. Etapas da Mineração de Dados adaptado de Han & Kamber [2001]

Segundo Jain & Ghosh [2005], o pré-processamento dos dados é necessário, na

maioria das vezes, para permitir que os dados sejam utilizados adequadamente no processo de DM. Para a realização desta etapa alguns passos devem ser seguidos:

- limpeza dos dados;
- integração dos dados;
- transformação dos dados;
- discretização dos dados;
- redução dos dados; e
- seleção dos dados.

A segunda etapa, DM, envolve a aplicação de algoritmos de associação, agrupamento, classificação e previsão, dependência, entre outros, sobre os dados já pré-processados para a extração de padrões úteis (Jain & Ghosh [2005]).

O último passo do KDD, segundo Jain & Ghosh [2005], é a interpretação do conhecimento, onde padrões extraídos devem ser interpretados adequadamente, de modo que eles possam ser usados para tomada de decisão, ou seja, para interpretar o conhecimento que os padrões estão armazenando. Padrões de representação do conhecimento são avaliados, ou seja, são corretamente identificados por medidas interessantes, tais como:

- não facilmente compreendidas;
- válidas em dados de testes com algum grau de certeza;
- potencialmente úteis;
- singulares; e
- válidas em termos da hipótese de que o usuário procurou confirmar.

2.3 Classificação

Classificação é uma das principais tarefas de DM. De acordo com Han & Kamber [2001], é também o processo de encontrar um modelo ou função que descreve e distingue classes de dados ou conceitos, a fim de ser capaz de usar o modelo para prever a classe de um objeto cuja classe é ainda desconhecida. O modelo derivado é baseado na análise de um conjunto de dados de treinamento.

Han & Kamber [2001] afirmam ainda que o modelo gerado pode ser representado de diversas formas, tais como:

- regras de classificação (IF-THEN) (Figura 2.2 a);
- árvores de decisão (Figura 2.2 b);
- fórmulas matemáticas; e
- redes neurais (Figura 2.2 c).

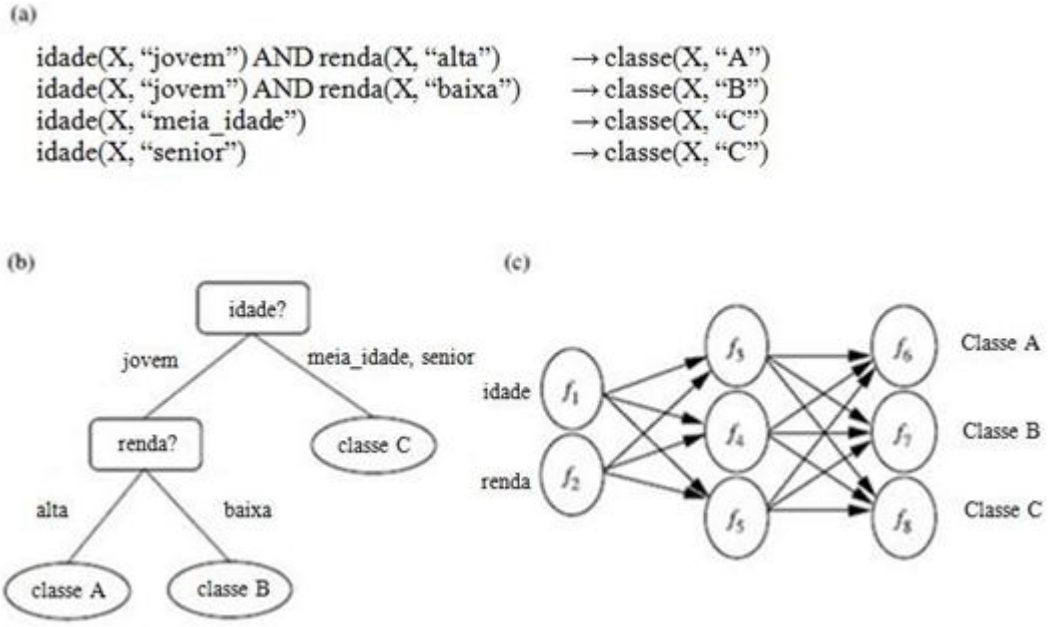

Figura 2.2. Formas de representação do modelo de classificação (a) regras de classificação, (b) árvore de decisão, (c) rede neural, adaptado de Han & Kamber [2001]

Holden & Freitas [2007] descrevem que uma regra de classificação é composta por um antecedente (valores de atributos) e um consequente (classe), como exemplificado a seguir:

$$IF < atributo = valor > AND...AND < atributo = valor > THEN < classe >$$

O antecedente é composto por um conjunto de termos, sendo cada termo uma tripla atributo - operador - valor, onde:

- valor é pertencente ao domínio do atributo;

- operador pode ser "="em caso de categoria/atributo nominal ou "$>, <, =, \geq, \leq$ "em caso de atributo contínuo.

A representação do padrão encontrado, em formato de regra, tem a vantagem de ser facilmente compreendido pelo usuário.

Regras de classificação podem ser consideradas como um tipo particular de previsão onde o antecedente de regra contém a combinação de condições para a previsão de atributos e o consequente da regra contém uma previsão do valor do atributo Han & Kamber [2001].

2.4 Classificação Multirrótulo - MLC

Existe uma grande quantidade de problemas em que alguns exemplos dos dados podem pertencer a mais de uma classe (rótulo) simultaneamente. Esses problemas são conhecidos como classificação multirrótulo (Trohidis et al. [2008]) (Tabela 2.1).

Tabela 2.1. Exemplo de base multirrótulo

Problema Multirrótulo	
Exemplo	Classe
1	B,C
2	A,B,C
3	C
4	B,C

Zhang & Zhou [2007] afirmam que pesquisas em classificação multirrótulo foram inicialmente motivadas pelas dificuldades encontradas para categorização de textos, já que diversos documentos pertenciam a vários rótulos.

Atualmente a classificação de textos é a principal área de aplicação de técnicas de classificação multirrótulo. Seu uso está presente também em reconhecimento de padrões e bioinformática.

Em um problema multirrótulo, cada elemento do conjunto de treinamento está associado a uma série de rótulos Y, onde Y $\subseteq$ L, sendo L um conjunto de rótulos.

Li et al. [2006] definem o problema de classificação da seguinte forma: Sendo $\mathcal{X}$ um conjunto de treinamento, $\mathcal{Y} = \{1, 2, ..., k\}$ o conjunto de rótulos. Dado um conjunto de treinamento da forma $\langle x_i, Y_i \rangle, x_i \in \mathcal{X}, Y_i \in 2^{|\mathcal{Y}|}$, onde $2^{|\mathcal{Y}|}$ são todas as combinações possíveis em $\mathcal{Y}$. O objetivo é aproximar uma função $f(x)$, tal que $f(x)$ retorne valores

de $2^{|\mathcal{Y}|}$ com o menor erro. A dificuldade de definir o erro em multirrótulo é que diversas combinações de rótulos são possíveis.

Na maioria dos casos, a abordagem multirrótulo induz uma ordenação dos possíveis rótulos de uma dada instância de acordo com $f(x, l_n)$. Então, formalmente podemos definir $rank_f(x, l)$ como $rank_f$ do rótulo l para a instância x, e se $f(x, l_1) \leq f(x, l_2)$ então $rank_f(x, l_1) \leq rank_f(x, l_2)$.

Tanto Trohidis et al. [2008] como Zhang & Zhou [2007], em seus trabalhos, agrupam em duas categorias os métodos para resolver o problema multirrótulo:

- a primeira categoria é mais intuitiva, sendo a decomposição do problema em múltiplos e independentes problemas de classificação binária ou unirrótulo, mais não leva em consideração as correlações existentes entre as classes;

- a segunda categoria seria a realização de alterações nos algoritmos de classificação existentes para permitir a classificação multirrótulo.

Para ilustrar os diferentes métodos que são encontrados na literatura para o problema múltirrótulo, de Carvalho & Freitas [2009] organizaram de maneira hierárquica todos esses métodos, como observado na Figura 2.3.

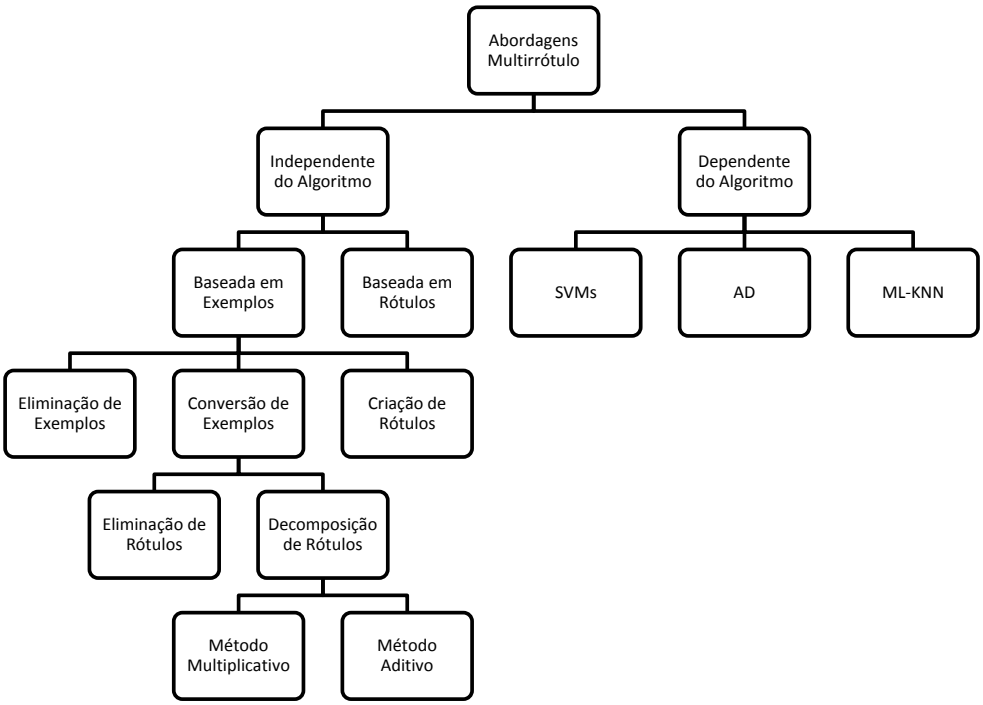

Figura 2.3. Técnicas para Classificação Multirrótulo adaptado de Cerri [2010]

A seguir serão discutidas essas abordagens.

2.4.1 Abordagem Independente do Algoritmo

A abordagem independente de algoritmo realiza a decomposição do problema multirrótulo em vários e independentes problemas de classificação binária ou unirrótulo. Essa decomposição é realizada baseando-se em exemplos ou em rótulos.

2.4.1.1 Baseada em Rótulos

Também conhecida como *Binary Relevance - BR*, técnica binária ou um-contra-todos (Tsoumakas & Vlahavas [2007]), ela divide o problema multirrótulo L em M problemas binários, para os quais são criados M classificadores, sendo M o número de classes do problema. Cada classificador criado é associado a uma classe e treinado para resolver um problema unirrótulo, como observado na Tabela 2.2.

Tabela 2.2. Tabelas unirrótulo resultantes da aplicação do método BR sobre a tabela multirrótulo da Tabela 2.1

Classificador A		Classificador B		Classificador C	
Exemplo	Classe	Exemplo	Classe	Exemplo	Classe
1	¬A	1	B	1	C
2	A	2	B	2	C
3	¬A	3	¬B	3	C
4	¬A	4	B	4	C

2.4.1.2 Abordagem Baseada em Exemplos

Como verificado na Figura 2.3, a decomposição baseada em exemplos é dividida em 3 grupos de estratégias:

- eliminação de exemplos multirrótulos ;
- criação de rótulos simples a partir de multirrótulos; e
- conversão de exemplos multirrótulos em exemplos unirrótulos.

Eliminação de Exemplos Multirrótulos

Sendo a mais simples das estratégias de decomposição baseada nos exemplos, porém a menos eficaz, a estratégia realiza a eliminação no conjunto de dados os exemplos que contêm mais de um rótulo. Com a eliminação destes exemplos, essa estratégia modifica o problema multirrótulo, transformando-o em um problema mais simples, o unirrótulo (Tabela 2.3).

Tabela 2.3. Demonstração da estratégia de eliminação de exemplos multirrótulos

Problema Multirrótulo	
Exemplo	Classe
1	B,C
2	A,B,C
3	C
4	B,C

→

Problema Unirrótulo	
Exemplo	Classe
3	C

Criação de Rótulos Simples a Partir de Multirrótulos

Chamada de *Label-Powerset*, realiza a criação de um novo rótulo para cada combinação de classes existentes no problema. Com isso o número de rótulos de classes existentes pode crescer exponencialmente e resulta em classes com pequenos números de instâncias de treino. Um exemplo da aplicação desta técnica pode ser visto na Tabela 2.4.

Tabela 2.4. Demonstração da estratégia de eliminação de exemplos multirrótulos

Problema Multirrótulo	
Exemplo	Classe
1	B,C
2	A,B,C
3	C
4	B,C
5	A
6	B

→

Problema Unirrótulo	
Exemplo	Classe
1	D
2	E
3	C
4	D
5	A
6	B

Conversão de Exemplos Multirrótulos em Exemplos Unirrótulos

Para a conversão de exemplos multirrótulos em exemplos unirrótulos, dois métodos diferentes podem ser aplicados. O primeiro, eliminação de rótulos, é a transforma-

ção exemplos multirrótulos em exemplos unirrótulos, utilizando a escolha de um dos rótulos que está associado ao exemplo e eliminando os demais rótulos. Essa escolha pode ser feita de maneira aleatória ou de forma determinista (Figura 2.4).

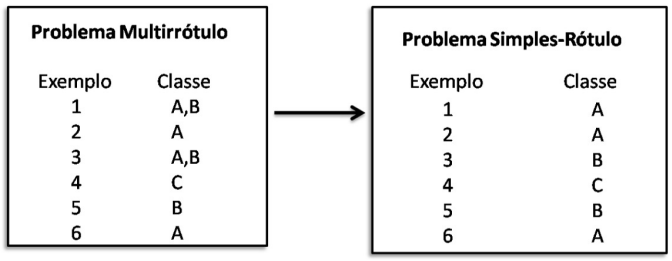

Figura 2.4. Método de eliminação dos rótulos (Vallim [2009])

O segundo método, decomposição dos rótulos, divide o problema multirrótulo em um conjunto de problemas unirrótulo. O método de decomposição dos rótulos é subdividido em método aditivo e método multiplicativo.

O método aditivo, ou *cross training*, foi proposto por Shen et al. [2004], onde, para cada exemplo, cada um dos possíveis rótulos será fixado em sequência. Por exemplo, dado uma instância com os rótulos A, B e C é treinado um classificador para a classe A e todos os exemplos que contêm o rótulo A se tornam exemplos unirrótulo para classe A (Figura 2.5).

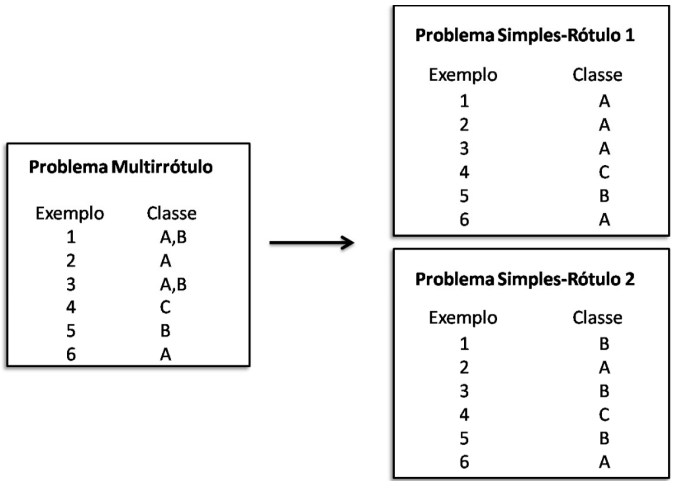

Figura 2.5. Método aditivo de decomposição dos rótulos (Vallim [2009])

O método multiplicativo divide o problema multiclasse em um conjunto de problemas binários e realiza a combinação de todos os possíveis classificadores. Sendo assim, é facilmente verificado que esse método não é escalável, uma vez que o número

de classificadores cresce exponencialmente com o número de rótulos dos exemplos, como observado na Figura 2.6.

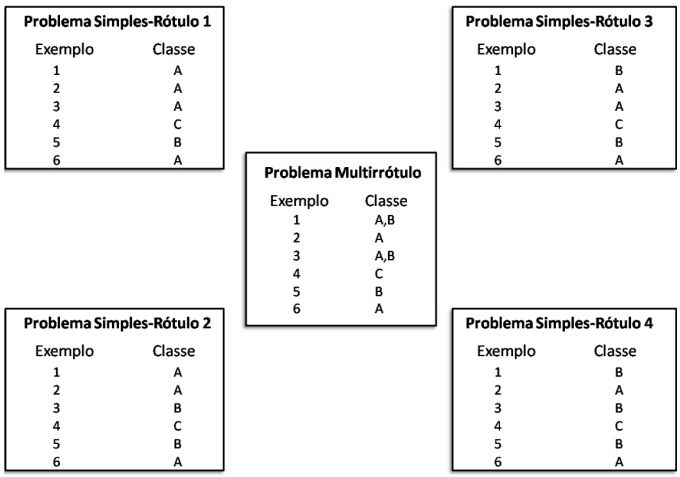

Figura 2.6. Método multiplicativo de decomposição dos rótulos (Vallim [2009])

2.4.2 Abordagem Dependente do Algoritmo

A abordagem dependente de algoritmo reune os algoritmos que foram criados para trabalhar com classificação multirrótulo e aqueles originalmente desenvolvidos para o problema de classificação unirrótulo que foram modificados para o problema.

Um dos métodos estado da arte para classificação multirrótulo é o BoosTexter. Proposto por Schapire & Singer [2000], o método mantém um conjunto de pesos para cada par exemplo-rótulo do conjunto de treinamento. Os pares que são difíceis de se prever corretamente têm os seus pesos incrementados e para os pares facilmente previstos os seus pesos são decrementados.

Elisseeff & Weston [2001] propuseram o RANK-SVM, um método baseado em *kernels* para classificação, que maximiza a soma das margens de todas as categorias simultaneamente, ordenando as categorias relevantes de cada instância da base de treinamento e penalizando as irrelevantes. Já Aiolli et al. [2004] desenvolveram um método em que o problema multirrótulo é dividido em várias SVMs binários em paralelo, combinando todos os classificadores por meio do compartilhamento da matriz kernel entre os diferentes classificadores.

2.4.2.1 ML-KNN

No trabalho de Zhang e Zhou (Zhang & Zhou [2007]) foi proposto o método *Multi Label K-Nearest Neighbors* (ML-KNN), derivado do método *K-Nearest Neighbors* (KNN)

tradicional e diferenciado pelo uso de probabilidades *a priori* e *a posteriori*.

No ML-KNN uma dada instância x está associada a um conjunto de rótulos $Y \subseteq \mathcal{Y}$, e k o número de vizinhos mais próximos. Seja $\vec{y}_x$ o vetor de categorias de x, onde o l-ésimo componente $\vec{y}_x(l)(l \in \mathcal{Y})$ tem valor de 1 se $l \in Y$ e 0 caso contrário. Adicionalmente, $N(x)$ denota o conjunto dos k vizinhos mais próximos de x, identificado no conjunto de treinamento. Baseado nos rótulos dos vizinhos de x, o vetor de rótulos vizinhos é definido como:

$$\vec{C}_x(l) = \sum_{a \in N(x)} \vec{y}_a(l), l \in \mathcal{Y} \tag{2.1}$$

onde $\vec{C}_x(l)$ conta o número de vizinhos de x contendo a l-ésima rótulo.

Para cada instância t, o ML-KNN inicialmente identifica os k vizinhos mais próximos $N(t)$ no conjunto de treinamento. Seja H_1^l o evento onde t contém o rótulo l, enquanto H_0^l o evento no qual t não contém l. Complementando, considere $E_j^l(j \in \{0, 1, 2, ..., k\})$ o evento que, de todos os k vizinhos de t, onde exatamente j instâncias contém o rotulo l. Portanto, baseado no vetor de rótulos vizinhos $\vec{C}_t$, o vetor de categorias $\vec{y}_t$ é determinado usando o seguinte princípio do *Maximum a Posteriori*:

$$\vec{y}_t(l) = \arg\max_{b \in \{0,1\}} P(H_b^l | E^l_{\vec{C}_t(l)}), l \in \mathcal{Y} \tag{2.2}$$

usando a regra de Bayes a equação acima pode ser reescrita como:

$$\begin{aligned} \vec{y}_t(l) &= \arg\max_{b \in \{0,1\}} \frac{P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l)}{P(E^l_{\vec{C}_t(l)})} \\ &= \arg\max_{b \in \{0,1\}} P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l) \end{aligned} \tag{2.3}$$

Como verificado na Figura 2.7, inicialmente o método identifica, para cada instância na base de treinamento, os seus k vizinhos mais próximos. Então, baseado na informação estatística *a priori* obtida do conjunto de rótulos destes vizinhos, é utilizado o princípio *Maximum a Posteriori* (MAP) para determinar o conjunto de rótulos da instância de teste.

2.4.3 Densidade e Cardinalidade de Rótulo

Nem todos os conjuntos de dados são igualmente multirrótulos. Em algumas aplicações, o número de rótulos em cada exemplo pode variar, sendo muito pequeno comparado ao conjunto de rótulos possíveis. Em outros casos pode acontecer o contrário. Isso pode

$[\vec{y}_t, \vec{r}_t] = \text{ML-KNN}(T, k, t, s)$
% Calcula a probabilidade *a priori* $P(H_b^l)$
para $l \in \mathcal{Y}$ **faça**
  $P(H_1^l) = (s + \sum_{i=1}^{m} \vec{y}_{x_i}(l))/(s \times 2 + m)$
  $P(H_0^l) = 1 - P(H_1^l)$
fim para
% Calcula a probabilidade *a posteriori* $P(E_j^l|H_b^l)$
Identifique $N(x_i), i \in \{1, 2, ..., m\}$
para $l \in \mathcal{Y}$ **faça**
  para $j \in \{0, 1, ..., k\}$ **faça**
    $c[j] = 0$; $c'[j] = 0$;
  fim para
  para $i \in \{0, 1, ..., m\}$ **faça**
    $\delta = \vec{C}_{x_i}(l) = \sum_{a \in N(x_i)} \vec{y}_a(l)$
    se $(\vec{y}_{x_i}(l) == 1)$ **então**
      $c[\delta] = c[\delta] + 1$
    fim se
    senão
      $c'[\delta] = c'[\delta] + 1$
    fim se
  fim para
  para $j \in \{0, 1, ..., k\}$ **faça**
    $P(E_j^l|H_1^l) = (s + c[j])/(s \times (k+1) + \sum_{p=0}^{k} c[p])$
    $P(E_j^l|H_0^l) = (s + c'[j])/(s \times (k+1) + \sum_{p=0}^{k} c'[p])$
  fim para
fim para
% Computa $\vec{y}_t$ e $\vec{r}_t$
Identifica $N(t)$
para $l \in \mathcal{Y}$ **faça**
  $\vec{C}_t(l) = \sum_{a \in N(t)} \vec{y}_a(l)$
  $\vec{y}_t(l) = \arg\max_{b \in \{0,1\}} P(H_b^l) P(E^l_{\vec{C}_t(l)}|H_b^l)$
  $\vec{r}_t(l) = P(E^l_{H_1^l|\vec{C}_t(l)}) = (P(H_1^l)P(E^l_{\vec{C}_t(l)}|H_1^l))/P(E^l_{\vec{C}_t(l)})$
  $= (P(H_1^l)P(E^l_{\vec{C}_t(l)}|H_1^l))/(\sum_{b \in \{0,1\}} P(H_b^l)P(E^l_{\vec{C}_t(l)}|H_b^l))$
fim para

Figura 2.7. Algoritmo ML-KNN

influenciar o desempenho de diferentes métodos classificadores multirrótulo (Tsoumakas & Katakis [2007]).

Para discutir o impacto dessas características, apresentamos a seguir os conceitos de densidade e cardinalidade de rótulo em um conjunto de dados.

Seja D um conjunto de dados multirrótulo que possui N exemplos $(x_i, Y_i), i = 1...N$ e $|L|$ o número de rótulos possíveis.

Cardinalidade de Rótulo: é o número médio de rótulos nos exemplos em D:

$$CR(D) = \frac{1}{N} \sum_{i=1}^{N} |Y_i|$$

Densidade de Rótulo: é o número médio de rótulos nos exemplos em D dividido

por $|L|$.

$$DR(D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i|}{L}$$

2.4.4 Métricas de Avaliação

Para avaliar o desempenho dos métodos de aprendizagem foram escolhidas as métricas comumente utilizadas na literatura nos problemas de classificação multirrótulo e que foram propostas por Schapire e Singer (Schapire & Singer [2000]).

Hamming Loss: avalia quantas vezes o par exemplo-rótulo é erroneamente classificado, ou seja, um rótulo não pertence ao exemplo e foi previsto, ou um rótulo que pertença ao exemplo e não foi previsto. Nesta métrica, quanto menor o valor do *Hamming Loss* melhor é a acurácia do classificador.

$$HammingLoss_S(h) = \frac{1}{P}\sum_{i=1}^{P}|h(x_i)\triangle Y_i| \tag{2.4}$$

onde $\triangle$ representa a diferença simétrica entre os conjuntos dos rótulos previstos ($h(x_i)$) e os rótulos corretos (Y_i).

One-error: avalia quantas vezes o melhor rótulo não pertence ao conjunto de possíveis rótulos.

$$One - error_S(f) = \frac{1}{P}\sum_{i=1}^{P}\left[[\texttt{arg}_{y\in\mathcal{Y}}\texttt{max}\ \ f(x_i,y)] \notin Y_i\right] \tag{2.5}$$

onde $i \in \mathcal{Y}$. O termo $[\texttt{arg}_{y\in\mathcal{Y}}\texttt{max}\ \ f(x_i,y)]$ representa o valor máximo da função $f()$. Quanto menor o valor do $One - Error_S(f)$ melhor é a performance.

Coverage: é definido como a distância em termos de rótulos no seu *ranking* para cobrir todos os possíveis rótulos de uma instância x.

$$Coverage_S(f) = \frac{1}{P}\sum_{i=1}^{P}\texttt{max}_{y\in Y}rank_f(x_i,y) - 1 \tag{2.6}$$

onde $i \in \mathcal{Y}$. Quanto menor o valor do *Coverage* melhor é a performance do classificador.

Ranking Loss: avalia a quantidade média dos rótulos que estão inversamente ordenados para a instância. Quanto menor o valor do *Ranking Loss* melhor é a performance do classificador.

$$RankLoss_S(f) = \frac{1}{P}\sum_{i=1}^{P}\frac{1}{|Y_i||\overline{Y}_i|}|(y_1,y_2)|f(x_i,y_1) \not< f(x_i,y_2),(Y_1,y_2) \in Y_i \times \overline{Y}_i| \quad (2.7)$$

Average Precision: avalia a proporção média de rótulos que ocupam, no *ranking*, uma posição superior a um dado rótulo, e que pertencem ao conjunto de rótulos desejados (de Carvalho & Freitas [2009]). Quanto maior o valor desta métrica melhor é a performance do classificador.

$$AveragePrecision_S(f) = \frac{1}{P}\sum_{i=1}^{P}\frac{1}{|Y_i|} \times \sum_{y \in Y_i}\frac{|\{y'|rank_f(x_i,y') \not< rank_f(x_i,y'),y' \in Y_i\}|}{rank_f(x_i,y)} \quad (2.8)$$

2.5 Considerações Finais

Neste capítulo foram introduzidos os conceitos para o entendimento do problema de classificação multirrótulo, as abordagens existentes na literatura para se conseguir resolver o problema e as métricas que são utilizadas para a avaliação dos métodos.

Capítulo 3

Computação Evolutiva

3.1 Considerações Iniciais

Computação Evolutiva é o nome dado para uma série de técnicas de resolução de problemas baseados em princípios de evolução biológica, tais como a seleção natural, sobrevivência dos mais aptos, reprodução e herança genética. Estas técnicas estão sendo cada vez mais amplamente aplicadas a uma variedade de problemas, que vão desde aplicações práticas na indústria e comércio de ponta da investigação científica (Eiben & Smith [2008]).

Neste capítulo são introduzidos os conceitos do processo evolutivo para o entendimento da Computação Evolutiva.

3.2 Algoritmos Evolucionários

Diversos paradigmas de Computação Evolutiva (EC) foram desenvolvidos, e para cada paradigma existem diferentes variações de algoritmos (Engelbrecht [2006]). O paradigma dos Algoritmos Evolucionários (EA) consiste em algoritmos de buscas estocásticos inspirados no processo de evolução natural de Darwin (Maimon & Rokach [2005]). Ainda, segundo Maimon & Rokach [2005], os EA inicialmente geram uma população de indivíduos, sendo cada um deles uma solução candidata do problema, que "evoluem"no sentido de soluções cada vez melhores para o problema. Este processo de evolução é implementado em forma de algoritmo de otimização. O pseudocódigo que demonstra o princípio dos EA pode ser visualizado na Figura 3.1.

As duas partes da teoria da evolução de Darwin, no qual a EC se baseia, são facilmente observadas no pseudocódigo do EA (Figura 3.1):

Seja $t = 0$ um contator de gerações.;
enquanto *condição de parada é verdadeiro* **faça**
  Avalia o *Fitness*, $f(x_i)$, para cada indivíduo, x_i, na população, $P(t)$;
  Realiza o cruzamento para a geração dos descendentes;
  Realiza a mutação na geração dos descendentes;
  Seleciona a população $P(t+1)$ da nova geração;
  Avança para nova geração $t = t + 1$;
fim enqto

Figura 3.1. Algoritmo Evolucionário

- A seleção natural ocorre na operação de cruzamento quando os melhores pares da geração atual são escolhidos para a geração dos descendentes e serem selecionados para a próxima geração.
- Mudanças aleatórias são realizadas através do operador de mutação.

Ao contrário da evolução biológica que não converge, os EA incluem um critério de convergência para finalizar o processo de otimização. Alguns critérios de parada são:

- foi alcançado um número de gerações estabelecidos;
- não nenhuma melhora do *fitness* de uma população, comparada as suas consecutivas gerações;
- encontrou-se o melhor indivíduo ótimo.

3.3 Representação

Na EC cada indivíduo representa uma solução candidata ao problema de otimização. As características de cada indivíduo são representadas por um cromossomo, também conhecido como genoma. As características referem-se às variáveis do problema de otimização, para que uma ótima atribuição seja procurada. Cada variável que precise ser otimizada é chamada de gene, a menor unidade de informação.

Um importante passo no *design* de um EA é encontrar uma representação apropriada para os cromossomos (solução candidata). A eficiência é a complexidade do algoritmo depende desta representação. Diferentes algoritmos de EA utilizam diferentes esquemas para representar o indivíduo de sua população (solução candidata). A maioria dos EA representam a solução candidata como um vetor.

3.4 População Inicial

Sendo os EA estocásticos, eles mantêm uma população de soluções candidatas , sendo assim algoritmos de busca baseados em população. O primeiro passo para que os EA solucionem um problema de otimização é a geração da população inicial. O padrão de se criar uma população inicial é atribuir valores aleatórios do domínio do problema para cada gene de cada cromossomo. O objetivo de atribuir valores aleatórios é que a população inicial seja uma representação uniforme de todo o espaço de busca. Caso algumas regiões do espaço de busca não sejam cobertas pela população inicial existem chances que elas sejam negligenciadas no processo de busca.

O tamanho da população inicial tem consequências em termos de complexidade computacional e habilidade de exploração. Um grande número de indivíduos aumenta a diversidade, melhorando a capacidade de exploração da população. Por outro lado porém, o maior número de indivíduos aumenta a complexidade computacional por geração. Enquanto o tempo de execução por geração aumenta, porém com as chances que com poucas gerações consigam encontrar a melhor solução. Com menor número de indivíduos o contrário acontece.

3.5 Função *Fitness*

Na teoria proposta por Darwin, indivíduos com melhores características têm a melhor chance de sobreviver e de se reproduzir. Para determinar a habilidade de um indivíduo do EA tem para sobreviver, uma função matemática é utilizada para quantificar o quão boa é a solução representada por aquele cromossomo (indivíduo). A função *fitness*, f, mapeia a representação do cromossomo em um valor escalar:

$$f : X^{n_x} \rightarrow \mathbb{R} \tag{3.1}$$

onde X representa o tipo do dado do cromossomo

As operações como seleção, cruzamento, mutação e elitismo usualmente utilizam-se do *fitness*.

3.6 Seleção

Ao final de cada geração é selecionada, dentre as soluções candidatas, uma nova população; esta servirá como a população a ser utilizada na próxima geração. A nova

população é gerada utilizando a aplicação de três técnicas (operadores) conhecidas como cruzamento, mutação e elitismo.

Em termos de cruzamento, indivíduos cujas variações se adaptam melhor ao ambiente terão maior probabilidade de sobreviver e se reproduzir, e assim as próximas gerações herdarão as características dos melhores indivíduos. No caso da mutação, os mecanismos de seleção se focam nos indivíduos mais fracos, assim a mutação desses indivíduos aumenta a sua chance de sobrevivência. Já no elitismo, os melhores indivíduos são selecionados para irem para a próxima geração.

Para a seleção dos indivíduos que irão compor a nova geração são utilizados os operadores de seleção. Estes são caracterizados por sua "pressão de seleção", a qual relata o tempo que é requerido para produzir uma população uniforme (Engelbrecht [2006]). Para essa operação existem na literatura diversos métodos. A seguir estão os mais utilizados:

- Aleatório
- Roleta
- Torneio
- Baseado em *Rank*
- Elitismo
- Hall da Fama

3.7 Operadores de Reprodução

Reprodução é o processo de se produzir descendentes a partir dos indivíduos selecionados, aplicando os operadores de cruzamento e/ou mutação. Cruzamento é o processo de criar um ou mais novos indivíduos a partir da combinação de dois ou mais indivíduos. Se a seleção foca nos indivíduos com o melhor *fitness*, pode ocorrer a convergência prematura e assim reduzir a diversidade em uma nova população.

Mutação é o processo que altera de maneira arbitrária uma ou mais características do indivíduos. O objetivo da mutação é a introdução de uma nova característica na população, aumentando assim a diversidade. A mutação deve ser aplicada com o cuidado para não desfigurar a qualidade da solução dos melhores indivíduos. Por conta disto esse processo é aplicado com uma baixa probabilidade. Alternativamente, o uso

da mutação pode estar relacionada inversamente ao *fitness* do indivíduo, quanto menor o *fitness* maior a chance de ocorrer a mutação.

A reprodução pode ser aplicada juntamente com a substituição, ou seja, os descendentes irão substituir os seus respectivos pais somente se o seu *fitness* for melhor.

3.8 Abordagem *Pittsburg*

A abordagem *Pittsburg* é uma abordagem de representação dos indivíduos. Recebeu esse nome por ter sido desenvolvida na Universidade de *Pittsburg* (Pila [2006]).

Nela cada indivíduo da população representa um conjunto de regas candidatas à solução do problema. Sendo assim, cada indivíduo contém vários conjuntos de regras, sendo que cada indivíduo pode representar uma solução mais homogênea do problema.

3.9 Abordagem *Michigan*

A abordagem *Michigan* surgiu proveniente de trabalhos de pesquisas realizados na Universidade de *Michigan*, daí a sua nomeação.

Segundo Maimon & Rokach [2005], nesta abordagem cada indivíduo representa uma única regra (parte da solução), que seria uma solução candidata dentre todas as regras (população). Pila [2006] diz que na abordagem *Michigan* a solução é dada por uma única regra evoluída e que representa o melhor indivíduo da população, entretanto nem sempre essa regra é capaz de representar a solução do problema. Existem outras formar de coevoluir todos os indivíduos de forma que a solução do problema é data pela população (todas as regras).

3.10 Considerações Finais

Neste capítulo foi apresentado o conceito da Computação Evolutiva, suas características, passando pelos operadores de seleção e reprodução e, por fim, foram introduzidas as abordagens *Pittsburg* e *Michigan*.

Capítulo 4

Método de Enxame de Partículas

4.1 Considerações Iniciais

Neste capítulo será apresentado o *Particle Swarm Optimization* (PSO) ou Método de Enxame de Partícula, sua utilização para realizar a classificação e uma adaptação conhecida como *Michigan Particle Swarm Optimization - MPSO*.

4.2 Método de Enxame de Partícula

Kennedy & Eberhart [1995], em seus trabalhos, propôem o *Particle Swarm Optimization* (PSO) ou Método de Enxame de Partícula, que é um método de otimização baseado em comportamento social de um bando de pássaros. O método foi descoberto através da simulação de um modelo social simplificado. O PSO tem as suas raízes em duas metodologias, Vida Artificial (A-life) e Computação Evolutiva. Um dos aspectos atrativos do PSO é a facilidade de implementação den Bergh [2001].

O PSO geralmente é enquadrado como Computação Evolutiva, juntamente com outros paradigmas como Algoritmos Genéticos, Programação Genética, Estratégias Evolutivas e Programação Evolutiva Omran et al. [2006].

O PSO tem sido utilizado com sucesso para resolver diversos problemas (Pugh et al. [2005]; Omran et al. [2006]; Reddy & Kumar [2007]). O método apresentou-se bastante promissor, onde mostrou ser comparável em desempenho a *Simulated Annealing* e algoritmos genéticos Nebti & Meshoul [2009]. De acordo com den Bergh [2001], o algoritmo PSO é estocástico e não necessita de informações gradientes derivadas da função erro, permitindo que ele possa ser usado em funções onde o gradiente não está disponível ou é computacionalmente caro para se obter.

Segundo Omran et al. [2005], o PSO mantém um enxame de soluções candidatas para o problema de otimização em consideração, onde cada partícula representa uma potencial solução para o problema. Ainda, segundo o autor, se o problema possuir n variáveis, cada partícula será representada por um ponto n-dimensional no espaço de busca. A qualidade da partícula é medida usando uma função de *fitness*, a qual diz o quão perto a partícula está da solução ideal (Omran et al. [2005]).

Um dos princípios do PSO é que há troca de informações entre os membros de um enxame, onde essa informação é usada para determinar a melhor partícula e sua posição no enxame, de modo que outras partículas possam utilizar desta para ajustar a sua posição (Omran et al. [2005]).

4.2.1 O Algoritmo PSO

O algoritmo PSO mantém um enxame de partículas, onde cada partícula representa uma potencial solução do problema. Sendo s o tamanho do enxame, cada partícula i é representada por um objeto com inúmeras características [BERGH, 2001]. Essas características são:

- $x_i(t)$: a posição da partícula i no instante t;
- $v_i(t)$: a velocidade da partícula i no instante t;
- $pbest(t)$: a melhor posição encontrada pela partícula i até o momento t;
- $gbest(t)$: a melhor posição encontrada pelo enxame de partículas até o momento t;
- ω: o peso de inércia (momento);
- c_1 e c_2: coeficientes de aceleração pessoal da partícula x que influenciam no tamanho máximo do passo que uma partícula pode dar em uma iteração;
- $rand_1$ e $rand_2$: variáveis aleatórias que variam entre 0 e 1.

A cada iteração a velocidade da partícula é atualizada de acordo com a seguinte equação:

$$v_{i,k}(t+1) = \omega v_{i,k}(k) + c_1 rand_{1,k}(pbest_{i,k}(t) - x_{i,k}(t)) + c_2 rand_{2,k}(pbest_{i,k}(t) - x_{i,k}(t))$$

Onde $v_{i,k}$ representa a *k-ésima* dimensão do vetor velocidade associado à *i-ésima* partícula. A velocidade é atualizada separadamente para cada dimensão da partícula e $rand_1$ e $rand_1$ contribuem para a natureza estocástica do algoritmo den Bergh [2001].

A inércia é que controla o impacto das velocidades anteriores na velocidade atual. A partícula se ajusta a sua trajetória com base em informações sobre seu melhor desempenho e no melhor desempenho dos seus vizinhos. A inércia também é usada para controlar o comportamento de convergência do PSO. A fim de reduzir esse peso ao longo das iterações, permitindo que o algoritmo possa explorar algumas áreas específicas, ela é atualizada de acordo com a seguinte equação Shi & Eberhart [1998]:

$$\omega = \omega_{max} - \frac{\omega_{min} - \omega_{max}}{iter_{max}} * iter$$

Onde ω_{min} e ω_{max} são os valores mínimos e máximos que a inércia pode assumir, e iter é a iteração atual que está o algoritmo enquanto $iter_{max}$ é o número total de iterações.

De acordo com Omran et al. [2005], a performance do PSO é sensível a valores de entrada ω, c_1 e c_2. Várias sugestões baseadas em estudos empíricos com bons valores para essas variáveis podem ser encontradas na literatura Omran et al. [2005]. A posição de cada partícula é atualizada usando o novo vetor de velocidade daquela partícula:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

O funcionamento do algoritmo se dá através de repetidas aplicações das equações de atualização da velocidade e posição, como ilustrado na Figura 4.1:

Inicializar as posições e velocidades das partículas aleatoriamente no espaço n-dimensional.;
para cada *partícula* $i \in [1..s]$ **faça**
  Atualiza a velocidade (v_i) da partícula i
  Atualiza a posição (x_i) da partícula i
  Calcula o *Fitness*
  se x_i *melhor que* $pbest_i(t)$ **então**
    $pbest_i(t) = x_i$
  fim se
  se $pbest_i(t)$ *melhor que* $gbest(t)$ **então**
    $pbest(t) = pbest_i(t)$
  fim se
fim para cada

Figura 4.1. Metodo de Enxame de Partícula - PSO

Para cada atributo posição $x_{i,j}$ é inicializado aleatoriamente com um valor pertencente ao intervalo $[-x_{max}, +x_{max}]$, para todo $i \in 1..s$ e $j \in 1..n$. O mesmo é realizado

para o atributo velocidade $v_{i,j}$, inicializado aleatoriamente com valor pertencente ao intervalo $[-v_{max}, +v_{max}]$, para todo $i \in 1..s$ e $j \in 1..n$. O critério de parada pode ser um número determinado de iterações ou outro critério dependendo do problema den Bergh [2001].

4.3 PSO e Classificação

Sousa et al. [2004] propuseram o uso do PSO para a descoberta de regras de classificação. Utilizando diversas variações de implementações do PSO como, *Discrete Particle Swarm Optimisation* - DPSO, *Linear Decreasing Weight Particle Swarm Optimization* - LSWPSO e o *Constricted Particle Swarm Optimization* - CPSO, o PSO teve comprovada a sua competitividade em descoberta de regras de classificação frente ao algoritmo genético e o J48.

De Falco et al. [2007] afirmam que o problema de classificação pode ser facilmente codificado como um problema de otimização multivariável, onde em um espaço multidimensional um centroide representaria um protótipo de classe, e a classificação pode ser vista como um problema de encontrar a posição ideal de todos os centroides, isto é, determinar a posição ótima de cada centroide.

Ainda em seu trabalho, De Falco et al. [2007] citam que para um problema de classificação com C classes e N parâmetros é sugerida a seguinte codificação:

- $(\vec{p}_i^1, ..., \vec{p}_i^C, \vec{v}_i^1, ..., \vec{v}_i^C)$ representa o *i-ésimo* indivíduo da população;
- $\vec{p}_i^j = \{\vec{p}_{1,i}^j, ..., \vec{p}_{N,i}^j\}$ posição do *j-ésimo* centroide, onde N indica as dimensões do componente da posição;
- $\vec{v}_i^j = \{\vec{v}_{1,i}^j, ..., \vec{v}_{N,i}^j\}$ velocidade do *j-ésimo* centroide, onde N indica as dimensões do componente da velocidade.

Sendo assim, cada indivíduo da população é composto por $2 * C * N$ componentes. Já Esmin [2007], em seu trabalho, utilizou o PSO para extrair regras *fuzzy*.

Encontram-se também, na literatura, trabalhos os quais utilizam de algoritmos híbridos com o PSO para resolver o problema de classificação. Holden & Freitas [2007] utilizaram um algoritmo híbrido PSO/Ant Colony para classificar dados biológicos. Sivakumari et al. [2009] utilizam um híbrido de PSO-SVM para melhorar a performance do classificador SVM.

No PSO padrão cada partícula do enxame é dada como uma possível solução. Em Cervantes et al. [2009b] foi proposta uma adaptação do PSO, na qual foi aplicada a

abordagem *Michigan* ao PSO padrão para a descoberta de regras de classificação. Essa adaptação foi denominada *Michigan Particle Swarm Optimizatiton* (MPSO). Nela cada partícula representa apenas uma parte da solução do problema e o enxame de partículas representa uma possível solução.

4.3.1 Michigan PSO

Cervantes et al. [2009b] propuseram em seu trabalho uma adaptação do PSO denominada Michigan PSO (MPSO). No método MPSO, cada partícula representa um protótipo em potencial para ser usado para classificar, utilizando a regra do vizinho mais próximo. Cada partícula representa uma classe e essas classes são pré-determinadas no momento da criação de cada partícula, sendo que o número mínimo de partículas do enxame é dado pelo número de classes presentes no conjunto de treino de um dado problema.

As vantagens do MPSO em relação ao PSO padrão se dão pela redução da dimensão do espaço de busca e um número variável de partículas que formam a solução.

Segundo Cervantes et al. [2009b], para esse novo método foi introduzido o uso da definição do vizinho de uma partícula e o conceito de força de atração e repulsão. Para tal, durante a movimentação da partícula no espaço de busca, cada partícula seleciona uma outra partícula de um conjunto "não-competidores" para ser o centro de atração, e uma segunda partícula é selecionada de um conjunto "competidores" para ser o centro de repulsão. Essas duas partículas são definidas dinamicamente em cada iteração de acordo com as classes das partículas. Nessa definição, partículas competem com partículas da mesma classe e cooperam com partículas de classes diferentes. O pseudo-algoritmo pode ser observado na Figura 4.2 para o seu melhor entendimento.

A redução do número de partículas é aplicada após o número máximo de iterações. Iniciando pela partícula com o pior *fitness*, é verificado para cada partícula se a sua retirada não afetar a taxa de acerto enxame (classificador); então a partícula é retirada. Com isso o algoritmo removerá um grande número de partículas, reduzindo o enxame. O enxame resultante do algoritmo é considerado a solução.

No PSO padrão a variável de atratabilidade social tenta mover as partículas para as posições que contêm o melhor *fitness*. Já o MPSO tende a evitar essa convergência de partículas, pois a partícula representa apenas uma parte da solução(um protótipo).

Para mudar este comportamento do PSO padrão, no MPSO foram introduzidas várias modificações baseadas no uso da classe de uma partícula e de sua vizinhança, para dividir o enxame de partículas em concorrentes e não concorrentes:

```
Carregue as instâncias de treino;
Inicializa o enxame;
Inserir N partículas para cada classe das instâncias de treino;
enquanto (max iteração ou taxa de sucesso de 100%) faça
    Calcular para cada classe, as partículas são competidoras e não competidoras ;
    para cada partícula faça
        Calcular o Fitness local;
        Calcular o fator de adaptação social;
        Encontrar a partícula de mesma classe e que esteja no conjunto de partículas
        competidoras (centro de repulsão);
        Encontrar a partícula de mesma classe e que esteja no conjunto de partículas não
        competidoras (centro de atração);
        Calcular a próxima posição da partícula baseada em sua velocidade anterior, sua melhor
        posição, centros de atração e repulsão;
    fim para cada
    Movimente as partículas;
    Atribuir classes às instâncias de treino, utilizando a partícula mais próxima;
    Avaliar a taxa de acerto da classificação feita pelo enxame;
    Se o enxame conseguiu um melhor resultado, armazena todas as posições das partículas como
    "melhor enxame";
fim enqto
Apagar do enxame de partículas aquelas partículas que, ao serem retiradas, não afetam a taxa de
acerto do enxame.;
```

Figura 4.2. Pseudo Algoritmo do MPSO

- Para cada partícula da classe C_i, partículas "não concorrentes" são aquelas que suas classes $C_j \neq C_i$ e que classificam pelo menos uma instância da classe C_i.
- Para cada partícula da classe C_i, partículas "concorrentes" são todas as partículas da classe C_i e que classificam pelo menos uma instância da classe C_i.

Durante a movimentação de cada partícula ela é atraída pela partícula "não concorrente" mais próxima do enxame, que se transforma no centro de atração do movimento. Sendo assim, as partículas "não concorrentes" orientam a busca de padrões de diferentes classes. E é repelida pela partícula "concorrente" mais próxima do enxame, que se transforma no centro de repulsão do movimento. Desta forma, as partículas "concorrentes" empurram umas às outras para encontrar novos padrões de suas classes em diferentes áreas do espaço de busca.

Para a realização da movimentação da partícula utilizando os centros de atração e repulsão, a fórmula da velocidade que era utilizada no PSO padrão teve que ser modificada para:

$$\begin{aligned} v_{i,k}(t+1) = \omega v_{i,k}(k) &+ c_1 rand_{1,k}(pbest_{i,k}(t) - x_{i,k}(t)) \\ &+ c_2 rand_{2,k} sign(a_{i,k}(t) - x_{i,k}(t)) Sf_i \\ &+ c_3 rand_{3,k} sign(x_{i,k}(t) - r_{i,k}(t)) Sf_i \end{aligned}$$

onde

- $x_i(t)$: a posição da partícula i no instante t;
- $v_i(t)$: a velocidade da partícula i no instante t;
- $pbest(t)$: a melhor posição encontrada pela partícula i até o momento t;
- $gbest(t)$: a melhor posição encontrada pelo enxame de partículas até o momento t;
- ω: o peso de inércia (momento);
- c_1 e c_2: coeficientes de aceleração pessoal da partícula x que influenciam no tamanho máximo do passo que uma partícula pode dar em uma iteração;
- c_3: coeficiente de repulsão;
- $rand_1$, $rand_2$ e $rand_3$: variáveis aleatórias que variam entre 0 e 1;
- a_i: centro de atração da partícula i;
- r_i: centro de repulsão da partícula i;
- Sf_i: fator de adaptação social, inversamente dependente do *fitness* da partícula.

Se a_i ou r_i não existir, os termos são ignorados.

Para se determinar a melhor posição encontrada pela partícula a função *fitness* é utilizada. Como cada partícula representa um protótipo, a função pega as instâncias mais próximas da partícula em todo o enxame e as classificam.

Para o cálculo da função *fitness* local, inicialmente se calcula o G_f e o B_f. O G_f avalia quantas instâncias a partícula conseguiu classificar corretamente, e o quão perto as partículas estão delas. Já o B_f verifica a quantidade das instâncias foram classificadas incorretamente.

$$G_f = \sum_{\{g\}} \frac{1}{d_{g,i} + 1.0}$$

$$B_f = \sum_{\{b\}} \frac{1}{d_{b,i} + 1.0}$$

sendo $\{g\}$ o conjunto de instâncias que a mesma classe da partícula; $\{b\}$ o conjunto de instâncias de classe diferente da partícula; $d_{g,i}$ e $d_{b,i}$ são as distâncias Euclidianas entre as instâncias e as partículas.

Utilizando do G_f e o B_f, o cálculo da função *fitness* local é realizado da seguinte forma:

$$LocalFitness = \begin{cases} 0 & if\{g\} = \{b\} = \emptyset \\ \frac{G_f}{N_P} + 2.0 & if\{b\} = \emptyset \\ \frac{G_f - B_f}{G_f + B_f} + 2.0 & outros\ casos \end{cases}$$

sendo N_P a quantidade de instâncias no conjunto de treinamento.

Na função de *fitness* duas contantes (1.0 e 2.0) são usadas para definir dois intervalos distintos para as partículas, de acordo com as características das instâncias que elas classificam:

- Partículas que classificam instâncias de suas classes (verdade positivo) e de classe diferente (falso positivo) contêm *fitness* no intervalo de [0.0, 2.0).

- Partículas que classificam todas as instâncias corretamente têm o seu *fitness* maior que 2.0.

O *Social Adaptability Factor* ou Fator de Adaptabilidade Social determina que as partículas se movimentem constantemente em direção aos seus vizinhos "não-concorrentes" e que afastem dos vizinhos "concorrentes". No entanto, partículas que estão com um bom *fitness* devem tentar melhorá-lo e assim influenciar as outras partículas.

$$Sf_i = \frac{1}{(Best\ Local\ Fitness_i + 1.0)}$$

Para a avaliação do enxame Cervantes et al. [2009b] utilizaram a seguinte taxa de sucesso de classificação:

$$Avaliação\ do\ Enxame = \frac{Classificados\ Corretamente}{(Total\ de\ Instâncias)} * 100$$

O método armazena o melhor enxame obtido durante o processo de treinamento e essa função é utilizada para a avaliação do melhor enxame obtido.

4.4 Considerações Finais

Neste capítulo foi demonstrado e exemplificado o Método de Enxame de Partículas e como foi idealizada uma adaptação denominada *Michigan Particle Swarm Optimization*, mostrando suas diferenças e vantagens.

Capítulo 5

O método proposto: ML-KMPSO

5.1 Considerações Iniciais

Neste capítulo será apresentado em detalhes o método proposto neste trabalho. Através de uma abordagem híbrida, criou-se um novo método capaz de realizar a classificação multirrótulo.

Em um problema multirrótulo, os dados podem pertencer a mais de uma classe simultaneamente, o que pode resultar em o classificador associar apenas uma parte dos rótulos pertinentes a um determinado exemplo. Nesta situação, o classificador é considerado parcialmente correto, pois, mesmo acertando alguns rótulos, ainda há rótulos que ele não conseguiu associar. Diferente do problema unirrótulo, neste tipo de problema há a possibilidade de o resultado do classificador esteja correto, incorreto ou parcialmente correto.

Para abordar o problema multirrótulo, o trabalho propõe o uso conjunto da abordagem *Michigan* do método de enxame de partícula (MPSO), juntamente com as informações estatísticas retornadas pelo método ML-KNN para prever os rótulos das novas instâncias.

Este capítulo apresenta o método proposto, denominado ML-KMPSO (*Multi-Label K Michigan Particle Swarm Optimization*), que realiza a classificação multirrótulo utilizando o enxame de partículas e as informações estatísticas retiradas dos dados de treinamento e do enxame.

5.2 Uma Abordagem Híbrida

Problemas de classificação multirrótulo são vistos como generalizações de problemas binários e multiclasses, sendo suas instâncias associadas a classes que não são mutuamente exclusivas. Estas características deixam o problema multirrótulo mais complexo e desafiador em termos de pesquisa.

Para a concepção de um classificador multirrótulo que consiga explorar a correlação existente entre as classes e que tenha precisão para classificar estas classes isoladamente, optamos por desenvolver uma nova abordagem. Como visto anteriormente, existem duas abordagens para o problema de classificação multirrótulo: dependente do algoritmo e independente do algoritmo. Neste trabalho foi proposta uma terceira abordagem, a híbrida (Figura 5.1).

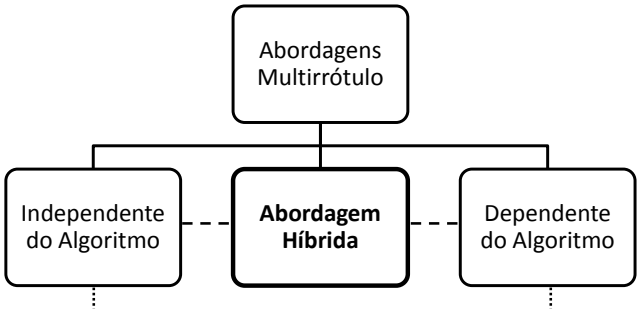

Figura 5.1. Nova Organização dos Métodos para Resolver o Problema Multirrótulo

No contexto da abordagem híbrida, foi desenvolvido o método ML-KMPSO (*Multi-Label K Michigan Particle Swarm Optimization*) que, como o próprio nome indica, foi criado a partir do ML-KNN, sendo um dos algoritmos clássicos de classificação multirrótulo, e do MPSO, que se mostrou bastante eficiente na criação de enxames especializados em classificação.

5.3 ML-KMPSO

O algoritmo ML-KMPSO foi desenvolvido utilizando-se duas estratégias. A primeira, MPSO, transforma o problema multirrótulo em diversos problemas de classificação unirrótulo, estratégia que não leva em consideração a correlação existente entre as classes e resulta na criação de modelos de classificadores altamente especializados para resolver o problema unirrótulo. A segunda estratégia é complementar, pois emprega

o ML-KNN para modelar a correlação entre as classes. Assim como o MPSO, o ML-KMPSO também gera um modelo de classificação, que uma vez treinado, pode ser utilizado para futuras classificações. Sendo assim método desenvolvido é classificado como local e global, pois utiliza um método local (ML-KNN) e um outro global (MPSO) para a resolução do problema de classificação multirrótulo.

5.3.1 Funcionamento do ML-KMPSO

O funcionamento do método proposto se dá da seguinte forma: é passado para o método um conjunto de treinamento, um para teste, a quantidade de vizinhos que serão utilizados (k) e a quantidade de partículas que serão produzidas para cada rótulo (Q).

Inicialmente o método calcula a probabilidade *a priori* para cada um dos rótulos existente no conjunto de treinamento e guarda essa informação para ser utilizada futuramente.

Após essa etapa o MPSO é utilizado sobre a base de treinamento, passando para ele a quantidade de partículas que serão criadas para cada rótulo existente, para a geração de classificadores especializados. Note que no passo 5 do MPSO (Figura 4.2), partículas que não são utilizadas são marcadas para serem retiradas da solução (enxame), iniciando por aquela que detém o pior *fitness*. Após a verificação de que esta ação não irá interferir na qualidade da classificação do enxame, a partícula é então retirada. O resultado do MPSO é um enxame de partículas especializadas em classificação unirrótulo. É importante ressaltar que o número máximo de partículas existentes no enxame é dado pela quantidade de partículas por rótulo (Q) multiplicado pelos rótulos existentes.

Retornando o enxame de partículas, o método ML-KMPSO realiza uma junção entre o enxame e as instâncias de treinamento. Observa-se assim que as instâncias de treinamento agora contém pontos (partículas) de referência, onde que naquele local o rótulo dominante é o da partícula. Por fim é calculada a probabilidade *a posteriori* dos rótulos existentes no conjunto de treinamento utilizando os (k) vizinhos.

Finalmente, utilizando as informações estatísticas obtidas dos rótulos do conjunto de vizinhos, incluindo as partículas resultantes do MPSO, é aplicado o princípio do *Maximum a Posteriori* (MAP) para determinar os rótulos que serão aplicados à instância de teste.

Na Figura 5.2 é demonstrado o pseudo algoritmo do método ML-KMPSO proposto.

1. Calcula a probabilidade a priori de todos os rótulos do conjunto de treinamento
2. Encontra o enxame de partículas usando o MPSO (Figura 4.2)
3. Junte o enxame de partículas com os dados de treinamento
4. Calcula a probabilidade a posteriori de todos os rótulos usando os k vizinhos
5. Determina os rótulos de cada instância de teste utilizando o MAP

Figura 5.2. Pseudo algoritmo do ML-KMPSO

A complexidade da metodologia apresentada neste trabalho é de ordem $O(n^3)$; isso ocorre devido a fase mais custosa do ML-KNN, a execução do algoritmo MPSO, que apresenta uma complexidade de ordem $O(n^3)$.

5.4 Considerações Finais

Neste capítulo foi demonstrado o método proposto, aqui chamado de ML-KMPSO, para resolver o problema de classificação multirrótulo utilizando de uma abordagem híbrida. Nesta nova abordagem uniu-se a estratégia de transformar o problema multirrótulo em diversos problemas unirrótuo, com uma estratégia complementar usando o ML-KNN para modelar a correlação entre as classes e, assim, realizar a classificação.

No capítulo seguinte serão apresentados os experimentos e os resultados obtidos utilizando-se o método proposto.

Capítulo 6

Experimentos e Resultados

6.1 Considerações Iniciais

Neste capítulo serão apresentados os experimentos aos quais o método híbrido proposto foi submetido e os resultados obtidos. Na literatura são encontradas diversas bases de dados, tanto reais quanto sintéticas. Para os experimentos foram utilizadas bases de dados reais, comumente presentes na avaliação de métodos para problemas multirrótulo. O resultado obtido pelo método foi comparado aos métodos ML-KNN, BoosTexter e Rank-SVM.

Este capítulo está organizado da seguinte forma: na Seção 6.2 serão apresentadas as bases de dados utilizadas nos experimentos. Em seguida, na Seção 6.3, são descritos como os resultados foram validados e avaliados. Já a Seção 6.4 descreve e analisa os resultados que foram obtidos pelo método proposto. Por fim, na 6.5 são feitas as considerações finais do capítulo.

6.2 Bases de Dados

Para a realização dos experimentos foram utilizadas duas bases de dados reais já pré processadas: a base *Yeast*(Elisseeff & Weston [2005]) e a base *Scene* (Boutell et al. [2004]).

6.2.1 Base *Yeast*

Yeast é uma base de dados multirrótulos biológica obtida através de estudos realizados sobre levedura *Saccharomyces cerevisiae*, um dos organismos mais estudados da literatura. A base é descrita como um conjunto de microvetores que expressam as

características e os perfis filogenéticos de 2417 genes, onde cada gene está associado a um conjunto de 14 classes funcionais.

No trabalho de Elisseeff & Weston [2005], a fim de se tornar mais fácil o uso da base, foi realizado o pré-processamento dela onde somente as estruturas funcionais das classes conhecidas são utilizadas.

Neste trabalho utilizou-se apenas as classes funcionais que estão no primeiro nível da hierarquia das classes funcionais dos genes da levedura, como observado na Figura 6.1, sendo que as que estão em destaque são as classes funcionais do gene YAL041W.

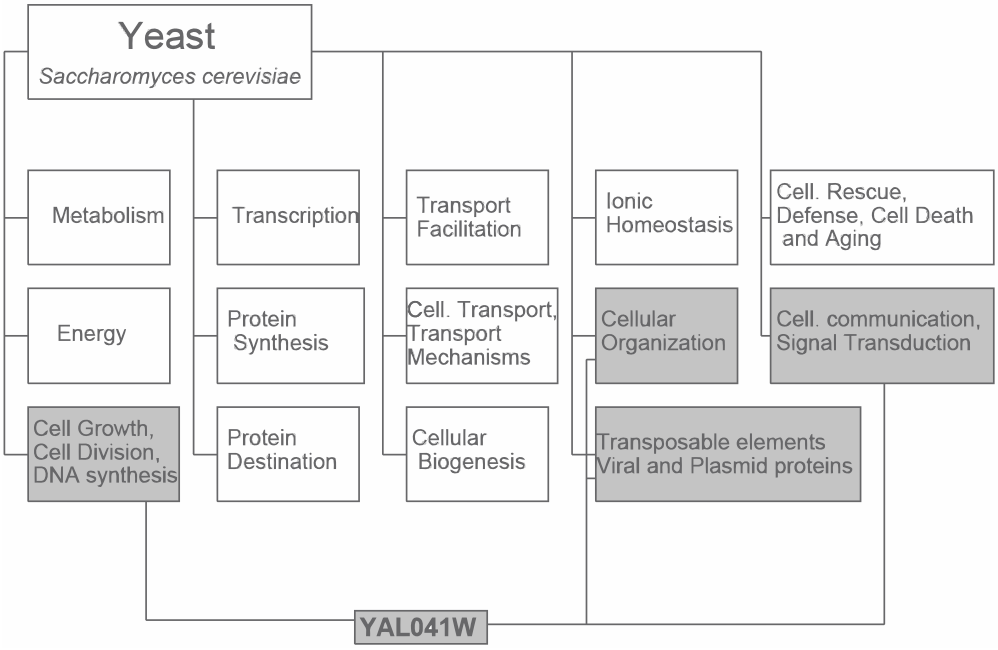

Figura 6.1. Classes Funcionais dos Genes da Levedura *Saccharomyces cerevisiae* Elisseeff & Weston [2005]

Em resumo, a base Yeast contém 2417 genes representados por um vetor de 103 dimensões, tendo 14 possíveis rótulos e uma média de 4.327 rótulos por gene.

6.2.2 Base *Scene*

Scene (Boutell et al. [2004]) é uma base de dados reais formada por imagens, onde cada uma delas está relacionada com diversas classes simultaneamente. Na Figura 6.2 são vistos dois exemplos de imagens multirrótulos, onde tem-se uma imagem A com montanhas e deserto, e uma segunda imagem B de um por do sol e oceano.

A base *Scene* contém, no total, 2000 imagens naturais, todas elas são classificadas manualmente e representadas por um vetor de 294 dimensões e 5 possíveis rótulos,

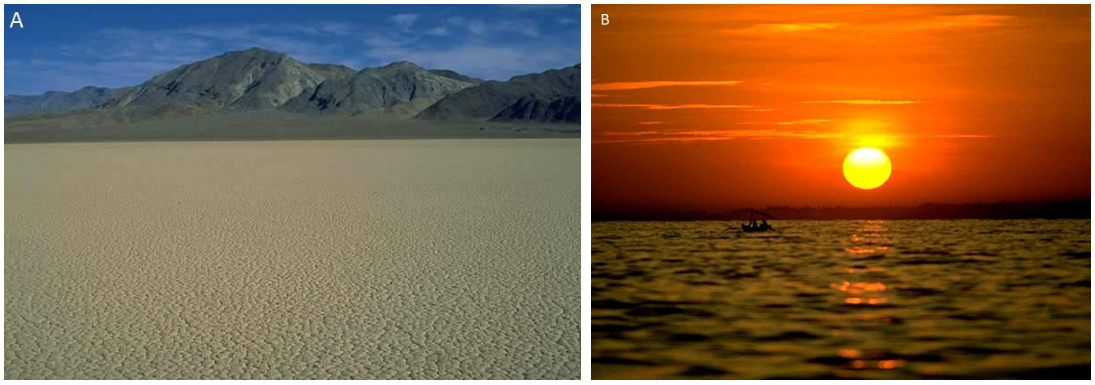

Figura 6.2. Figuras multirrótulos

cada imagem possuindo em média 1.24 rótulos. A Tabela 6.1 descreve em detalhes a distribuição dos rótulos deste banco de dados.

Tabela 6.1. Informações sobre a base de dados *Scene*

Classes	Total
Desert	340
Mountains	268
Sea	341
Sunset	216
Trees	378
Desert + mountains	19
Desert + sea	5
Desert + sunset	21
Desert + trees	20
Mountains + sea	38
Mountains + sunset	19
Mountains + trees	106
Sea + sunset	172
Sea + trees	14
Sunset + trees	28
Desert + mountains + sunset	1
Desert + sunset + trees	3
Mountains + sea + trees	6
Mountains + sunset + trees	1
Sea + sunset + trees	4
Total	2000

O objetivo do classificador nesta base é que, após o treinamento, ele possa classificar automaticamente imagens que ainda serão apresentadas a ele.

6.3 Avaliação dos Resultados

Para avaliar os resultados dos algoritmos presentes no trabalho foram utilizadas as métricas apresentadas na seção 2.3.4, que são: *Hamming Loss*, *One Error*, *Coverage*, *Rank Loss* e *Average Precision*. A fim devalidar estatisticamente os resultados obtidos pelos algoritmos foi utilizado o método conhecido como *k-fold cross-validation*.

Para este trabalho foi escolhido o uso de 10 *folds* para a validação; sendo assim o conjunto dos dados são divididos em 10 *folds*, onde a cada iteração do algoritmo são utilizados 9 *folds* como conjunto de treinamento e 1 *fold* como conjunto de teste. Este processo é repetido conforme o número de *folds*, sendo que a cada iteração é utilizado um novo *fold* para teste. Como resultados foram consideradas as médias, os valores mínimos e máximos de cada uma das métricas nas 10 iterações.

6.4 Resultados dos Experimentos

Para a realização dos experimentos foi adotada a mesma metodologia utilizada em outros trabalhos relacionados com o problema de classificação multirrótulo (Zhang & Zhou [2007]; Elisseeff & Weston [2001]; Santos et al. [2010]), para melhor comparação com o ML-KNN foram utilizados os mesmos valores de k utilizados em [Zhang & Zhou, 2007] e para fim de comparação os resultados dos algoritmos ML-KNN, BoosTexter e Rank-SVM foram retirados do trabalho de [Zhang & Zhou, 2007].

Em cada uma das bases utilizadas para a realização dos testes, os dados foram reordenados de maneira aleatória e divididos em dez subconjuntos, a partir destes subconjuntos foram realizados dez testes. Para cada teste um subconjunto era escolhido para servir como teste e os outros nove subconjuntos foram utilizados para o treinamento das metodologias avaliadas. Para o resultado foi considerada a média da execução dos dez testes realizados. Lembrando que após a execução do ML-KMPSO não foram verificados a diferença de vizinhança com o ML-KNN, após a inserção do PSO.

Os parâmetros utilizados pelo método ML-KMPSO nos experimentos estão relacionados na Tabela 6.2. Durante a realização destes experimentos foram utilizadas todas as possíveis combinações dos parâmetros e utilizado o método de validação *k-fold cross-validation*. Nesta tabela pode-se observar o número de vizinhos (k), a quantidade de partículas por classe (Q), os coeficientes da aceleração pessoal (c_1 e c_2), o coeficiente de repulsão (c_3) e a inércia (ω), e colocado nos resultados apenas aquele que obteve o melhor resultado.

Tabela 6.2. Parâmetros utilizados durante todos os experimentos

Parâmetros	Valores
Número de Vizinhos (k)	[8, 9, 10, 11, 12]
Quantidade de Partículas por Classe (Q)	[3, 5, 10, 15, 20]
Inércia (w)	[0.1]
Coeficientes de Aceleração (c_1 e c_2)	[1]
Coeficiente de Repulsão (c_3)	[0.25]

6.4.1 Resultados dos Experimentos na Base *Yeast*

Na Tabela 6.3 são apresentados os resultados dos experimentos obtidos pela execução dos quatro métodos utilizando a base *Yeast*.

Tabela 6.3. Resultados na Base *Yeast*

Métricas	Algoritmos			
	ML-KNN	BoosTexter	Rank-SVM	ML-KMPSO
Hamming Loss ↓	0.194 ± 0.010	0.220 ± 0.011	0.207 ± 0.013	0.198 ± 0.008
One-Error ↓	0.230 ± 0.030	0.278 ± 0.034	0.243 ± 0.039	0.225 ± 0.045
Coverage ↓	6.275 ± 0.240	6.550 ± 0.243	7.090 ± 0.503	6.355 ± 0.232
Ranking Loss ↓	0.167 ± 0.016	0.186 ± 0.015	0.195 ± 0.021	0.172 ± 0.009
Average Precision ↑	0.765 ± 0.021	0.737 ± 0.022	0.749 ± 0.026	0.762 ± 0.012

Na Figura 6.3 encontra-se um histograma do resultado do experimento para a métrica *Hamming Loss*, onde verificou-se que para esta métrica os resultados são bem próximos para os métodos ML-KMPSO e ML-KNN e, em média, inferiores aos métodos BoosTexter e Rank-SVM. Lembrando que essa métrica avalia quantas vezes o par instância-rótulo é erroneamente classificado, ou seja, um rótulo não pertence ao exemplo e foi previsto, ou um rótulo que pertença ao exemplo e não foi previsto

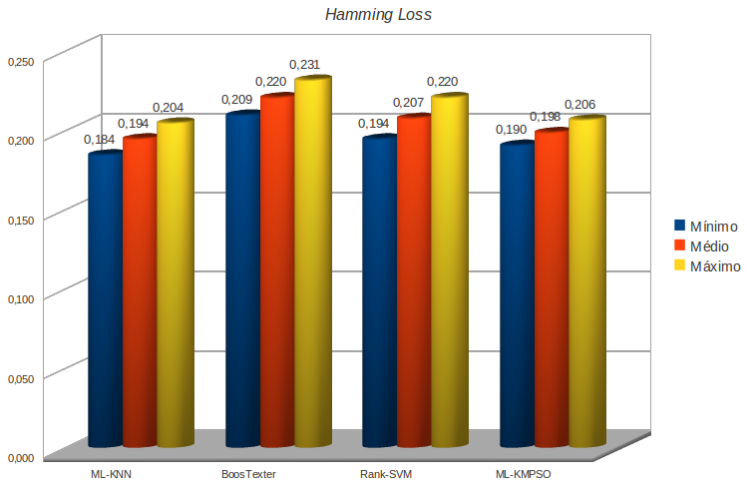

Figura 6.3. *Hamming Loss* na Base *Yeast*

Na métrica *One-Error*, visualizada na Figura 6.4, pode ser observado que o ML-KMPSO foi o método com a melhor performance dentre os avaliados. Mesmo tendo o maior desvio padrão comparado aos outros métodos, o ML-KMPSO se posiciona como o segundo melhor quando verificado apenas o pior caso (Máximo).

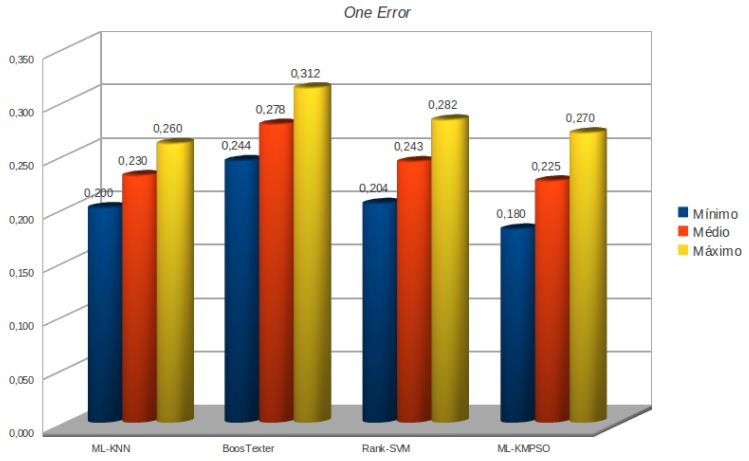

Figura 6.4. *One-Error* na Base *Yeast*

Observando-se o histograma da métrica *Coverage* (Figura 6.5), o ML-KNN foi o método que obteve o melhor resultado, seguido pelo ML-KMPSO e BoosTexter. O Rank-SVM foi o que obteve o pior desempenho.

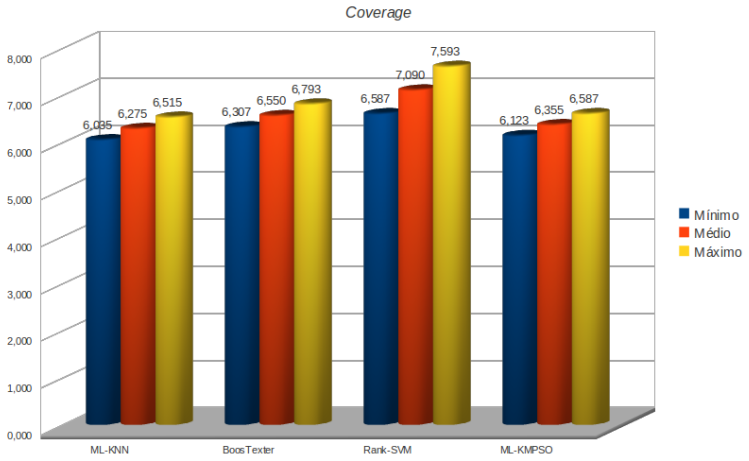

Figura 6.5. *Coverage* na Base *Yeast*

Na métrica *Ranking Loss* (Figura 6.6) o ML-KMPSO tem o segundo melhor resultado; verificando-se os piores casos, ele se posiciona como o melhor, juntamente com o ML-KNN.

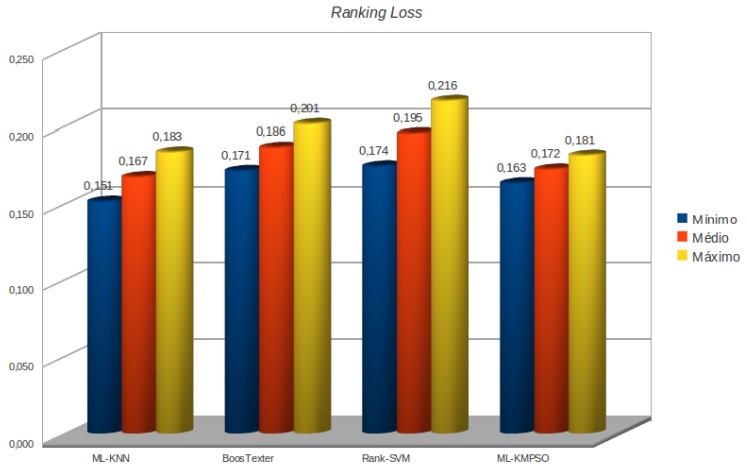

Figura 6.6. *Ranking Loss* na Base *Yeast*

A última métrica avaliada foi a *Average Precision*; nela o ML-KMPSO é estatisticamente semelhante ao ML-KNN, porém apresenta menor variação do desvio padrão. Sendo assim o melhor e o pior caso apresentam resultados próximos.

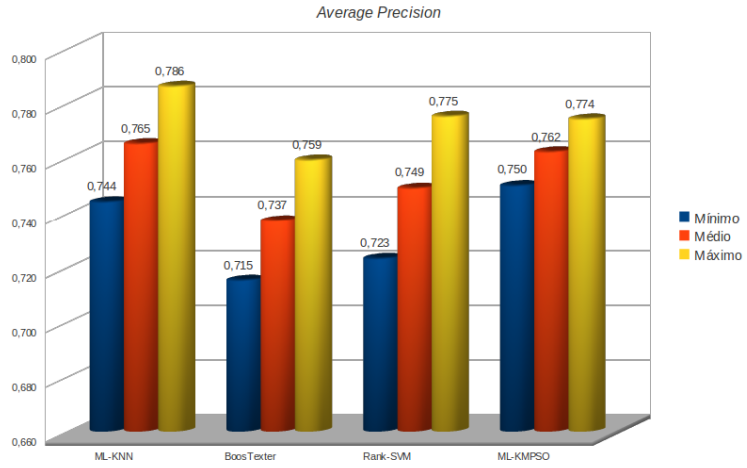

Figura 6.7. *Average Precision* na Base *Yeast*

Na Tabela 6.4ilustra a relação dos melhores para os piores métodos para cada uma das métricas.

Os resultados dos experimentos utilizando-se a base *Yeast* foram interessantes, apontando que o método proposto conseguiu se adaptar e realizar a classificação multirrótulo melhor que os métodos Rank-SVN e BoosTexter, se igualando estatisticamente ao método ML-KNN.

Tabela 6.4. Relação de Performance dos Métodos de Classificação Multirrótulo para a Base *Yeast*

Métricas	Algoritmos
HL	ML-KNN > ML-KMPSO > Rank-SVM > BootTexter
OE	ML-KMPSO > ML-KNN > Rank-SVM > BootTexter
Co	ML-KNN > ML-KMPSO > BootTexter > Rank-SVM
RL	ML-KNN > ML-KMPSO > BootTexter > Rank-SVM
AP	ML-KNN > ML-KMPSO > Rank-SVM > BootTexter

6.4.2 Resultados dos Experimentos na Base *Image*

A Tabela 6.5 mostra, de uma maneira resumida, os resultados dos experimentos nos quais os quatro métodos de classificação multirrótulo foram submetidos utilizando a base *Image*.

Tabela 6.5. Resultados na Base *Image*

Métricas	Algoritmos			
	ML-KNN	BoosTexter	Rank-SVM	ML-KMPSO
Hamming Loss ↓	0.169 ± 0.016	0.179 ± 0.015	0.253 ± 0.055	0.153 ± 0.010
One-Error ↓	0.300 ± 0.036	0.311 ± 0.041	0.491 ± 0.135	0.311 ± 0.036
Coverage ↓	0.939 ± 0.100	0.939 ± 0.092	1.382 ± 0.381	0.872 ± 0.274
Ranking Loss ↓	0.168 ± 0.024	0.168 ± 0.020	0.278 ± 0.096	0.173 ± 0.013
Average Precision ↑	0.803 ± 0.027	0.798 ± 0.024	0.682 ± 0.093	0.796 ± 0.018

Na Figura 6.8, o ML-KMPSO conseguiu se adaptar melhor à base *Image* em comparação à mesma métrica na base *Yeast*. Neste caso o método proposto se apresentou como o melhor método.

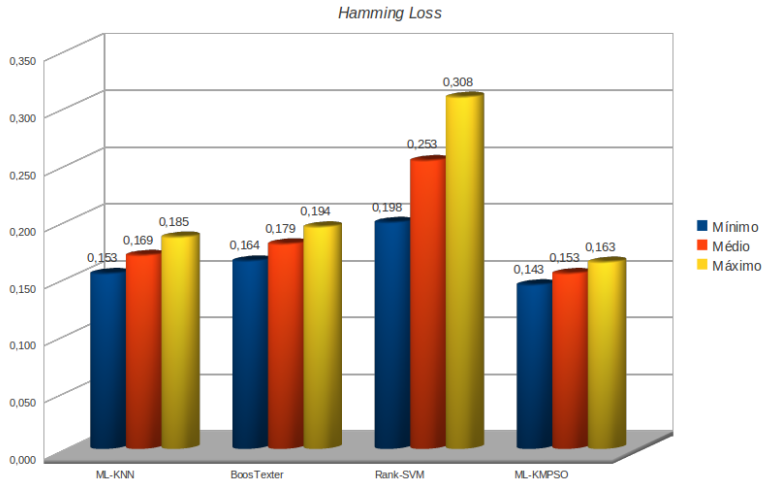

Figura 6.8. Average Precision na Base Image

Na métrica *One-Error* (Figura 6.9), o ML-KMPSO ficou empatado com o BoosTexter, sendo que no pior caso o método proposto teve um melhor resultado.

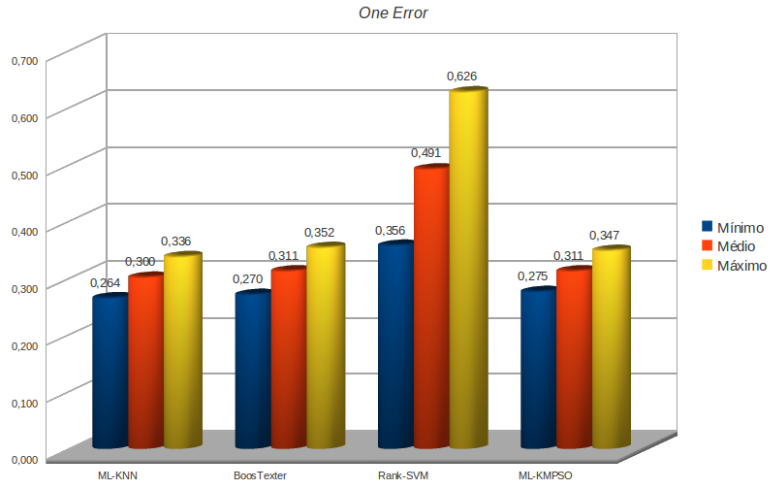

Figura 6.9. One Error na Base Image

O ML-KMPSO na métrica *Coverage*, como pode ser visualizado na Figura 6.10, conseguiu o melhor resultado dentre os outros métodos avaliados, porém com o desvio padrão alto. No pior caso tem um desempenho pior que o ML-KNN e o BoosTexter, mas compensando no melhor caso, sendo expressivamente eficiente comparado ao segundo melhor método.

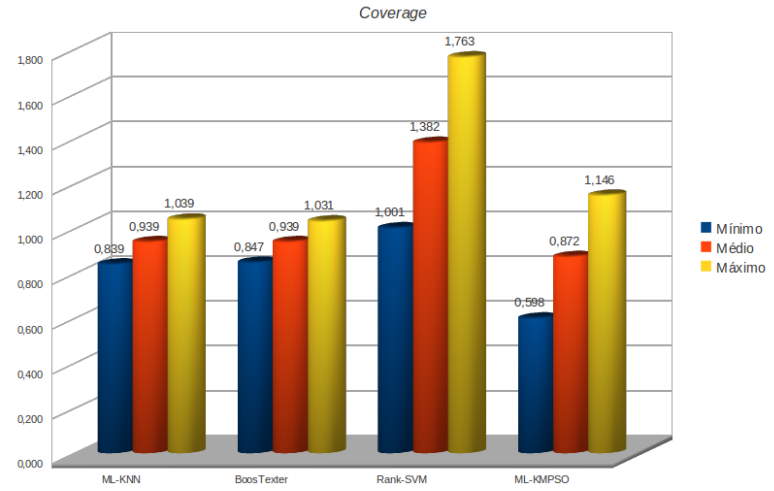

Figura 6.10. Coverage na Base Image

Conforme pode ser observado na Figura 6.11, referente à métrica *Ranking Loss*, o método Rank-SVM apresentou o pior desempenho, enquanto que os métodos ML-KNN e BoosTexter apresentaram os melhores desempenhos. Entretanto, o método ML-KMPSO apresenta um resultado favorável, atingindo um valor médio de 0,173, distante apenas 0,05 do melhor desempenho. Além disso, o ML-KMPSO apresenta

pouca variação de desempenho entre o melhor e o pior caso.

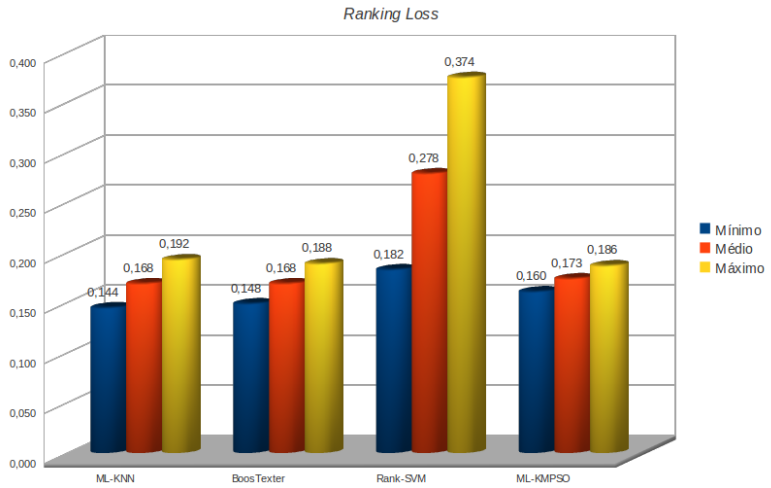

Figura 6.11. *Ranking Loss* na Base *Image*

Como se pode ver na Figura 6.12, os valores obtidos durante os experimentos para a métrica *Average Precision* pelos métodos ML-KNN, BoosTexter e ML-KMPSO são muito próximos, e o Rank-SVM conseguiu um resultado bastante expressivo.

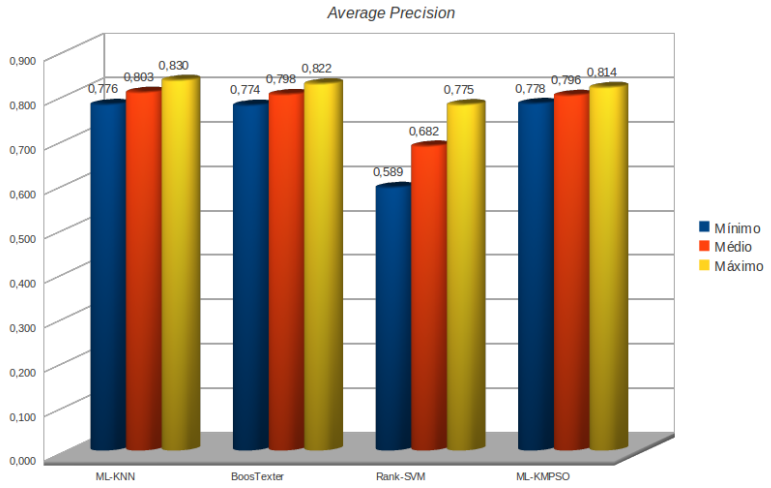

Figura 6.12. *Average Precision* na Base *Image*

Na Tabela 6.6 os métodos de classificação multirrótulo foram ordenados de acordo com a sua performance para cada uma das métricas de avaliação.

Tabela 6.6. Relação de Performance dos Métodos de Classificação Multirrótulo para a Base *Image*

Métricas	Algoritmos
HL	ML-KMPSO > ML-KNN > BootTexter > Rank-SVM
OE	ML-KNN > ML-KMPSO > BootTexter > Rank-SVM
Co	ML-KMPSO > ML-KNN > BootTexter > Rank-SVM
RL	ML-KNN > BootTexter > ML-KMPSO > Rank-SVM
AP	ML-KNN > BootTexter > ML-KMPSO > Rank-SVM

6.5 Considerações Finais

Neste capítulo foi apresentada e discutida a metodologia dos experimentos utilizados para avaliar o método ML-KMPSO proposto, bem como os resultados obtidos. A metodologia conduzida neste trabalho foi baseada na utilizada no trabalho de Zhang & Zhou [2007].

Para os experimentos foram escolhidas duas bases reais de classificação multirrótulo amplamente utilizadas na literatura.

Analisando os resultados obtidos pelo ML-KMPSO, verificou-se que foram bastantes promissores e de qualidade, e quando comparados aos resultados de algoritmos clássicos para o problema de classificação multirrótulo, como o Rank-SVM e ao BoosTexter, ele se mostrou superior e, ao se comparar com o ML-KNN, o método obteve um resultado estatisticamente semelhante.

Como observado nos resultados dos experimentos realizados, o método proposto obteve melhores resultados na base *Scene*. Sendo uma característica desta base a existência de uma quantidade expressiva de instâncias que possuem apenas um único rótulo, determinando uma baixa cardinalidade de rótulos. O método ML-KMPSO conseguiu se adaptar e obter melhores resultados por ter como uma de suas estratégias a criação de um enxame altamente especializado para classificação unirrótulo.

Capítulo 7

Conclusão

A classificação multirrótulo é um problema bastante recente e foi inicialmente motivado pelo desafio da categorização de textos, em que diversos rótulos poderiam ser associados a um único documento. Sendo assim, o problema de classificação multirrótulo pode ser visto como uma generalização do problema de classificação, sendo que, neste último, um documento só pode estar associado a apenas um único rótulo. Essa generalização deixa o problema mais difícil, complexo e intrigante.

Para se trabalhar o problema de Classificação Multirrótulo (MLC) e buscando em uma maneira de se obter um classificador que seja especializado, como os classificadores binários, e que consigam manter a correlação em que as classes que o problema MLC possam apresentar, foi proposto um método híbrido chamado de *Multi Label k Michigan Particle Swarm Optimization* (ML-KMPSO). O ML-KMPSO é derivado do *Michigan Particle Swarm Optimization* (MPSO) e do *Multi Label K-Nearest Neighbors* (ML-KNN). O MPSO se mostrou bastante eficiente na criação de enxames, onde cada partícula do enxame representa um protótipo em potencial para ser utilizado para classificar, e o ML-KNN que é um dos métodos clássicos para classificação multirrótulos.

Como apresentado no Capítulo 2.4, existem duas metodologias para se abordar o problema de MLC, uma dependente do algoritmo e outra independente do algoritmo. Mas, como o método proposto na neste trabalho se adequava em ambas as categorias foi proposta uma terceira metodologia, a híbrida.

O método neste trabalho reúne a ideia de se ter enxames com partículas especializadas em classificação, juntamente com a de extração de características e informações estatísticas dos vizinhos (dados de treino), para poder realizar a classificação multirrótulo.

O método ML-KMPSO foi avaliado utilizando duas bases de dados reais, a *Yeast* e a *Scene*. A base Yeast é uma base biológica da levedura *Saccharomyces cerevisiae*,

na qual se pretende encontrar as classes funcionais de cada gene. E a base Scene é uma base formada por imagens, onde se pretende realizar a classificação semântica das imagens. Para validação estatística do método foi utilizado o *k-fold cross validation* e foram utilizadas como métricas de avaliação o *Hamming Loss*, *One Error*, *Coverage*, *Rank Loss* e *Average Precision*. Após a realização dos experimentos, os resultados foram comparados aos seguintes algoritmos: ML-KNN, BoosTexter e Rank-SVM. Na comparação dos resultados obtidos, o nosso método se mostrou similar com o algoritmo ML-KNN e melhores que o BoosTexter e Rank-SVM.

A principal contribuição deste trabalho foi a criação do método ML-KMPSO que, com resultados bastante promissores, mostrou que a abordagem híbrida pode ser um dos caminhos para se trabalhar com o problema MLC.

Outra contribuição foi a proposta de uma metodologia híbrida, por não haver na literatura um método que se utilizava da abordagem dependente do algoritmo juntamente com a abordagem independente do algoritmo para resolver o problema MLC, abrindo um novo campo de pesquisa para desenvolvimento de novos métodos. A partir deste trabalho foi submetido e aceito um artigo no *Genetic and Evolutionary Computation Conference* - GECCO 2011.

Para trabalhos futuros deseja-se realizar vários estudos para melhorar o desempenho do método proposto, testando-o com outras bases multirrótulos e verificando o seu desempenho frente a elas. Outros estudos poderão ser realizados a fim de se verificar uma melhor configuração das variáveis do MPSO que foram pouco exploradas durante este trabalho, utilizar um método para a seleção de características ou até mesmo interpretar cada partícula como uma instância com pesos, podendo assim melhorar ainda mais os resultados.

Referências Bibliográficas

Aiolli, F.; Portera, F. & Sperduti, A. (2004). Speeding up the solution of multilabel problemas with support vector machines. pp. 118--121.

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

Cerri, R. (2010). Técnicas de classificação hierárquica multirrótulo.

Cervantes, A.; Galvan, I. M. & Isasi, P. (2009a). Ampso: A new particle swarm method for nearest neighborhood classification. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(5):1082--1091.

Cervantes, A.; Galván, I. M. & Viñuela, P. I. (2009b). Michigan particle swarm optimization for prototype reduction in classification problems. *New Generation Comput.*, 27(3):239–257.

C.Gonçalves, E. (2005). Regras de associação e suas medidas de interesse objetivas e subjetivas. *INFOCOMP Journal of Computer Science*, 4(1):26--35.

de Carvalho, A. & Freitas, A. (2009). *A tutorial on multi-label classification techniques*, volume Foundations of Computational Intelligence Vol. 5 of *Studies in Computational Intelligence 205*, pp. 177–195. Springer.

De Falco, I.; Della Cioppa, A. & Tarantino, E. (2007). Facing classification problems with particle swarm optimization. *Appl. Soft Comput.*, 7(3):652--658.

den Bergh, F. V. (2001). *An Analysis of Particle Swarm Optimizers*. PhD thesis, s. n.

Eiben, A. E. & Smith, J. E. (2008). *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer.

Elisseeff, A. & Weston, J. (2001). A kernel method for multi-labelled classification. In *In Advances in Neural Information Processing Systems 14*, pp. 681--687. MIT Press.

Elisseeff, A. & Weston, J. (2005). A kernel method for multi-labelled classification. In *Annual ACM Conference on Research and Development in Information Retrieval*, pp. 274--281.

Engelbrecht, A. P. (2006). *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons.

Esmin, A. A. A. (2007). Generating fuzzy rules from examples using the particle swarm optimization algorithm. In *HIS*, pp. 340–343.

Esmin, A. A. A.; Pereira, D. L. & Araújo, A. F. R. (2008). Study of different approach to clustering data by using the particle swarm optimization algorithm. In *IEEE Congress on Evolutionary Computation*, pp. 1817–1822.

Han, J. & Kamber, M. (2001). *Data mining: concepts and techniques*. Morgan Kaufmann, San Francisco.

Hansen, L. K. & Salamon, P. (1990). Neural network ensembles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(10):993--1001.

Hipp, J.; Guntzer, U. & Nakhaeizadeh, G. (2000). Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explor. Newsl.*, 2:58--64.

Holden, N. P. & Freitas, A. A. (2007). A hybrid pso/aco algorithm for classification. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pp. 2745--2750, New York, NY, USA. ACM.

Ishibuchi, H.; Nozaki, K.; Yamamoto, N. & Tanaka, H. (1995). Selecting fuzzy if-then rules for classification problems using genetic algorithms. *Fuzzy Systems, IEEE Transactions on*, 3(3):260 – 270.

Jain, L. C. & Ghosh, A. (2005). *Evolutionary Computation in Data Mining (Studies in Fuzziness and Soft Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Kennedy, J. & Eberhart, R. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pp. 1942--1948 vol.4.

Li, T.; Zhang, C. & Zhu, S. (2006). Empirical studies on multi-label classification. In *ICTAI '06: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, pp. 86--92, Washington, DC, USA. IEEE Computer Society.

Maimon, O. & Rokach, L. (2005). *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Nebti, S. & Meshoul, S. (2009). Predator prey optimization for snake-based contour detection. *International Journal of Intelligent Computing and Cybernetics*, 2(2):228 – 242.

Omran, M. G.; Engelbrecht, A. P. & Salman, A. A. (2005). Particle swarm optimization method for image clustering. *IJPRAI*, 19(3):297–321.

Omran, M. G.; Salman, A. A. & Engelbrecht, A. P. (2006). Dynamic clustering using particle swarm optimization with application in image segmentation. *Pattern Anal. Appl.*, 8(4):332–344.

Phyu, T. N. (2009). Survey of classification techniques in data mining.

Pierrakos, D.; Paliouras, G.; Papatheodorou, C. & Spyropoulos, C. D. (2003). Web usage mining as a tool for personalization: A survey. *User Modeling and User-Adapted Interaction*, 13:311--372.

Pila, A. (2006). História e terminologia a respeito da computação evolutiva. *Revista de Ciências Exatas e Tecnologia*, 1(1).

Pugh, J.; Zhang, Y. & Martinoli, A. (2005). Particle swarm optimization for unsupervised robotic learning. In *Swarm Intelligence Symposium*, pp. 92--99.

Reddy, J. M. & Kumar, N. D. (2007). Multi-objective particle swarm optimization for generating optimal trade-offs in reservoir operation. *Hydrological Processes*, 21(21):2897--2909.

Santos, A.; Canuto, A. & Neto, A. (2010). Evaluating classification methods applied to multi-label tasks in different domains. In *Hybrid Intelligent Systems (HIS), 2010 10th International Conference on*, pp. 61 –66.

Schapire, R. E. & Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135--168.

Shelokar, P. S.; Jayaraman, V. K. & Kulkarni, B. D. (2004). An ant colony classifier system: application to some process engineering problems. *Computers & Chemical Engineering*, 28(9):1577–1584.

Shen, X.; Boutell, M.; Luo, J. & Brown, C. (2004). Multi-label Machine Learning and Its Application to Semantic Scene Classification. In *International Symposium on Electronic Imaging*, San Jose, CA.

Shi, Y. & Eberhart, R. (1998). A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pp. 69--73.

Sivakumari, S.; Priyadarsini, R. P. & Amudha, P. (2009). Performance evaluation of svm kernels using hybrid pso-svm. *ICGST International Journal on Artificial Intelligence and Machine Learning, AIML*, 9:19--25.

Sousa, T.; Silva, A. & Neves, A. (2004). Particle swarm based data mining algorithms for classification tasks. *Parallel Comput.*, 30(5-6):767--783.

Trohidis, K.; Tsoumakas, G.; Kalliris, G. & Vlahavas, I. (2008). Multilabel classification of music into emotions. In *Proc. 9th International Conference on Music Information Retrieval (ISMIR 2008), Philadelphia, PA, USA, 2008.*

Tsoumakas, G. & Katakis, I. (2007). Multi label classification: An overview. *International Journal of Data Warehouse and Mining*, 3(3):1--13.

Tsoumakas, G. & Vlahavas, I. (2007). Random k-Labelsets: An Ensemble Method for Multilabel Classification. In *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*, pp. 406--417, Warsaw, Poland.

Ultsch, A. (1999). Data mining and knowledge discovery with emergent self-organizing feature maps for multivariate time series. In *in Kohonen Maps*, pp. 33--46. Elsevier.

Vallim, R. M. M. (2009). Sistemas classificadores evolutivos para problemas multirrótulo. Master's thesis, Universidade de São Paulo.

Zhang, M.-L. & Zhou, Z.-H. (2007). Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038--2048.

Anexo A

Artigo Publicado no GECCO 2011

Particle Swarm Optimization for Multi-Label Classification*

Tiago Amador Coelho
Universidade Federal de Minas Gerais - UFMG
tiago@dcc.ufmg.br

Ahmed Ali Abdalla Esmin
Universidade Federal de Lavras - UFLA
ahmed@dcc.ufla.br

Wagner Meira Júnior
Universidade Federal de Minas Gerais - UFMG
meira@dcc.ufmg.br

ABSTRACT

Multi-label classification learning first arose in the context of text categorization, where each document may belong to several classes simultaneously and has attracted significant attention lately, as a consequence of both the challenge it represents and its relevance in terms of application scenarios. In this paper, we propose a new hybrid approach, Multi Label K-Nearest Michigan Particle Swarm Optimization (ML-KMPSO), that is based on two strategies: Michigan Particle Swarm Optimization (MPSO) and ML-KNN. We evaluated the performance of ML-KMPSO using two real-world datasets and the results show that our proposal matches or outperforms well-established multi-label classification learning algorithms.

Categories and Subject Descriptors

I.5 [**PATTERN RECOGNITION**]: Models—*Statistical*

General Terms

Design, Experimentation, Performance

Keywords

Multi-Label Classification, Particle Swarm Optimization

1. INTRODUCTION

Real applications such as semantic scene classification, protein function classification, text categorization, and music categorization, among others, require increasingly multi-label classification methods. In a traditional classification problem, the examples are associated with a single label Y of a label set $\mathcal{Y}$, $|\mathcal{Y}| > 1$. However, there are several problems where data may belong to more than one class simultaneously, $|Y| \geq 2$, in which classes are not disjoint, called multi-label classification (MLC) problems.

In this paper, we propose a new hybrid approach, ML-KMPSO. It is based on two strategies. The first strategy is the Michigan Particle Swarm Optimization (MPSO), which breaks the MLC task into several binary classification problems, but it does not take into account the correlations among the various classes [1]. The second strategy is ML-KNN, which is complementary and takes into account the correlations among classes.

The performance of the ML-KMPSO is evaluated through two different multi-label learning problems. The experimental results show that ML-KMPSO produced results that match or outperform well-established multi-label learning algorithms.

*This work was partially supported by CNPq, CAPES and FAPEMIG.

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

2. MULTI-LABEL CLASSIFICATION

Research on MLC was initially motivated by the challenge it represents in text categorization, as many labels may be associated with the same document [5], which we call the multi-label problem. In [4], the multi-label classification problem is defined as follows: given a set of samples $\mathcal{X}$, a label set $\mathcal{Y} = \{1, 2, ..., k\}$, and a training set $\langle x_i, Y_i \rangle$, $x_i \in \mathcal{X}$, $Y_i \in 2^{|\mathcal{Y}|}$, where $2^{|\mathcal{Y}|}$ are all possible combinations of $\mathcal{Y}$. Learning a multi-label classifier $h : \mathcal{X} \rightarrow 2^{|\mathcal{Y}|}$ aims to maximize a function $f(x)$, such that $f(x)$ return values of $2^{|\mathcal{Y}|}$ with the smallest error.

In most cases, the multi-label approach induces an ordering of the possible labels of a given instance according to $f(x, l_n)$. Thus, we can formally define $rank_f(x, l)$ as $rank_f$ of label l for instance x, such that if $f(x, l_1) \leq f(x, l_2)$ then $rank_f(x, l_1) \leq rank_f(x, l_2)$.

Zhang and Zhou in [5] grouped the methods to solve the multi-label problem into two groups: The first category and more intuitive, is decomposition of the problem into multiple independent binary classification problems; The second category is algorithm adaptation for MLC problems. Our proposal falls into the second group.

3. PSO CLASSIFICATION

The Particle Swarm Optimization algorithm (PSO) is a population-based optimization method first proposed by [3]. The PSO technique finds the optimal solution using a population of particles. Each particle represents a candidate solution to the problem [3]. Recently, some work has already been done to adapt PSO to classification problems and most of it concerns rule-based classifiers. In [2] the PSO is used to extract fuzzy rules.

There is another method, called Michigan PSO (MPSO) that uses the Michigan approach. In the MPSO approach, a member of the population does not encode the whole solution to the problem, but only part of it. The whole swarm is the potential solution to the problem [1].

In this paper, we present an adaptation of MPSO to solve the MLC problem, called the ML-KMPSO. As the name

indicates, ML-KMPSO is derived from both ML-KNN and MPSO.

1. Computing the prior probabilities of all labels using training set
2. Computing swarm with MPSO and merge with training set
3. Computing the posterior probabilities of all labels using k neighbours using merged data
4. Determine the label set for each test instance using MAP

Figure 1: ML-KMPSO pseudo code

The algorithm ML-KMPSO (Figure 1) works as follows. First, it calculates the prior probabilities from the training set, then it generates the classification model using MPSO (step 2) and identifies a set of expert classification particles. According to statistical information gained from the labeled sets of these neighbour instances, i.e., the number of neighbor instances including the MPSO classification particles belonging to each possible class, it uses the maximum a posteriori (MAP) [5] principle to determine the label set for the test instance, which becomes the classification result.

4. EXPERIMENTAL EVALUATION

We used two real-world datasets in the experiments: the Yeast dataset and the Scene dataset. We present some statistics about the datasets in Table 1.

Table 1: Multi-label Statistics for the Datasets Used

		Attributes		Distinct	Number	Label
Dataset	Examples	Numeric	Labels	Subsets	Labels	Density
Yeast	2417	103	14	198	4.327	0.302
Scene	2000	294	5	15	1.24	0.248

To ensure more reliable results, all experiments employed the ten fold cross validation method. For the sake of evaluating the performance of the learning methods, we choose commonly-used criteria in MLC, as [5].

Table 2 presents the comparison of ML-KPMSO with state-of-the-art algorithms for the MLC problem for the dataset Yeast. For each evaluation criterion, down arrow indicates that the smaller the better, while up arrow indicates that the higher the better. As shown in this table, the ML-KMPSO performed better w.r.t. all five metrics when compared to BoosTexter and Rank-SVM. Considering ML-KNN, it was better just for the one-meter error, and was similar for most of the remaining metrics.

Table 2: Result on Yeast Data

	Algorithms			
Metrics	ML-KNN	BoosTexter	Rank-SVM	ML-KMPSO
HL↓	**0.194±0.010**	0.220±0.011	0.207±0.013	0.198±0.008
OE↓	0.230±0.030	0.278±0.034	0.243±0.039	**0.225±0.045**
Co↓	**6.275±0.240**	6.550±0.243	7.090±0.503	6.355±0.232
RL↓	**0.167±0.016**	0.186±0.015	0.195±0.021	0.172±0.009
AP↑	**0.765±0.021**	0.737±0.022	0.749±0.026	0.762±0.012

The results for the Scene dataset are presented in Table 3. ML-KPMSO was very promising, being the best considering the Hamming Loss and Coverage metrics, second in One-Error and third in Ranking Loss and Average Precision. The improvements associated with Coverage (Co) and Hamming Loss (HL) are a consequence of the low cardinality and density of the dataset, which favors MPSO, that comprises several expert classifiers and is able to predict accurately a larger number of tuples (samples x_i - labels Y_i). On the other hand, Rank-SVM provided the worst results.

Table 3: Result on Scene Data

	Algorithms			
Metrics	ML-KNN	BoosTexter	Rank-SVM	ML-KMPSO
HL↓	0.169±0.016	0.179±0.015	0.253±0.055	**0.153±0.010**
OE↓	**0.300±0.036**	0.311±0.041	0.491±0.135	0.311±0.036
Co↓	0.939±0.100	0.939±0.092	1.382±0.381	**0.872±0.274**
RL↓	**0.168±0.024**	0.168±0.020	0.278±0.096	0.173±0.013
AP↑	**0.803±0.027**	0.798±0.024	0.682±0.093	0.796±0.018

The experimental results show that the proposed algorithm outperforms other strategies such as Rank-SVN, BoosTexter and matches ML-KNN. We believe that further calibration to adjust the parameters of MPSO in the context of the proposed approach should enable better results than the ML-KNN.

5. CONCLUSION

This paper presented a new hybrid approach to solve MLC problems called ML-KMPSO, which is an adaptation of the MPSO the for Multi-label Problem. The paper has compared the ML-KMPSO to others well-established multi-label methods (ML-KNN, RankSVN and BoosTexter) and the experimental results using two real-world dataset show that our proposal outperforms or at least matches those methods, indicating that it is very promising.

As future work we plan to perform more experiments towards adjusting parameters and to use other multi-label data sets (like Yahoo web page categorization data set) to fully evaluate the efficacy of ML-KMPSO.

6. REFERENCES

[1] A. Cervantes, I. M. Galván, and P. I. Viñuela. Michigan particle swarm optimization for prototype reduction in classification problems. *New Generation Comput.*, 27(3):239–257, 2009.

[2] A. A. A. Esmin. Generating fuzzy rules from examples using the particle swarm optimization algorithm. In *HIS*, pages 340–343, 2007.

[3] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, 1995.

[4] T. Li, C. Zhang, and S. Zhu. Empirical studies on multi-label classification. In *ICTAI '06: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, pages 86–92, Washington, DC, USA, 2006. IEEE Computer Society.

[5] M.-L. Zhang and Z.-H. Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, July 2007.