Sistemas Classificadores Evolutivos para Problemas Multirrótulo

Rosane Maria Maffei Vallim

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 05/06/2009

Assinatura: _____

Sistemas Classificadores Evolutivos para Problemas Multirrótulo

Rosane Maria Maffei Vallim

Orientador: ***Prof. Dr. André C. P. L. F. de Carvalho***

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

USP - São Carlos
Junho/2009

Agradecimentos

Em primeiro lugar agradeço a Deus por ter me dado saúde e disposição para a realização do trabalho.

Aos meus pais e minha família por todo carinho, dedicação e amor que me dão e por serem meus companheiros e melhores amigos em todos os momentos.

Agradeço ao meu orientador, André Carvalho, por seus ensinamentos, sua compreensão e seu incentivo para que eu me candidatasse ao programa de trilha Graduação-Mestrado.

Ao professor David Goldberg por todas as suas idéias, pelas conversas que me faziam enxergar caminhos para solucionar os problemas e por me ensinar a importância de se fazerem as perguntas certas.

Ao Thyago Duque, por estar presente em todos os momentos, ouvindo e discutindo idéias a respeito do projeto, me ajudando com a escrita e revisão de artigos e ouvindo as minhas reclamações. Mas principalmente por seu carinho e companheirismo para todas as horas.

Ao professor Alex Freitas, por ter gentilmente cedido os conjuntos de dados de Bioinformática utilizados na fase experimental desse projeto.

Agradeço ao Xavier Llorà pelas explicações sempre oportunas sobre LCS, e aos demais membros do Illinois Genetic Algorithms Laboratory por sua amizade e apoio.

À Fapesp e ao CNPQ pelo apoio financeiro para a realização desse projeto.

Resumo

Classificação é, provavelmente, a tarefa mais estudada na área de Aprendizado de Máquina, possuindo aplicação em uma grande quantidade de problemas reais, como categorização de textos, diagnóstico médico, problemas de bioinformática, além de aplicações comerciais e industriais (Beasley, 2000). De um modo geral, os problemas de classificação podem ser categorizados quanto ao número de rótulos de classe que podem ser associados à cada exemplo de entrada. A abordagem mais investigada pela comunidade de Aprendizado de Máquina é a de classes mutuamente exclusivas. Entretanto, existe uma grande variedade de problemas importantes em que cada exemplo de entrada pode ser associado a mais de um rótulo ou classe. Esses problemas são denominados problemas de classificação multirrótulo.

Os *Learning Classifier Systems*(LCS) constituem uma técnica de Indução de Regras de Classificação que tem como principal mecanismo de busca um Algoritmo Genético. Essa técnica busca encontrar um conjunto de regras que tenha alta precisão de classificação, que seja compreensível e que possua regras consideradas interessantes sob o ponto de vista de classificação. Apesar de existirem na literatura diversos trabalhos sobre os LCS para problemas de classificação com classes mutuamente exclusivas, pouco se tem conhecimento sobre um LCS que seja capaz de lidar com problemas multirrótulo. Dessa maneira, o objetivo desta monografia é apresentar uma proposta de LCS para problemas multirrótulo, que pretende induzir um conjunto de regras de classificação que produza um resultado eficaz e comparável com outras técnicas de classificação. De acordo com esse objetivo, apresenta-se também uma revisão bibliográfica dos temas envolvidos na proposta, que são: Sistemas Classificadores Evolutivos e Classificação Multirrótulo.

Abstract

Classification is probably the most studied task in the Machine Learning area, with applications in a broad number of real problems like text categorization, medical diagnosis, bioinformatics and even comercial and industrial applications (Beasley, 2000). Generally, classification problems can be categorized considering the number of class labels associated to each input instance. The most studied approach by the community of Machine Learning is the one that considers mutually exclusive classes. However, there is a large variety of important problems in which each instance can be associated to more than one class label. This problems are called multi-label classification problems.

Learning Classifier Systems (LCS) are a technique for rule induction which uses a Genetic Algorithm as the primary search mechanism. This technique searchs for sets of rules that have high classification accuracy and that are also understandable and interesting on the classification point of view. Although there are several works on LCS for classification problems with mutually exclusive classes, there is no record of an LCS that can deal with the multi-label classification problem. The objective of this work is to propose an LCS for multi-label classification that builds a set of classification rules which achieves results that are efficient and comparable to other multi-label methods. In accordance with this objective this work also presents a review of the themes involved: Learning Classifier Systems and Multi-label Classification.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Siglas

AM: Aprendizado de Máquina.

AG: Algoritmos Genéticos.

AD: Árvores de Decisão.

EE: Estratégias de Evolução.

LCS: Learning Classifier Systems.

LTR: Long-term Reputation

MAP: Maximum a Posteriori Principle.

ML-OCS: Multi-label OCS

OCS: Organizational Classifier System

OGC: Organizational Growth Component

PF: Priority Factor

PG: Programação Genética.

RE: Reward Estimate

RL: Reinforcement Learning

RNA: Redes Neurais Artificiais

STR: Short-term Reputation

SVM: Support Vector Machines

CAPÍTULO

1

Introdução

1.1 Contextualização

A quantidade de dados gerados e armazenados tem crescido em velocidade cada vez maior. Essa imensa quantidade de dados guarda informações valiosas sobre o domínio de onde eles foram extraídos, podendo fornecer uma melhor compreensão de transações e processos e, em alguns casos, até mesmo uma vantagem competitiva. Entretanto, o que se observa é que, enquanto o volume de dados cresce continuamente, o número de analistas de dados não acompanha esse crescimento. Deste fato surge a clara necessidade em se desenvolver métodos (semi)automáticos para extrair o conhecimento contido nos dados brutos. Isso tem incentivado o desenvolvimento de diversas técnicas combinando os esforços das áreas de Aprendizado de Máquina (AM) e de Estatística, que são os pilares de uma área chamada de Mineração de Dados (MD). Algumas das técnicas mais conhecidas em MD podem ser agrupadas em quatro sub-áreas: classificação, regressão, associação e agrupamento. Nesse trabalho será investigada a sub-área de classifição.

A sub-área de classificação está associada com predição, ou seja, baseando-se em conhecimento prévio extraído de um conjunto de dados, procura-se descobrir para qual das classes existentes novos dados serão rotulados. A maioria dos problemas de classificação investigados em AM consideram classificação em classes mutuamente exclusivas. Entretanto, dificuldades causadas por ambiguidades em problemas de classificação de textos motivaram o surgimento de uma nova linha de pesquisa denominada de classificação multirrótulo (Tsoumakas e Katakis, 2007). Em um problema de classificação multirrótulo, as classes não são necessariamente exclusivas, podendo haver mais de uma classe associada a um dado exemplo. Apesar da motivação inicial para o problema ter

advindo da área de classificação de textos, existem outras áreas em que o problema de classificação multirrótulo se faz presente, como, por exemplo, a bioinformática.

Diversas técnicas têm sido utilizadas para classificação de dados, dentre elas podemos citar as Redes Neurais Artificiais (Haykin, 1999), as Árvores de Decisão (Quinlan, 1990), as Support Vector Machines (Smola et al., 1999; Lorena e Carvalho, 2003), os Conjuntos de Regras (Pappa e Freitas, 2006), entre outras. Dessas, apenas as Árvores de Decisão e os Conjuntos de Regras apresentam, ao final da classificação, o conjunto de regras induzido durante o processo de aprendizado, permitindo ao usuário analisar e compreender as relações existentes entre os dados de uma mesma classe. Apesar de as demais técnicas citadas não produzirem, originalmente, regras de classificação, existem muitos trabalhos que procuram extrair regras de classificadores considerados caixas-preta.

Uma área que vem sendo bastante estudada é a da utilização de técnicas da Computação Bioinspirada para induzir regras de classificação a partir de um conjunto de dados. Algoritmos Genéticos (AGs) (Goldberg, 1989), Algoritmos de otimização baseados em Colônias de Formigas (Dorigo e Stützle, 2004) e Sistemas Imunológicos Artificiais (de Castro e Timmis, 2002) são algumas das técnicas que têm sido utilizadas para induzir classificadores. Em especial, a técnica que utiliza AGs recebe a denominação de *Learning Classifier Systems*, e existem diversos trabalhos na literatura que abordam tal uso. Para o problema da classificação multirrótulo, as técnicas mais frequentes na literatura consistem em combinar diversos classificadores comuns para abordar o problema multi-rótulo ou realizar modificações internas nos algoritmos de classificação existentes para permitir este tipo de classificação (Tsoumakas e Katakis, 2007). Recentemente, estudos têm sido conduzidos para desenvolver técnicas de indução de classificadores por meio do uso de algoritmos evolutivos que sejam capazes de realizar classificação multirrótulo (Chan e Freitas, 2006). Entretanto, não se tem conhecimento da utilização de um AG para realização dessa tarefa.

1.2 Motivação

Uma vez que existem diversos problemas reais que se apresentam como problemas de classificação multirrótulo, e como essa é uma área que ainda não possui muitos algoritmos específicos para sua resolução, é importante que novos métodos sejam estudados e que seus resultados sejam comparados com os métodos disponíveis na literatura. Os LCS já possuem uma teoria muito bem fundamentada e vêm se mostrando capazes de alcançar bons resultados em classificação com classes mutuamente exclusivas. Outros algoritmos evolutivos utilizados para indução de classificadores também têm alcançado resultados animadores, demonstrando que essa é uma área de pesquisa promissora. Entretanto, apesar de existirem trabalhos de classificadores evolutivos para problemas multirrótulo (Chan e Freitas, 2006), pouco se tem conhecimento sobre um LCS com aplicação em problemas dessa natureza. Assim, abre-se um nicho de pesquisa original e relevante

a ser estudado, possibilitando a evolução da área de classificação multirrótulo e dos sistemas de classificação evolutivos.

1.3 Objetivo

O objetivo desta dissertação é apresentar uma nova alternativa para classificação multirrótulo utilizando a teoria dos LCS. Para a concretização dessa proposta, realiza-se antes uma revisão sobre os principais temas relacionados e que serviram como base para a construção do método proposto nesse trabalho.

1.4 Organização

Esta dissertação está organizada da seguinte forma. No Capítulo 2 são abordados os conceitos de classificação multirrótulo e os principais métodos existentes na literatura, como os indepedentes e dependentes de algoritmo. No Capítulo 3 são apresentados os conceitos fundamentais para entendimento dos LCS, assim como os principais trabalhos desenvolvidos na área. Nesse capítulo é realizada uma breve introdução aos AGs, principal mecanismo de busca dos LCS. O sistema de partilha de crédito em um LCS geralmente utiliza técnicas de Aprendizado por Reforço, e, por esse motivo, esse tópico também é brevemente discutido.

No Capítulo 4 é apresentado o método para classificação multirrótulo proposto nesse projeto de mestrado. Em seguida, no Capítulo 5, são apresentados a metodologia para experimentação do método proposto e os principais resultados obtidos. Finalmente, no Capítulo 6 são apresentadas as conclusões desse trabalho e sugestões para trabalhos futuros.

CAPÍTULO

2

Classificação Multirrótulo

2.1 Considerações Iniciais

Neste capítulo é realizada uma breve introdução ao problema de classificação multirrótulo e às estratégias existentes na literatura para resolvê-lo.

Primeiramente, na Seção 2.2, é realizada uma introdução ao Aprendizado de Máquina. Na Seção 2.3 é apresentada uma definição de classificação de dados. Em seguida, na Seção 2.4, o problema de classificação multirrótulo é apresentado, assim como uma breve revisão dos trabalhos existentes nessa área. Ainda nesta seção são introduzidas as métricas existentes para avaliar o desempenho de classificadores multirrótulo. Na 2.5 são feitas as considerações finais desse capítulo.

2.2 Aprendizado de Máquina

Aprendizado de Máquina (AM) é uma sub-área da Inteligência Artificial cujo objetivo é desenvolver métodos e técnicas computacionais que possibilitem a um computador a tomada de decisão a partir de experiência acumulada (Duda et al., 2000). Portanto, as técnicas de AM visam possibilitar a obtenção automática de conhecimento por meio de um conjunto de dados. Um dos principais focos da pesquisa em AM é produzir (induzir) modelos automaticamente a partir de dados, como por exemplo regras e padrões. O processo de indução consiste em realizar inferências gerais a partir de um sub-conjunto particular de dados. Esse processo de aquisição de conhecimento é denominado de Aprendizado Indutivo (Costa, 2008).

Na hierarquia do aprendizado proposta em (Monard e Baranauskas, 2003), o aprendizado indutivo pode ser dividido em duas categorias: o aprendizado supervisionado e o aprendizado não-supervisionado. Por sua vez, o aprendizado supervisionado pode ser dividido em classificação e regressão. O aprendizado supervisionado compreende os algoritmos de indução que realizam inferências a partir de dados rotulados. Quando os rótulos são discretos o problema é denominado de classificação e quando são contínuos de regressão. Redes Neurais Artificiais, SVM's e algoritmos baseados em Árvores de Decisão são exemplos de técnicas de aprendizado supervisionado.

No aprendizado não-supervisionado, são disponibilizados para o algoritmo apenas os exemplos de entrada, não existindo informação sobre a saída esperada. Uma das principais sub-áreas de aprendizado não-supervisionado é agrupamento de dados. Algoritmos de agrupamento de dados têm como objetivo encontrar padrões entre os exemplos por meio da formação de agrupamentos. Um exemplo é o algoritmo *k-means* (Duda et al., 2000).

Existem ainda outras categorias de aprendizado, entre elas aprendizado semi-supervisionado e aprendizado por reforço. No aprendizado semi-supervisionado, gera-se um indutor a partir da combinação de exemplos rotulados e não rotulados. Já no aprendizado por reforço, o algoritmo desenvolve uma política de como se comportar a partir de respostas às suas ações (Smith e Goldberg, 1992). O aprendizado por reforço será discutido com mais detalhes no Capítulo 3.

2.3 Classificação de Dados

A quantidade de dados armazenados em grandes bases de dados cresce continuamente, e, intuitivamente, essa enorme quantidade de dados deve guardar informação valiosa sobre o contexto do qual os dados se originam. Entretanto, o número de analistas de dados cresce em uma quantidade muito menor que o crescimento da quantidade de dados, o que dificulta o descobrimento de padrões e características importantes por esses analistas. Existe, portanto, uma clara necessidade em se desenvolver métodos (semi) automáticos para extrair o conhecimento contido nos dados que possam auxiliar o processo de análise.

Desta necessidade, surgiu uma área denominada Mineração de Dados, que faz parte de uma área maior, denominada Descoberta de Conhecimento, que utiliza técnicas da Estatística e de AM para tentar extrair conhecimento em alto nível das bases de dados (Freitas, 2002). Algumas das principais tarefas em Mineração de Dados são classificação, agrupamento, descoberta de regras de associação e regressão. Essas e outras tarefas em Mineração de Dados são descritas em (Freitas, 2002). Este trabalho está focado na tarefa de classificação.

Classificação é, provavelmente, a tarefa mais estudada em Mineração de Dados, tendo recebido a atenção das comunidades de AM e Estatística por várias décadas. Um classificador faz um mapeamento do espaço de padrões, denotado por R^m, em que m é o número de atributos de entrada, para um espaço de classes, denotado C. Algumas técnicas de AM podem ser utilizadas para induzir um classificador a partir de um conjunto de treinamento composto de n pares ($\mathbf{x_i}$,

y_i), onde $\mathbf{x_i} \in R^m$ e $y_i \in C$ é a classe do padrão $\mathbf{x_i}$. Esse classificador deve ser capaz de inferir corretamente a classe de novos padrões de R^m, não presentes no conjunto de treinamento. A essa capacidade de inferência é dado o nome de generalização.

2.4 Classificação com classes não mutuamente exclusivas

Considerando o número de rótulos de classe que podem ser associados a cada exemplo, os problemas de classificação podem ser classificados como único-rótulo ou multirrótulo. A abordagem mais utilizada em trabalhos de classificação de dados é a único-rótulo (do inglês, *single-label*), ou seja, problemas de classificação em que é associado apenas um rótulo de classe a cada exemplo.

Entretanto, existe uma grande quantidade de problemas relevantes em que cada exemplo pode ser associado a mais de uma classe simultaneamente. Esse grupo de problemas, em que as classes não são disjuntas, é conhecido como classificação multirrótulo (do inglês, *multi-label*) (Tsoumakas e Katakis, 2007). Por exemplo, um filme pode ser classificado como ação e aventura, um artigo no jornal pode ser classificado como pertencente à seção de música e cultura, além de outros. A motivação inicial para as pesquisas na área de classificação multirrótulo surgiu com a dificuldade causada por ambigüidades em problemas de categorização de textos (Schapire e Singer, 2000). A área de classificação de textos é até hoje a principal área de aplicação de técnicas de classificação multirrótulo. Apesar disso, a classificação multirrótulo também é importante em outras áreas, como reconhecimento de padrões e bioinformática. Em (Boutell et al., 2004), um classificador multirrótulo é utilizado para classificação semântica de cenas. Em (Clare e King, 2001), o classificador C4.5, baseado em árvores de decisão, foi adaptado para lidar com múltiplos rótulos de classe e aplicado a problemas de análise de expressão de genes. Mais recentemente, (Trohidis et al., 2008) realizaram uma comparação de diversas técnicas de classificação multirrótulo aplicadas a classificação de música em emoções.

De acordo com (Elisseeff e Weston, 2001), as classificações binária e multi-classes podem ser vistas como um caso especial do problema de classificação multirrótulo, em que o número de rótulos associado a cada exemplo é igual a 1.

Vários métodos foram propostos na literatura para lidar com o problema de classificação multirrótulo. Esses métodos podem ser divididos em independentes e dependentes de algoritmo. A abordagem independente combina classificadores único-rótulo para resolver o problema multirrótulo, enquanto que a abordagem dependente de algoritmo modifica internamente os classificadores único-rótulo de modo que esses possam ser usados para resolver problemas multirrótulo.

2.4.1 Abordagem Independente de Algoritmo

A abordagem para classificação multirrótulo independente de algoritmo pode ser vista como uma etapa de pré-processamento, podendo, a princípio, ser utilizada em conjunto com qualquer algoritmo de aprendizado. Essa abordagem geralmente lida com o problema de classificação multirrótulo transformando-o em um conjunto de problemas único-rótulo. Essa transformação pode ser realizada basendo-se nos rótulos de classe ou nos exemplos.

Transformação baseada em rótulos de classe

Nesta abordagem, conhecida como *Binary Relevance*, N classificadores, em que N é o número de classes do problema, são utilizados. Cada classificador é associado à uma das classes e treinado para resolver um problema de classificação binária. Os exemplos que possuam a classe associada ao classificador são considerados positivos e os demais exemplos negativos. Após todos os classificadores terem sido treinados, quando um exemplo ainda não visto é apresentado aos classificadores, todos aqueles que produzirem uma classe positiva terão sua classe associada ao novo exemplo (Aiolli et al., 2003). Um exemplo pode ser observado na Figura 2.1.

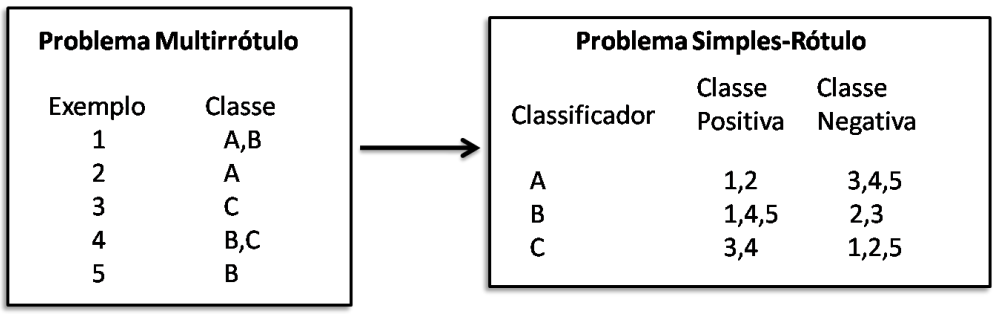

Figura 2.1: Exemplo de transformação baseada nos rótulos de classe

Essa abordagem considera que os rótulos de cada exemplo são independentes entre si, o que não é sempre verdade. Por ignorar possíveis relações entre os rótulos de classe, essa abordagem pode levar à uma pouca capacidade de generalização. Além disso, o modelo de classificação gerado é de baixa legibilidade, uma vez que é gerada uma regra em separado para cada um dos rótulos associado a um exemplo de teste.

Transformação baseada em exemplos

Na transformação baseadas em exemplos, o problema multirrótulo é convertido em um, ou mais, problemas único-rótulo, de acordo com o conjunto de rótulos associados à cada exemplo. Com essa redefinição, pode-se produzir mais de um problema de classificação único-rótulo, sendo que esse pode ser tanto um problema de classificação binária quanto multi-classes. As estratégias encontradas na literatura para transformação baseada em exemplos podem ser divididas em três grupos:

- Eliminação de exemplos multirrótulo;
- Criação de novos rótulos simples a partir de multirrótulo;
- Conversão de exemplos multirrótulo em exemplos único-rótulo.

A eliminação de exemplos multirrótulo é a mais simples, porém a menos eficaz, estratégia de transformação baseada em exemplos. Nessa estratégia, os exemplos que possuem mais de um rótulo de classe são simplesmente retirados do conjunto de dados, de maneira que restam apenas exemplos único-rótulo, como demonstrado na Figura 2.2. Assim, o problema de classificação multirrótulo deixa de existir. Entretanto, essa eliminação dos exemplos multirrótulo transforma o problema de classificação original em um outro problema, consideravelmente mais simples, e, muito possivelmente, não tão relevante quanto o original.

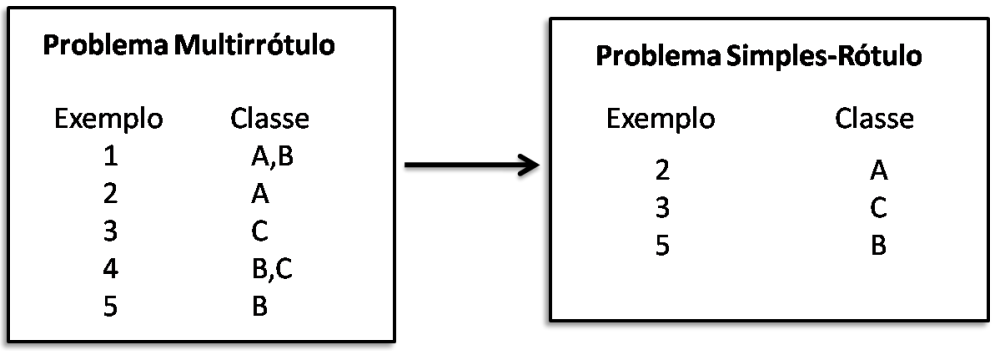

Figura 2.2: Exemplo de transformação por eliminação de exemplos multirrótulo

Na segunda estratégia, criação de novos rótulos, cada possível combinação de mais de uma classe é convertida em um novo rótulo ou classe. O problema dessa estratégia é que essa combinação das classes originais pode fazer com que o número de rótulos de classe aumente exponencialmente, resultando em classes com um pequeno número de exemplos. Esse tipo de transformação também é conhecida como *Label Power-Set*. Um exemplo da estratégia de criação de novos rótulos pode ser observado na Figura 2.3.

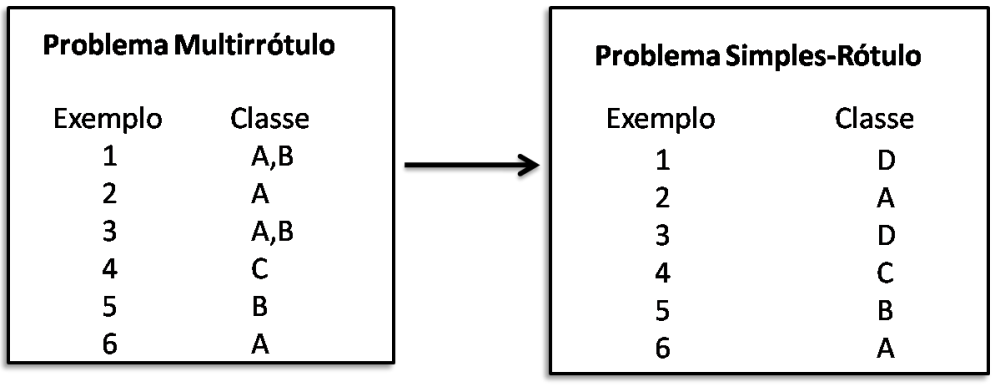

Figura 2.3: Exemplo de transformação criando novos rótulos

A terceira estratégia, conversão de exemplos multirrótulo em exemplos único-rótulo, pode ser realizada de duas maneiras distintas. A primeira delas consiste em transformar exemplos multi-rótulo em único-rótulo escolhendo um dos rótulos associados ao exemplo em questão e simplesmente eliminando os demais, como pode ser observado na Figura 2.4. A seleção do rótulo pode ser feita tanto de uma maneira determinística quanto aleatória. O principal problema dessa estratégia é que ela simplifica em excesso o problema ao escolher apenas uma entre as várias classes possíveis para um exemplo.

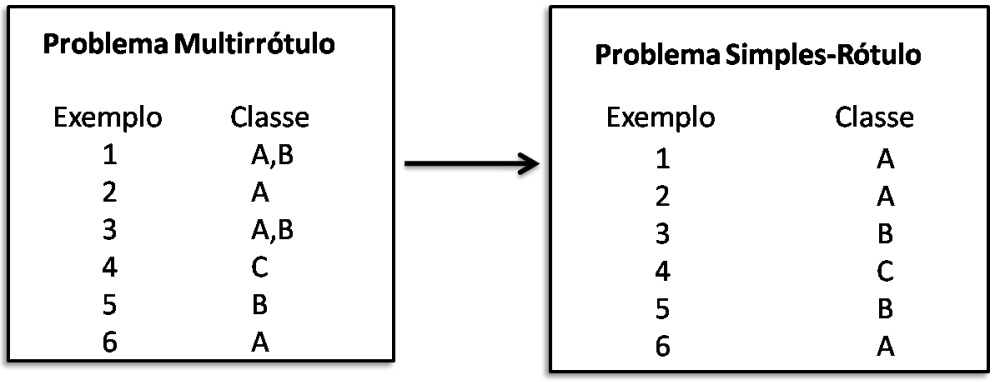

Figura 2.4: Exemplo de transformação por meio de eliminação de rótulos

A segunda maneira de se converter exemplos multirrótulo, por sua vez, divide o problema multirrótulo com N classes e M exemplos de entrada em K conjuntos de problemas único-rótulo. O valor de K varia de 1, quando nenhum exemplo possui mais de um rótulo, à $(N-1)^M$ se todos os exemplos possuem $N-1$ rótulos. Duas alternativas podem ser aplicadas para realizar essa decomposição.

A primeira delas, chamada método aditivo, considera que, para cada exemplo, cada um dos possíveis rótulos será a classe positiva em sequência. Um classificador é então treinado para discriminar cada classe das demais. Por exemplo, se os rótulos A, B e C aparecem nos exemplos multi-rótulo, quando o classificador para a classe A for treinado, todos os exemplos multirrótulo que possuem o rótulo A se tornam exemplos único-rótulo para a classe A. O mesmo acontece para os outros rótulos. Esse método foi proposto em (Shen et al., 2003) e também é conhecido como *cross-training*. Um exemplo pode ser observado na Figura 2.5.

A segunda alternativa para se decompor o problema multirrótulo, denominada método multiplicativo, é bastante similar à abordagem *one-against-one* (Hsu e Lin, 2002) utilizada para dividir problemas multi-classes em um conjunto de problemas de classificação binários. Nessa alternativa, realiza-se uma combinação de todos os possíveis classificadores. O número de classificadores é dado pelo produto do número de rótulos para cada exemplo. Esse método é claramente não escalável, uma vez que o número de classificadores cresce exponencialmente com o número de rótulos nos exemplos. A Figura 2.6 apresenta um exemplo.

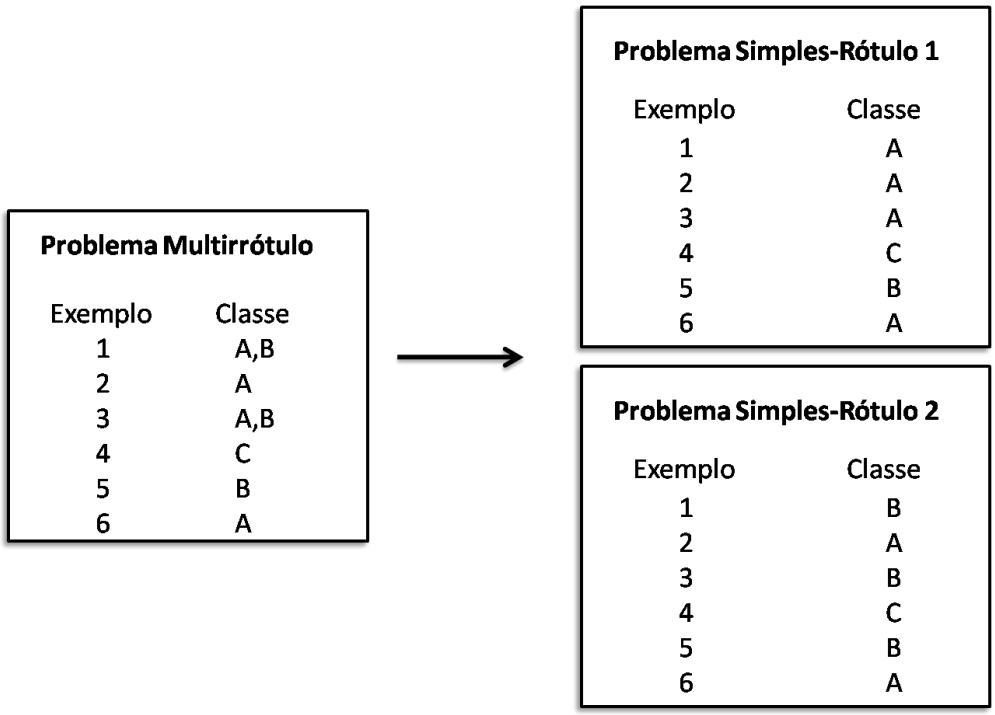

Figura 2.5: Exemplo de transformação usando o método aditivo

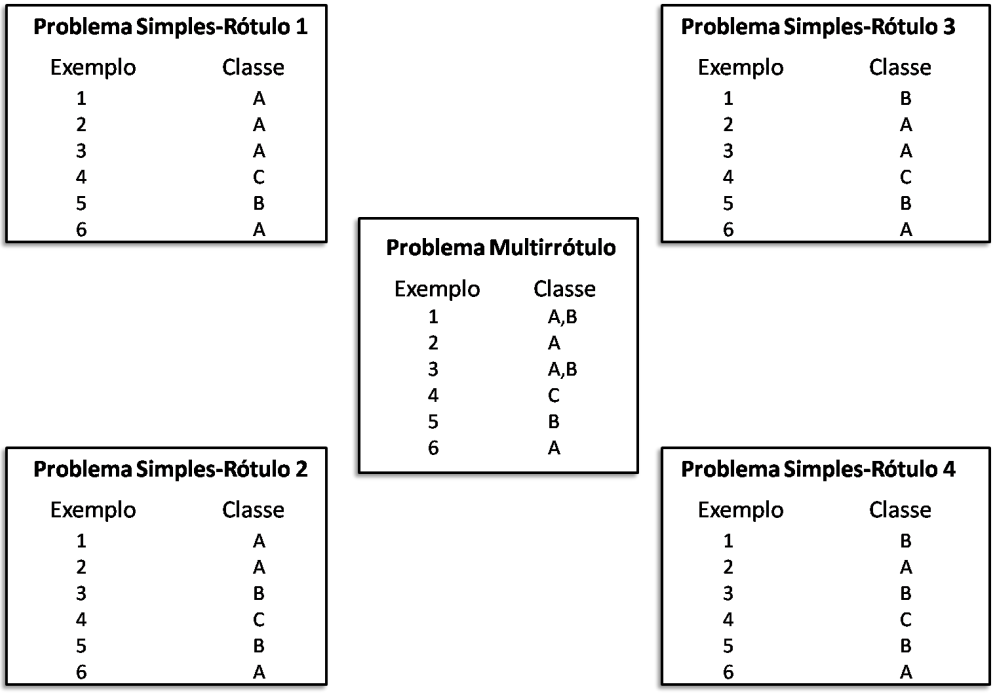

Figura 2.6: Exemplo de transformação usando o método multiplicativo

2.4.2 Abordagem Dependente de Algoritmo

Os métodos que seguem essa abordagem foram propostos para algoritmos de AM específicos.

No trabalho de (Karalic e Pirnat, 1991), os autores investigaram a utilização de Árvores de Decisão (AD) para problemas multirrótulo. Eles propuseram um método baseado em AD que

divide o problema multirrótulo em N problemas único-rótulo, onde N é o número de classes do problema. Em um outro trabalho utilizando AD (Clare e King, 2001), os autores modificaram o algoritmo C4.5 para classificar genes de acordo com sua função. Os autores modificaram a fórmula para cálculo de entropia, utilizada para definir os atributos associados aos nós da árvore no algoritmo C4.5, de maneira a permitir classificação multirrótulo. A Equação 2.1 apresenta a fórmula modificada.

$$Entropia(S) = \sum_{i=1}^{N} (p(c_i)logp(c_i) + q(c_i)logq(c_i)) \tag{2.1}$$

onde $p(c_i) =$ frequência relativa da classe c_i e $q(c_i) = 1 - p(c_i)$. No mesmo trabalho, os autores modificaram o algoritmo para permitir múltiplos rótulos nas folhas da árvore.

Duas extensões do algoritmo AdaBoost (Freund e Schapire, 1997) para classificação multirrótulo são os algoritmos Adaboost.MH e Adaboost.MR (Schapire e Singer, 2000). Ambos algoritmos aplicam AdaBoost em classificadores fracos da forma $H : X \times L \rightarrow R$, onde X pertence ao espaço de padrões, L é o conjunto de classes e R pode assumir os valores 1 ou -1. No AdaBoost.MH se o sinal de saída dos classificadores fracos for positivo para um novo exemplo $\mathbf{x}$ e um rótulo de classe l, então considera-se que esse exemplo deve ser rotulado com l. Por outro lado, se o sinal for negativo, o exemplo não recebe o rótulo l. No AdaBoost.MR, o sinal de saída dos classificadores fracos é utilizado para construir um ranking de todos os rótulos em L. Embora esses dois algoritmos sejam adaptações de um algoritmo específico, pode-se dizer que, em sua essência, eles produzem uma transformação do problema. Cada exemplo $(\mathbf{x}, Y)$ é decomposto em $|L|$ exemplos da forma $(\mathbf{x}, l, Y[l])$ para todo l em L, onde $Y[l] = 1$ se $l \in L$ e $Y[l] = -1$ caso contrário (Tsoumakas e Katakis, 2007).

Existem também trabalhos em classificação multirrótulo que utilizam SVMs. Em (Elisseeff e Weston, 2001), é proposto um método para classificação multirrótulo baseado em SVMs. Em (Aiolli et al., 2003), os autores propõem um método em que o problema multirrótulo é dividido entre várias SVMs em paralelo, combinando todos os classificadores em um único procedimento de otimização, por meio do compartilhamento da matriz *kernel* entre os diferentes classificadores.

ML-KNN (Zhang e Zhou, 2005, 2007) é uma adaptação do algoritmo KNN para dados multirrótulo. Na verdade, ML-KNN segue o princípio de transformação do problema baseada em rótulos de classes, explicado na Seção 2.4.1, utilizando o algoritmo K-NN independentemente para cada rótulo l. Para cada novo exemplo, identificam-se os seus k vizinhos mais próximos. Em seguida, para determinar o conjunto de rótulos desse novo exemplo, utiliza-se o princípio *Maximum a posteriori* (MAP) de acordo com os conjuntos de rótulos dos vizinhos. O que diferencia esse método da aplicação do K-NN ao problema transformado é a utilização de probabilidades *a priori*. Os autores conduziram experimentos em um conjunto de dados de bioinformática e demonstraram que o algoritmo proposto alcançou resultados comparáveis aos de outras técnicas de aprendizado multirrótulo.

Chan e Freitas (Chan e Freitas, 2006), utilizam um algoritmo de colônia de formigas para realizar classificação multirrótulo, chamado MuLAM, aplicado à problemas de bioinformática. O algoritmo MuLAM é uma adaptação do algoritmo Ant-Miner (Parpinelli et al., 2002) para permitir classificação multirrótulo. Os autores compararam o desempenho do algoritmo desenvolvido com o de algoritmos para classificação único-rótulo aplicados a problemas multirrótulo. O algoritmo proposto alcançou resultados melhores que a maioria dos algoritmos utilizados na comparação.

2.4.3 Questões referentes a conjuntos de dados multirrótulo

O número e a distribuição dos rótulos para cada exemplo pode variar bastante em diferentes conjuntos de dados. Por exemplo, em algumas aplicações, o número de rótulos de cada exemplo é pequeno quando comparado ao conjunto de todos os rótulos possíveis, enquanto que em outras aplicações o inverso ocorre. Além disso, existem casos em que os exemplos tem números similares ou muito distintos de rótulos. Isso pode trazer influência para o desempenho de diferentes métodos para classificação multirrótulo. A seguir são apresentados os conceitos de Cardinalidade e Densidade de rótulo (Tsoumakas e Katakis, 2007). Seja D um conjunto de dados multirrótulo que possui N exemplos $(\mathbf{x_i}, Y_i), i = 1..N$. Seja $|L|$ o número de rótulos possíveis.

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

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

Densidade de Rótulo: é o número médio de rótulos nos exemplos em D dividido por $|L|$.

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

2.4.4 Métricas de Avaliação

Diferente dos problemas único-rótulo, em que a classificação de um exemplo pode ser correta ou incorreta, num problema multirrótulo essa classificação pode ser parcialmente correta, ou parcialmente incorreta. Esses seriam os casos em que o classificador produz uma das classes corretas, porém não associa algum rótulo que deveria ter sido associado ao exemplo, ou associa um rótulo incorreto.

Algumas medidas foram propostas para avaliar a acurácia preditiva de classificadores multirrótulo (Shen et al., 2003; Zhang e Zhou, 2005; Karalic e Pirnat, 1991). O critério de avaliação pode tanto ser baseado na classificação realizada, utilizando para isso os rótulos produzidos pelo classificador para um dado exemplo, quanto em uma função de ranking, que utiliza a posição em um ranking associado à cada rótulo pelo classificador, para cada exemplo de entrada.

(Schapire e Singer, 2000) utilizam uma medida conhecida como *HammingLoss*. Seja D o conjunto de pares ($\mathbf{x_i}$, Y_i) a serem classificados, N o número de exemplos em D, L o conjunto de possíveis rótulos, H um classificador multirrótulo e Z_i o conjunto de rótulos predito por H para o exemplo $\mathbf{x_i}$. A medida é então definida como:

$$\text{HammingLoss}(H, D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \Delta Z_i|}{|L|}$$

onde Δ corresponde a diferença simétrica entre dois conjuntos e é equivalente à operação XOR da lógica Booleana. Outro trabalho (Godbole e Sarawagi, 2004) utiliza as seguintes métricas de avaliação:

$$\text{Acurácia}(H, D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \bigcap Z_i|}{|Y_i \bigcup Z_i|}$$

$$\text{Precisão}(H, D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \bigcap Z_i|}{|Z_i|}$$

$$\text{Revocação}(H, D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \bigcap Z_i|}{|Y_i|}$$

Algumas medidas de avaliação baseadas em rankings, como *One-Error* e Cobertura, são utilizadas em (Zhang e Zhou, 2007). Para as medidas baseadas em rankings, o classificador multirrótulo produz como saída uma função real $f(\mathbf{x_i}, Y_i)$, de modo que dados um exemplo $\mathbf{x_i}$ e seu conjunto de rótulos Y_i, $f(\mathbf{x_i}, y_1) > f(\mathbf{x_i}, y_2)$ para todo $y_1 \in Y_i$ e $y_2 \notin Y_i$. A função f pode ser transformada em uma função de ranking $rank_f(\cdot, \cdot)$ de modo que se $f(\mathbf{x_i}, y_1) > f(\mathbf{x_i}, y_2)$ então $rank_f(\mathbf{x_i}, y_1) < rank_f(\mathbf{x_i}, y_2)$. Assim, para um conjunto de teste $S = \{(\mathbf{x_1}, Y_1), ..., (\mathbf{x_p}, Y_p)\}$ a medida *One-Error* é definida como:

$$One - Error_S(f) = \frac{1}{p}\sum_{i=1}^{p}[[[\arg\max f(\mathbf{x_i}, y)] \notin Y_i]]$$

onde y pertence ao conjunto de rótulos possíveis. O termo $[\arg\max f(\mathbf{x_i}, y)]$ representa o valor máximo da função de $f()$ para o exemplo $\mathbf{x_i}$. Para todo π, $[[\pi]]$ é igual a 1 se π for verdadeiro e 0 caso contrário. Quanto menor o valor de *One-Error*, melhor o desempenho do classificador.

A medida de cobertura avalia, em média, quanto é necessário descer na lista de rótulos de modo a cobrir todos os rótulos de um exemplo. Quanto menor o valor de cobertura, melhor o desempenho do classificador. A medida é definida como:

$$Cobertura_S(f) = \frac{1}{p}\sum_{i=1}^{p}\max rank_f(\mathbf{x_i}, y) - 1$$

2.5 Considerações Finais

Neste capítulo foi abordado o problema de classificação multirrótulo. Para isso, uma introdução ao problema de classificação com classes disjuntas foi apresentada, visando facilitar a compreensão do problema multirrótulo, mais abrangente. Foram revisadas as principais formas de se abordar um problema de classificação multirrótulo, passando pelas abordagens que transformam o problema em um problema de classifição único-rótulo e são independentes de algoritmo, até as soluções propostas para a modificação de algoritmos de classificação existentes.

Por fim, foram discutidas algumas das métricas de avaliação para problemas multirrótulo existentes na literatura.

CAPÍTULO

3

Sistemas Classificadores Evolutivos

3.1 Considerações Iniciais

Neste capítulo são definidos os principais conceitos envolvendo os sistemas classificadores evolutivos, com ênfase para os chamados LCS, formando parte do embasamento teórico necessário para o entendimento da proposta de trabalho.

A Seção 3.2 contem uma breve introdução à Computação Evolutiva, indicando suas principais técnicas, sendo uma delas os AGs, e as características comuns entre elas. Na Seção 3.3 são apresentados os principais conceitos sobre AGs, principal mecanismo de busca dos LCS. Na Seção 3.4, os aspectos básicos dos LCS são discutidos e um breve histórico sobre a evolução dessa técnica e os diversos algoritmos desenvolvidos é apresentado. Na Seção 3.5 são apresentadas as considerações finais sobre este capítulo.

3.2 Computação Evolutiva

Computação Evolutiva é um termo que engloba técnicas para resolução de problemas que se baseiam na Teoria da Seleção Natural e na Genética. As técnicas da Computação Evolutiva, também chamadas Algoritmos Evolutivos, podem ser divididas em três áreas principais, conhecidas como Estratégias de Evolução (EEs),AGs e Programação Genética (PG). Os Algoritmos Evolutivos geralmente apresentam os seguintes componentes (Jong, 2006):

- Indivíduo: uma representação computacional de uma possível solução para o problema investigado;

- População: um conjunto de indivíduos;
- Função de Aptidão: uma função que quantifica ou compara a qualidade de um indivíduo;
- Herança: mecanismo que faz com que as características dos indivíduos mais aptos sejam transmitidas para indivíduos da próxima geração;
- Ciclo de Nascimento e Morte: uma maneira de atualizar a população.

Esses sistemas diferem entre si pela forma como especificam o tamanho da população, a seleção de indivíduos, os mecanismos de reprodução e herança, a representação dos indivíduos, a função aptidão, entre outros.

3.3 Algoritmos Genéticos

A evolução pode ser entendida como um processo de otimização que não visa a perfeição, mas que é capaz de encontrar soluções em geral eficazes e precisas para problemas impostos por um ambiente a um organismo (Mayr, 1987) . A Teoria da Evolução moderna baseia-se na Teoria da Seleção Natural, proposta pelo naturalista e fisiologista inglês Charles Darwin em seu livro A Origem das Espécies de 1859 (Darwin, 1859), e em princípios de herança genética, cujos conceitos foram introduzidos pelos trabalhos de Gregor Mendel (Mendel, 1865). A Teoria da Seleção Natural de Darwin sugere que indivíduos de uma população que estejam mais adaptados ao ambiente em que vivem têm maiores chances de sobreviver e gerar descendentes para as futuras populações.

Os AGs foram popularizados por John Holland em seu livro *Adaptation in Natural and Artificial Systems* (Holland, 1975), publicado na década de 70 e ainda hoje considerado referência básica ao tema. Aproximadamente na mesma época, EEs, uma técnica muito parecida com AGs, foram propostas. Entretanto, essa técnica não será discutida neste trabalho.

Goldberg, em (Goldberg, 1989) provê uma introdução aos AGs, e em seu mais recente livro (Goldberg, 2002) provê uma visão mais detalhada sobre o tema levando à construção de AGs competentes - AGs que resolvem problemas complexos de maneira rápida, precisa e confiável.

Esta seção apresenta uma breve revisão das principais características dos AGs. Para uma introdução ao tema, (Holland, 1975; Goldberg, 1989; Michalewicz, 1994) são recomendados. Para uma análise mais aprofundada sobre AGs, recomenda-se a leitura de (Goldberg, 2002; Back, 1996).

3.3.1 Algoritmo Genético Básico

Os primeiros conceitos utilizados hoje no que denomina-se AG foram introduzidos por John Holland (Holland, 1975). Na visão de Holland, a característica principal de um sistema adaptativo, natural e robusto era a utilização bem sucedida de competição e inovação. Essa característica

provinha aos indivíduos a habilidade de responder de maneira dinâmica a mudanças no ambiente e a eventos não antecipados.

Um AG evolui uma população (ou conjunto) de indivíduos, chamados cromossomos, que são definidos por um genótipo específico, ou seja, uma seqüência de genes. A decodificação do genótipo resulta no fenótipo do indivíduo, que especifica o seu significado. Por exemplo, ao tentar otimizar os ingredientes em uma receita, pode-se codificar a ausência (0) ou presença (1) de determinado ingrediente. O indivíduo decodificado, isto é, o fenótipo, seria os ingredientes colocados juntos de fato (Butz, 2004). A seguir é apresentado um AG típico.

Os AGs básicos podem iniciar seu processo evolutivo a partir de uma população inicial aleatória de indivíduos ou a partir de soluções conhecidamente promissoras. De qualquer forma, a idéia principal é que o AG irá guiar sua busca a partir de uma população de indivíduos, não um único indivíduo (Goldberg, 2002). Os indivíduos representam possíveis soluções codificadas para o problema que se deseja solucionar. A qualidade de cada indivíduo, seu *fitness*, é calculada por uma função aptidão, que geralmente produz um resultado escalar. Assim, se estamos lidando com um problema de maximização, quanto maior o valor do *fitness* de um indivíduo, melhor a sua qualidade. O AG irá guiar o processo evolutivo em busca de indivíduos com índices de aptidão cada vez maiores.

A partir de uma população de indivíduos avaliados, operadores genéticos de seleção e recombinação podem ser aplicados para gerar uma nova população, que espera-se conter melhores soluções para o problema. O processo de seleção representa o mecanismo de sobrevivência do mais apto, e pode ser realizado em uma grande variedade de formas, sendo as mais conhecidas a seleção por roleta, seleção por amostragem universal estocástica e seleção por torneio (Michalewicz, 1994; Lacerda et al., 1999; Goldberg e Deb, 1991). O ponto principal é, independentemente da maneira como a seleção é realizada, o importante é dar preferência às melhores soluções em detrimento das piores.

Uma vez selecionados os melhores indivíduos da população, esses sofrem a aplicação dos operadores genéticos de recombinação (*crossover*) e mutação. Os operadores de recombinação são baseados principalmente em conceitos de genética e recombinam características dos indivíduos pais gerando indivíduos filhos, na esperança de que os filhos gerados combinem as boas características estruturais dos pais. Novamente, essa idéia pode ser alcançada de várias formas. Para indivíduos codificados como cadeias binárias, *crossover* de um ponto, *crossover* muti-pontos e *crossover* uniforme estão entre as opções mais utilizadas (Lacerda et al., 1999). Para representação na forma de permutações, PBX, OBX, PMX são opções frequentes (Lacerda et al., 1999). Para mais informações a respeito de representações e operadores, consultar (Back et al., 2000; Baeck et al., 2000).

O operador de mutação altera arbitrariamente um ou mais componentes de um indivíduo, assegurando que a probabilidade de se chegar a qualquer ponto do espaço de busca nunca será zero. Além disso, a introdução de um operador de mutação permite contornar o problema de máximos (mínimos) locais, pois a direção da busca pode ser levemente alterada (Lacerda et al., 1999).

Em (Goldberg, 2002), Goldberg discute sobre como a estratégia utilizada pelos AGs pode ser vista como um modelo de inovação. Segundo o autor, os processos de seleção e de mutação, quando utilizados em conjunto, produzem uma melhoria contínua no processo de busca por soluções, como uma espécie de *hillclimbing* em que a mutação cria variações nas vizinhanças das soluções correntes e a seleção escolhe soluções cada vez melhores. Da mesma forma, seleção aliada a recombinação resulta em inovação, uma vez que o próprio ser humano muitas vezes utiliza a estratégia de combinação de idéias e conhecimentos prévios para criar algo novo.

3.3.2 Seleção e Substituição

Na natureza, os processos de seleção, reprodução e morte (exclusão) fazem parte do processo evolutivo como um todo (Mayr, 1987). Nos AGs, não haveria de ser diferente.

Guiados pelo valor do *fitness*, os métodos de seleção escolhem os melhores indivíduos da população para se reproduzirem e gerarem descendentes para as gerações seguintes. Existem muitas técnicas de seleção de indivíduos de uma população, cada qual com suas vantagens e desvantagens. Assim como a seleção é responsável por escolher os melhores indivíduos de uma população, os métodos de substituição são responsáveis por excluir indivíduos, trocando-os por novos para a geração seguinte.

Aqui, trataremos de duas técnicas de seleção, além do conceito de elitismo, e de algumas maneiras de substituir indivíduos da população.

A maioria das outras técnicas de seleção comumente utilizadas são comparáveis às técnicas aqui explicadas. Em (Goldberg e Deb, 1991; Goldberg e Sastry, 2001), uma comparação entre diferentes técnicas de seleção pode ser encontrada.

Seleção por Roleta

O método de seleção por roleta, também chamado de seleção proporcional, é talvez a forma mais simples e mais conhecida de se realizar seleção em AGs. Nesse método, os indivíduos são selecionados para reprodução proporcionalmente ao seu fitness. Dado um indivíduo com fitness f_i e a somatória dos fitness de todos os indivíduo na população (f), a probabilidade de seleção de um indivíduo S_i é dada por:

$$P_i = \frac{f_i}{f}$$

O método da roleta recebeu esse nome porque ele pode ser entendido como uma roleta que representa o *fitness* dos indivíduos na população. Para isso, a roleta é dividida em fatias, cada uma representando um indivíduo e possuindo tamanho proporcional ao *fitness* do indivíduo. Dessa forma, quanto mais apto o indivíduo, maior sua fatia na roleta. Para selecionar os indivíduos, gira-se a roleta n vezes, em que n é o tamanho da população. A cada girada, o indivíduo apontado

pela agulha é selecionado. Ao final, os indivíduos selecionados são inseridos em uma população intermediária (Lacerda et al., 1999; Goldberg, 1989). Um exemplo de seleção por roleta pode ser observado na Figura 3.1, onde S_i representa o i-ésimo indivíduo.

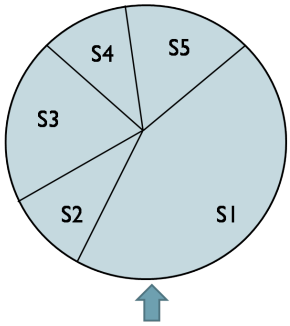

Figura 3.1: Método de seleção por roleta

O algoritmo da Roleta pode gerar problemas de convergência prematura, que é o caso em que o AG encontra um máximo (ou mínimo) local da função objetivo e não é capaz de redirecionar a busca para encontrar o máximo (ou mínimo) global. Isso ocorre por que não é possível controlar a pressão seletiva (Goldberg, 1989), ou seja, a predisposição do método em selecionar com frequência muito elevada os melhores indivíduos da população.

Seleção por Torneio

Ao contrário da seleção por roleta, a seleção por torneio não depende de um escalonamento de *fitness* (Lacerda et al., 1999). O método de seleção por Torneio seleciona m indivíduos da população aleatoriamente, com a mesma probabilidade. Entre esses m indivíduos, aquele que possuir a maior aptidão é selecionado para a população intermediária. O processo é então repetido até que a população intermediária seja completamente preenchida. Um exemplo de seleção por torneio pode ser observado na Figura 3.2, onde S_i, $i = 1, ..., 5$, representa o indivíduo i e $m = 3$.

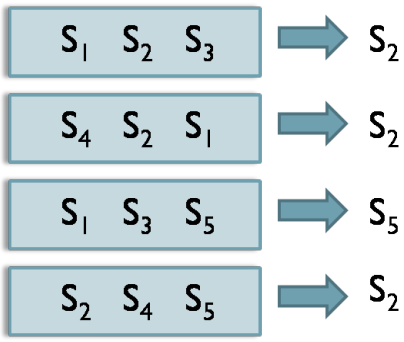

Figura 3.2: Método de seleção por torneio

O valor de m pode variar, mas geralmente utiliza-se m igual a 2 ou 3 quando a população é pequena (ordem de 10^1 indivíduos), mas esse valor pode ser muito maior quando a população do

algoritmo é realmente grande. Para uma análise da influência do valor de m no desempenho do AG e sua relação com o tamanho da população, ver (Goldberg, 2002). O valor de m pode ser usado para controlar a pressão seletiva, tornando o método do torneio mais apropriado em diversos problemas.

Elitismo

De Jong (Jong, 2006) propôs uma maneira de evitar que boas soluções encontradas pelo AG em uma dada geração sejam perdidas nas futuras gerações, devido a operações de recombinação e mutação. Essa estratégia é denominada elitismo e é muito comum nos AGs tradicionais. Elitismo consiste em incorporar na geração seguinte o melhor indivíduo da população atual, garantindo a permanência da melhor solução encontrada até então na próxima geração.

Técnicas de Nicho

Uma tendência natural nos AGs é que, com o passar das gerações, as soluções convirjam para uma mesma região, principalmente quando o problema possui vários máximos locais. Entretanto, em diversas aplicações, deseja-se obter não uma, mas diversas soluções que sejam igualmente boas. Para manter essa diversidade populacional, diminuindo a densidade de indivíduos em torno de uma única região, utilizam-se as chamadas Técnicas de Nicho. Essas técnicas são freqüentemente citadas na literatura como dois grandes grupos: *crowding* e *fitness sharing*.

As técnicas de *crowding* (Jong, 1975; Harik, 1994) consistem em substituir indivíduos similares na população por novos indivíduos. Na técnica da seleção por torneio restrita (Harik, 1994), um filho gerado é comparado com um subconjunto de indivíduos da população e substituirá aquele com o qual tenha maior similaridade, caso seu fitness seja mais alto.

As técnicas de *sharing* (Goldberg e Richardson, 1987), por sua vez, definem uma função de partilha de *fitness* que distribui o *fitness* de indivíduos de acordo com seu grau de similaridade. Em (Horn et al., 1994), os autores discutem a importância da utilização de técnicas de *fitness sharing* em LCS.

Métodos de Substituição

Uma vez que os AGs criam, a cada geração, novos indivíduos para fazer parte da nova população, é necessário estabelecer critérios que permitam decidir quais indivíduos serão incluídos na próxima geração e quais serão excluídos da população. A esses critérios dá-se o nome de métodos de substituição.

Na substituição geracional (Lacerda et al., 1999), a população inteira é substituída na geração seguinte, isto é, são gerados N filhos que substituirão N pais. Alternativamente, podem ser gerados N filhos e os N melhores indivíduos entre pais e filhos substituem a população atual. Outras variações dessa técnica também podem ser utilizadas (Lacerda et al., 1999).

Outra forma de realizar substituição de população é criar dois filhos que substituirão os dois piores indivíduos da população, ou alternativamente, os dois filhos podem substituir os pais, ou dois indivíduos aleatórios, ou ainda os dois indivíduos mais velhos da população. Essa forma de substituição é chamada de substituição de estado uniforme (Davis, 1991).

3.3.3 Recombinação e Mutação

Utilizar somente seleção não é suficiente para que um AG atinja seu objetivo, que é encontrar boas soluções para um dado problema. Para isso, mecanismos de recombinação de características e de mutação de indivíduos são necessários, pois esses realizam uma busca nas vizinhanças das boas soluções, encontrando soluções possivelmente melhores.

A mutação realiza uma busca ao redor de um indivíduo, alterando parte de sua estrutura de acordo com uma probabilidade de mutação. Esse é um importante mecanismo para manutenção da diversidade genética da população, pois garante que a probabilidade de se alcançar qualquer ponto do espaço de busca de soluções nunca será nula. Além disso, a introdução de um operador de mutação permite contornar o problema de máximos (mínimos) locais, pois a direção da busca pode ser levemente alterada. A probabilidade com que a mutação é aplicada é geralmente pequena, pois, caso contrário, a busca se torna essencialmente aleatória (Lacerda et al., 1999; Goldberg, 1989).

Figura 3.3: Exemplo de mutação em cromossomo binário

A recombinação, por sua vez, foi desenvolvida para combinar características de bons indivíduos, que foram previamente selecionados por algum método de seleção. A idéia da recombinação, também conhecida como *crossover*, é que dois indivíduos da população, chamados de pais, troquem informação entre eles e gerem novos indivíduos, chamados filhos, que serão inseridos na próxima geração. Assim, a recombinação realiza uma busca nas vizinhanças de dois indivíduos ao mesmo tempo, resultando em uma espécie de troca de conhecimento para solução do problema tratado (Goldberg, 2002).

Existem várias estratégias para recombinar indivíduos, freqüentemente denominadas operadores de recombinação ou *crossover*. A estratégia mais simples de *crossover* é o chamado *crossover* de um ponto. Nessa modalidade, escolhe-se um ponto de cruzamento no indivíduo e, a partir desse ponto, as informações genéticas dos pais são trocadas. Toda informação anterior a esse ponto no pai 1 é ligada a informação posterior a esse ponto no pai 2 e vice-versa, formando dois novos indivíduos.

Uma estratégia mais sofisticada, *crossover* multi-pontos, amplia a estratégia anterior, permitindo escolher não apenas um, mas vários pontos de cruzamento. Outra técnica bastante utilizada

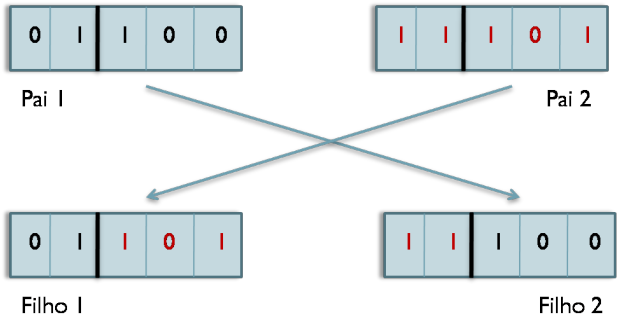

Figura 3.4: Crossover de um ponto

é o *crossover* uniforme. Nessa estratégia, ao invés de utilizar pontos de cruzamento, emprega-se uma máscara que indica quais genes de cada cromossomo pai cada filho herdará. Por exemplo, para uma máscara binária, um valor 1 na máscara faz com que o gene correspondente do pai 1 seja herdado pelo filho 1 e o filho 2 herde o gene correspondente do pai 2. Para um valor 0 na máscara, o inverso ocorre.

Para uma análise mais detalhada sobre operadores de recombinação sugere-se os trabalhos de (Holland, 1975; Goldberg, 1989; Michalewicz, 1994; Lacerda et al., 1999).

3.4 Learning Classifier Systems

Existem diversas técnicas de AM que são capazes de lidar com a tarefa de classificação, entre elas podemos citar as Redes Neurais Artificiais (RNAs) (Haykin, 1999), as Máquinas de Suporte Vetoriais (SVMs) (Smola et al., 1999; Lorena e Carvalho, 2003), as Árvores de Decisão (AD), entre outras.

Algumas dessas técnicas, por exemplo AD, realizam a tarefa de inferência de um classificador produzindo um conjunto de regras que pode ser analisado por um ser humano. Estas técnicas são conhecidas como Algoritmos de Indução de Regras de Classificação. Outras técnicas realizam esta tarefa de forma matemática ou estatística, dificultando a análise dos modelos gerados por um especialista humano. Estas técnicas são chamadas de "caixa-preta"e os principais exemplos são as RNAs e as SVMs. Produzir regras legíveis por seres humanos é muitas vezes mais importante do que a própria tarefa de classificação.

As pesquisas na área de algoritmos de indução de regras de classificação têm sido realizadas há mais de 30 anos, produzindo uma grande quantidade de algoritmos (Pappa e Freitas, 2006). Existem três estratégias comuns para se induzir regras de classificação a partir de um conjunto de dados (Pappa e Freitas, 2006):

- Dividir para conquistar, também conhecida como cobertura sequencial (do inglês, *sequential covering*);
- Geração de uma AD e extrair uma regra para cada nó folha da arvore.

- Utilização de algoritmos evolutivos, como os AGs e PG, para extrair regras dos dados.

Entre as três, a estratégia de cobertura sequencial é a mais explorada. A estratégia de cobertura sequencial aprende uma regra em um conjunto de treinamento, remove desse conjunto os exemplos cobertos pela regra e continua recursivamente aprendendo outra regra que cubra os exemplos restantes, até que todos, ou quase todos, os exemplos sejam cobertos. Essa é a abordagem mais comumente utilizada por algoritmos de indução de regras, como por exemplo, os algoritmos CN2 (Clark e Boswell, 1991) e RIPPER (Cohen, 1995).

O foco deste trabalho está na indução de regras de classificação utilizando algoritmos evolutivos, mais especificamente, AGs. Esses algoritmos têm sido denominados *Learning Classifier Systems* por alguns autores (Bernadó-Mansilla et al., 2005). As origens dos LCS remetem aos trabalhos de Holland sobre teoria de sistemas adaptativos (Holland, 1971). Este trabalho foi a base para a primeira implementação prática de um sistema de classificação, o Cognitive System Level One (CS1) (Holland e Reitman, 1978).

Além dos AGs, outras técnicas da Computação Evolutiva têm sido utilizadas para induzir regras de classificação. Em (Parpinelli et al., 2002; Smaldon e Freitas, 2006), um algoritmo de colônia de formigas é utilizado. Em (Holden e A., 2007), os autores propõem um algoritmo híbrido que mistura as técnicas de colônia de formigas com a de enxame de partículas. No trabalho de (Pappa e Freitas, 2006), programação genética é utilizada para indução de regras de classificação.

3.4.1 Otimização Multi-objetivos

A tarefa de classificação baseada em regras pode ser vista como uma tarefa inerentemente multi-objetivo. Uma vez que classificação é uma tarefa de mineração de dados, essa é, em sua essência, uma tarefa de extração de conhecimento a partir de um conjunto de dados que, vistos por si só, podem não fazer sentido algum para o usuário final. Levanta-se então a questão de que tipo de conhecimento deseja-se descobrir. Embora essa seja uma questão bastante subjetiva, pode-se mencionar três propriedades gerais que o conhecimento descoberto deve satisfazer (Freitas, 2002): precisão, compreensibilidade e inovação.

Em problemas de classificação, é importante que o modelo gerado seja capaz de prever novos exemplos, ou seja, que tenha boa capacidade de generalização. Dessa forma, deseja-se obter modelos que tenham taxa de acurácia elevada ao classificar exemplos não vistos anteriormente.

Muito importante em diversas aplicações é o fato do modelo ser compreensível ao usuário final, por exemplo, quando o modelo gerado será utilizado para auxiliar na tomada de decisões em grandes corporações. Compreensibilidade geralmente significa modelos com representação compacta, ou seja, quanto menor a quantidade de regras e quanto mais simples for cada regra, mais compreensível será o modelo.

A terceira propriedade, ser inovador (ou interessante), é a mais difícil de se definir e quantificar. Os métodos existentes para selecionar regras interessantes podem ser divididos em métodos objetivos e métodos subjetivos. Os métodos subjetivos são guiados pelo usuário e geralmente incluem

a representação de algum conhecimento prévio fornecido pelo usuário para que regras sejam consideradas interessantes (Liu et al., 1997). Por outro lado, os métodos objetivos são guiados pelos dados somente, de forma que a capacidade de uma regra ser interessante é medida por meio de comparações com outras regras, e não somente pela sua capacidade preditiva. Algumas medidas objetivas são discutidas em (Freitas, 1998, 1999).

Geralmente, todos esses objetivos a serem alcançados são opostos, e os esquemas de classificação tentam balanceá-los heuristicamente.

O aprendizado multi-objetivos também está presente nos LCS, que procuram otimizar os múltiplos objetivos por meio da utilização de um AG como mecanismo de busca. Em muitos casos, lida-se com os múltiplos objetivos implicitamente, por meio da interação entre componentes do modelo (Butz, 2004; Bernadó-Mansilla et al., 2005). Em outros casos, codifica-se os objetivos diretamente na função de aptidão do AG (Bernadó-Mansilla et al., 2005).

3.4.2 Um LCS Simples

LCS são sistemas de aprendizado adaptativo baseado em regras. Um LCS simples consiste em uma população de regras, um sistema de partilha de crédito, que geralmente aplica técnicas de aprendizado por reforço (porém nem sempre), e um mecanismo de evolução de regras, que geralmente utiliza um AG. A população de regras representa o conhecimento atual do LCS, ou seja, o conjunto de regras geradas para classificar dados de um certo domínio. O sistema de partilha de crédito é responsável por estimar a utilidade de uma regra, controlando a interação com o ambiente. Finalmente, o mecanismo evolutivo é responsável por, a partir da estimativa realizada, evoluir o conjunto de regras, gerando novos classificadores.

A interação entre o componente de aprendizado por reforço e o AG é a chave para o sucesso dos LCS, embora somente essa interação não assegure o sucesso. Para uma discussão mais detalhada sobre essa interação, ver (Butz, 2004).

Uma característica fundamental dos LCS é a possível emergência de conjuntos de regras que se comportam de maneira hierárquica, chamados de hierarquias default (do inglês, *default hierarchies*). Hierarquias default são conjuntos de regras onde regras mais gerais são responsáveis pela maior parte do comportamento do sistema, enquanto que camadas de regras mais específicas são responsáveis por dar a resposta quando as regras gerais são incorretas. Os LCS organizam hierarquias default favorecendo as regras mais específicas sobre as regras mais gerais em seus mecanismos de resolução de conflitos e atribuição de crédito. Tradicionalmente, para permitir que um LCS suporte a estrutura das hierarquias default, o mecanismo de resolução de conflitos dá preferência a regras com maior especificidade, ou seja, com menor número de # (*don't care*) em seu antecedente. A presença de um símbolo # no lugar de um atributo significa que o atributo pode possuir qualquer valor entre o conjunto de valores possíveis.

Aprendizado por Reforço

Considere um ambiente, que se comporta como uma caixa preta, que possui dois canais de saída e um canal de entrada (Smith e Goldberg, 1992). O primeiro canal de saída desse ambiente provê informação parcial sobre o seu estado interno. O canal de entrada permite que um sistema de controle externo altere o estado interno do ambiente, por meio de mensagens enviadas a este. A utilidade das mensagens de entrada é indicada pelo ambiente por meio de um sinal de recompensa enviado pelo segundo canal de saída. É importante ressaltar que, nessa interação entre ambiente e sistema de controle, os efeitos dos sinais de entrada aplicados ao ambiente não podem ser previstos de antemão e o ambiente não provê ao sistema de controle nenhuma informação sobre qual sinal de entrada o sistema deveria ter aplicado. Dessa maneira, o objetivo do sistema de controle aplicado a esse ambiente é otimizar o sinal de recompensa recebido do ambiente, ou seja, o sistema de controle deve ser capaz de adaptar sua estrutura interna a qualquer ambiente desconhecido. Essa situação caracteriza um problema de aprendizado por reforço (RL, do inglês *Reinforcement Learning*).

Um problema de classificação pode ser reescrito como um problema de aprendizado por reforço que provê recompensa baseada na precisão da classe escolhida, por exemplo fornecendo um valor de recompensa de 100 para a classe correta e 0 para classe incorreta. Essa redefinição de um problema de classificação em um problema de aprendizado por reforço é frequentemente referida como *single-step RL problems*. Por outro lado, nos *multi-step RL problems* existe um atraso no recebimento da recompensa, pois estados sucessivos dependem uns dos outros e da ação escolhida. Nesse caso, é necessária uma propagação de recompensa. Um exemplo de *multi-step RL problem* são os chamados *Markov Decision Processes* (MDP) (Butz, 2004).

Formação de Hierarquias Default

Hierarquias default são conjuntos de regras organizados em camadas em que regras mais gerais são responsáveis por uma grande parte das respostas do sistema, enquanto que regras mais específicas (regras excessões) são responsáveis por dar a resposta quando as regras gerais são incorretas. Embora as regras específicas sejam utilizadas quando as regras gerais são imperfeitas, regras específicas também podem ser imperfeitas. Como um exemplo, considere o seguinte conjunto de regras adaptado de (Smith e Goldberg, 1992), em que / delimita ações, ou seja, o conjunto de rótulos advocados pela regra:

Antecedente/Consequente
`1##/000`
`111/111`

Se uma regra específica casa com um exemplo de entrada então outras regras menos específicas na hierarquia também casarão. Para garantir que a regra correta seja escolhida, é preciso

garantir que esta tenha prioridade sobre as demais regras no mecanismo de resolução de conflitos. Tradicionalmente, para garantir que isto aconteça, é introduzido um fator de especificidade nos lances (do inglês, *bids*) de cada regra. O lance de uma regra é uma porção de sua força utilizado no mecanismo de resolução de conflitos. Entretanto, Smith propõe um outro método que é capaz de aumentar a capacidade de um LCS em balancear a resolução de conflitos entre regras gerais e específicas.

Em seu trabalho, Smith afirma que o método tradicional utilizado pelos LCS para favorecer a formação de hierarquias default é um modelo bastante simplificado da dinâmica de leilão. Nesse método, cada regra calcula seu lance como uma fração de sua própria força, sem levar em consideração os lances de seus competidores. Em leilões reais, cada competidor calcula o seu lance tendo como base os lances de seus competidores. O método proposto por Smith utiliza um leilão de necessidades, onde as regras utilizam um lance potencial. A regra que vence a competição não paga o valor de seu lance potencial, mas sim um lance atual que é igual ao lance potencial de seu competidor mais próximo.

Smith também propõe dois fatores separados para cada regra. O fator Π, chamado de estimativa de recompensa, é uma estimativa da recompensa esperada da regra. O segundo fator, Φ, chamado de fator de prioridade, indica a prioridade da regra no leilão. Dessa forma, define-se o lance potencial de uma regra como a multiplicação desses dois fatores.

Para realizar a atualização dos fatores, utiliza-se um procedimento linear para Π, enquanto que o Φ é atualizado segundo o leilão de necessidades. As equações para atualização dos valores dos fatores são:

$$\Pi_x^{i+1} = \Pi_x^i(1 - C) + CR \tag{3.1}$$

$$\Phi_x^{i+1} = \Phi_x^i - C\Pi_y^i + CR \tag{3.2}$$

onde Π_y^i é a estimativa de recompensa do competidor mais próximo da regra x no tempo i, C é a constante de *bid* e R é a recompensa recebida. Se não houver nenhum competidor o fator de prioridade não é atualizado.

Este método atinge boa separação entre regras gerais e específicas, pois mantendo fatores de prioridade separados, cada regra pode utilizar a estimativa de recompensa do seu competidor mais próximo para atualizar a sua própria prioridade no leilão. Entretanto, esse esquema pode resultar em um crescimento excessivo da prioridade de regras, não permitindo que novas regras inseridas na população sejam suficientemente avaliadas. Os problemas causados por não limitar as prioridades das regras podem ser resolvidos limitando os efeitos do leilão de necessidades, por meio da inclusão de uma margem M que limita a separação. Assim, a equação de atualização do fator de prioridade é alterada para:

$$\Phi_x^{i+1} = \Phi_x^i + CR - \begin{cases} C\Pi_y & if\ \Phi_x - \Phi_y < M \\ C\Pi_x & if\ \Phi_x - \Phi_y \geq M \end{cases} \tag{3.3}$$

3.4.3 Representando Regras de Classificação

Regras de classificação podem ser representadas de diversas maneiras. Nessa seção, algumas dessas maneiras serão apresentadas. Por motivo de simplicidade, as estratégias aqui discutidas seguem a abordagem Michigan, porém muitas dessas idéias podem ser adaptadas para a abordagem Pittsburgh. Seja uma regra de classificação formada como na Figura 3.5, onde $cond_1 \ \ldots \ cond_n$ são condições do tipo atributo-valor e C_i é a classe predita pela regra.

Figura 3.5: Definição de uma regra de classificação

Uma regra pode ser dividida em duas partes principais, o antecedente (a parte IF da regra) e o conseqüente (a parte THEN da regra), e existem técnicas específicas para codificar cada uma delas em um cromossomo.

Uma abordagem bastante simples para representar o antecedente de uma regra que possua atributos categóricos é utilizar uma codificação binária. Nessa codificação, se um atributo pode tomar r valores distintos, então codifica-se uma condição sobre esse atributo utilizando r bits. O i-ésimo valor ($i = 1, ..., r$) do domínio dos valores que o atributo pode assumir fará parte da condição se, e somente se, o i-ésimo bit possuir o valor 1. A codificação binária também pode ser utilizada para atributos que possuam valores contínuos, utilizando bits que representam o valor do atributo em notação binária (Freitas, 2002).

Outra abordagem para representar antecedentes de regras é utilizar uma representação de mais alto nível, que codifica diretamente as condições de uma regra no genoma de um indivíduo. Algumas estratégias que utilizam essa abordagem são descritas em (Flockhart e Radcliffe, 1995; Freitas, 2002).

Para uma descrição didática sobre representação de antecedente de regras consultar (Freitas, 2002).

De maneira geral, existem pelo menos três maneiras para se representar o conseqüente de uma regra de classificação (Freitas, 2002). A primeira estratégia codifica o conseqüente diretamente no genoma do indivíduo (Jong et al., 1993), tornando-o sujeito à evolução. A segunda estratégia associa todos os indivíduos da população à uma mesma classe, que não sofre modificações durante a execução do algoritmo. Assim, para gerar um conjunto de regras predizendo k classes, o algoritmo deve ser executado k vezes (Janikow, 1993).

Finalmente, a terceira estratégia assume que, assim que o antecedente da regra é formado, pode-se escolher a classe mais adequada para a regra. A classe escolhida pode ser, por exemplo,

aquela com o maior número de representantes entre o conjunto de exemplos que satisfaçam o antecedente da regra (Giordana e Neri, 1995). Segundo (Freitas, 2002), a terceira alternativa seria a melhor delas.

3.4.4 Abordagens Michigan e Pittsburgh

Os LCS podem ser divididos em duas grandes abordagens, dependendo de como as regras de classificação são codificadas na população de indivíduos do AG. Na abordagem Michigan, cada indivíduo (cromossomo) codifica uma única regra de classificação, enquanto que na abordagem Pittsburgh cada indivíduo representa um conjunto de regras de classificação.

A abordagem Michigan foi inspirada pelo algoritmo CS1 (Holland e Reitman, 1978), e assim como nesse algoritmo, os LCS que seguem essa abordagem aprendem incrementalmente pela interação com o ambiente. Nos LCS que seguem a abordagem Michigan tradicional, em geral, inicia-se a partir de uma população aleatória de regras que são avaliadas por meio de um esquema de aprendizado por reforço utilizando um algoritmo de atribuição de crédito (Butz, 2004; Bernadó-Mansilla et al., 2005). Um algoritmo de atribuição de crédito clássico é o bucket brigade (Holland e Reitman, 1978), em que a qualidade de uma regra é atribuída de acordo com a recompensa que a regra receberia do ambiente caso sua ação fosse selecionada (Bernadó-Mansilla et al., 2005). A tarefa do AG é descobrir novas regras promissoras e também assegurar a co-evoluçao de um conjunto diverso de regras. Para isso, ele utiliza a qualidade da regra, calculada anteriormente, como aptidão do indivíduo.

Como cada indivíduo na abordagem Michigan codifica uma única regra, os indivíduos tendem a ser mais simples e sintaticamente mais curtos, o que tende a reduzir o tempo necessário para se calcular a aptidão do indivíduo e simplificar os operadores genéticos (Freitas, 2002). Porém, uma vez que a função de aptidão avalia a qualidade de um único indivíduo por vez, não é fácil levar em consideração as interações entre regras e assim computar a qualidade do conjunto de regras gerado como um todo. Outro problema é que os AGs tradicionais tendem a convergir para um único indivíduo, o que é indesejado, visto que procura-se obter um conjunto de regras de classificação, e não apenas uma. Esquemas não geracionais de substituição da população e técnicas de nicho são utilizadas para manter a diversidade de regras na população (Goldberg, 1989; Bernadó-Mansilla et al., 2005).

O maior problema dos LCS que seguem a abordagem Michigan tradicional é conseguir alcançar uma boa taxa de generalização e, ao mesmo tempo, conseguir manter diferentes nichos na população. Existe um equilíbrio delicado, e difícil de manter, entre a cooperação necessária para se encontrar um conjunto de regras e a competição entre os indivíduos por um lugar na população (Bernadó-Mansilla et al., 2005). O algoritmo Boole (Wilson, 1986) tenta manter esse equilíbrio por meio da utilização de técnicas de compartilhamento de *fitness* e de um AG aplicado aos nichos definidos pelos conjuntos de ação (conjuntos de regras que possuem atributos similares e a mesma classe).

Um outro algoritmo, chamado Cogin (Greene e Smith, 1993), evita a competição entre as regras não permitindo que novas regras geradas pelo AG cubram os mesmos exemplos já cobertos por outras regras.

O algoritmo Boole deu origem a um outro algoritmo, chamado New-Boole (Bonelli et al., 1990). Em testes realizados com bases de dados médicos, os resultados obtidos pelo New-Boole são comparáveis ao algoritmo de classificação CN2 e as redes neurais artificiais utilizando o algoritmo backpropagation. Modificações no New-Boole para lidar com bases de dados epidemiológicas originaram o algoritmo EpiCS (Holmes, 1997). Os resultados obtidos por EpiCS também são comparáveis aos de outros algoritmos de classificação (Bernadó-Mansilla et al., 2005).

Wilson (Wilson, 1995) introduziu o conceito de classificadores baseados em acurácia. O XCS, algoritmo desenvolvido por Wilson, é atualmente um dos padrões mais seguidos pelos LCS com abordagem Michigan. O trabalho de Wilson enfatiza a importância da acurácia ao se avaliar o fitness de um indivíduo. Além disso, define-se o conceito de regras que maximizam a acurácia e a generalização ao mesmo tempo.

Outras abordagens baseadas em acurácia têm sido estudadas, como é o caso do UCS (Bernadó-Mansilla e Garrell, 2003). O UCS possui muitas características em comum com o XCS, entretanto o UCS foi projetado especificamente para problemas de aprendizado supervisionado.

Ao mesmo tempo em que os primeiros desenvolvimentos dos LCS baseados na abordagem Michigan surgiam, uma outra linha de LCS estava sendo pesquisada sob o nome de abordagem Pittsburgh (Bernadó-Mansilla et al., 2005). Essa abordagem teve origem do desenvolvimento do sistema de classificação LS-1 (Smith, 1983), que inspirou outro algoritmo chamado GABL (Jong e Spears, 1991). Na abordagem Pittsburgh, cada indivíduo codifica um conjunto de regras, portanto, uma solução completa para o problema de classificação. A aptidão de cada indivíduo é calculada como o desempenho do conjunto de regras como um todo, geralmente utilizando um esquema de aprendizado supervisionado.

No algoritmo GABL, os indivíduos são codificados como um conjunto de regras de tamanho fixo, em que cada indivíduo é uma string binária. Para calcular a aptidão de cada indivíduo, apenas a precisão de classificação é levada em consideração.

Uma vez que cada indivíduo representa um conjunto completo de regras, não há necessidade de cooperação entre os indivíduos e, portanto, técnicas de nicho não são necessárias. Entretanto, os indivíduos gerados são sintaticamente mais longos, o que implica em maior custo computacional para calcular a aptidão de cada indivíduo. Outra dificuldade é que se tem pouco controle ao nível de regras (Bernadó-Mansilla et al., 2005). Para tentar lidar com esse problema, versões posteriores do GABL introduziram operadores de adição de condições e deleção de condições no antecedente da regra, além de aprendizado incremental. O algoritmo GIL (Janikow, 1991) também utiliza um grande conjunto de operadores que agem em diferentes níveis.

A flexibilidade dos LCS que seguem a abordagem Pittsburgh foi sensivelmente aumentada com o uso de indivíduos de tamanho variado. Entretanto, essa flexibilidade causou o crescimento

excessivo do tamanho dos indivíduos, com a inclusão de regras inúteis nos indivíduos ou evolução de regras super-especializadas. Alguns trabalhos foram propostos na literatura para solução desse problema, como é o caso de (Llorà et al., 2003). Ainda sob essa perspectiva, os desenvolvimentos mais recentes incluem os algoritmos GALE (Llorà, 2002) e GAssist (Bacardit e Garrell, 2007).

Todos os algoritmos previamente mencionados, independente da abordagem ao qual pertencem, tentam alcançar implicitamente os múltiplos objetivos da tarefa de se gerar um conjunto eficiente de regras. Por outro lado, existem algoritmos que descrevem esses objetivos explicitamente, com o uso de algoritmos evolutivos multi-objetivos (MOEA, do inglês Multiobjective Evolutionary Algorithms) . De acordo com essa perspectiva, seguindo a abordagem Michigan tem-se, por exemplo, o algoritmo MOLeCS (Bernadó-Mansilla e Garrell, 2001). Os algoritmos MOLS-GA e MOLS-ES (Llorà et al., 2003) são exemplos de MOEA que seguem a abordagem Pittsburgh.

A seguir são apresentados os algoritmos XCS e GAssist que seguem, respectivamente, as abordagens Michigan e Pittsburgh.

XCS

O algoritmo XCS é um LCS pertencente à abordagem Michigan que foi proposto por Wilson (Wilson, 1995). Como os demais algoritmos pertencentes à abordagem Michigan, o XCS codifica cada indivíduo como uma única regra. A população de indivíduos é avaliada pela interação com o ambiente, por meio de um mecanismo de aprendizado por reforço. Os indivíduos evoluem utilizando um AG.

O ciclo de treinamento do XCS funciona da seguinte maneira (Bernadó-Mansilla et al., 2005). A cada iteração, um exemplo é apresentado ao XCS. Com esse exemplo, o XCS encontra um conjunto M formado por todas as regras da população cujos antecedentes satisfaçam o exemplo de entrada. Se nenhuma regra satisfizer o exemplo de entrada, aciona-se um operador de cobertura, que gera regras para cobrir o exemplo de entrada.

O algoritmo então seleciona uma das classes presentes no conjunto M.Todas as regras de M que possuam essa classe em seu conseqüente são colocadas em um novo conjunto A. A classe escolhida é enviada ao ambiente, que calcula o erro de predição das regras em A com relação ao exemplo de entrada, computa uma medida de acurácia e então atualiza a aptidão das regras de A. O AG então é acionado no conjunto A, selecionando dois pais com probabilidade proporcional à aptidão e aplicando os operadores de crossover e mutação. Antes de serem inseridos na população, os indivíduos filhos são comparados com os pais. Caso um dos pais tenha acurácia suficiente e seja mais geral que o filho, então o filho é descartado. Esse mecanismo é utilizado para favorecer a generalização das regras. Caso a população esteja completa ao se tentar incluir um filho, uma das regras é escolhida para ser excluída da população.

Na fase de teste, o XCS atribui ao exemplo que se deseja classificar a melhor classe entre aquelas sugeridas pelos indivíduos contidos em M (Bernadó-Mansilla et al., 2005).

Uma explicação mais detalhada de como o XCS alcança implicitamente os objetivos de acurácia e generalização pode ser encontrada em (Bernadó-Mansilla et al., 2005).

GAssist

Como mencionado na subseção 3.4.4 o GAssist é um dos algoritmos da abordagem Pittsburgh que permite a utilização de indivíduos de tamanho variável. Uma vez que o que se deseja obter ao final do processo é um conjunto de regras de classificação, a utilização de indivíduos que não tenham um tamanho fixo pré-determinado soa mais natural.

Entretanto, a evolução de indivíduos de tamanho variado causa um efeito conhecido como *bloat* (Tackett, 1994), que é o crescimento do tamanho (código) de um indivíduo sem um conseqüente aumento da sua aptidão. Existem duas possíveis razões para o *bloat*, segundo Langdon e Poli (Langdon e Poli, 1998): aptidão e proteção de código. A primeira razão, aptidão, refere-se ao fato de que a seleção não é capaz de distinguir entre indivíduos com a mesma aptidão, mas com tamanhos diferentes. A segunda razão atesta que código neutro que não possui influência na aptidão do indivíduo tende a ser protetor, ou seja, tende a reduzir a probablidade de que os operadores genéticos destruam código útil.

Segundo (Bernadó-Mansilla et al., 2005), nos LCS que seguem a abordagem Pittsburgh, o efeito *bloat* pode aparecer de duas maneiras: adição de regras inúteis ou evolução de regras muito especializadas. Uma das formas de se tentar reduzir esse problema é usar uma combinação linear entre acurácia e generalização de um indivíduo, em que a capacidade de generalização seja inversamente proporcional ao tamanho do indivíduo (Bernadó-Mansilla et al., 2005).

O algoritmo GAssist opera de forma parecida com um GA geracional, evoluindo indivíduos que representam uma solução completa para o problema. Nesse algoritmo, uma função de aptidão baseada no princípio do *Minimum Description Length* (MDL) (Rissanen, 1993) é utilizada. Além disso, o GAssist utiliza um mecanismo de deleção de regras denominadas introns, que são as regras que não foram ativadas por nenhum exemplo (Bernadó-Mansilla et al., 2005).

Em (Bacardit e Butz, 2004), é realizada uma comparação entre o GAssist e o XCS. Os resultados demonstraram que eles alcançaram resultados similares quanto à acurácia, e que o conjunto de regras evoluído pelo GAssist foi muito menor que o conjunto de regras evoluído pelo XCS.

3.4.5 Uma terceira abordagem

Embora Michigan e Pittsburgh sejam as abordagens que detiveram maior atenção dos pesquisadores da área de LCS, existe uma terceira abordagem, proposta por Jason Wilcox (Wilcox, 1995). Essa abordagem procura oferecer um balanço entre comportamento individual e coletivo, possibilitando uma melhor separação entre regras ótimas e sub-ótimas. A inspiração para o seu desenvolvimento vem da economia, mais especificamente das teorias de redução de custos em transações financeiras.

Fundamentos

A grande maioria dos avanços na área dos LCS diz respeito aos algoritmos pertencentes as abordagens Michigan e Pittsburgh. Enquanto Michigan evolui regras individuais codificadas como uma única string, Pittsburgh evolui strings que codificam um conjunto completo de regras. De certa maneira, tanto Michigan quanto Pittsburgh abordam o mesmo problema a partir de extremos opostos entre comportamento individual e coletivo. Nenhuma das duas abordagens é completamente satisfatória. Os classificadores Michigan convergem mais rapidamente, porém falham em obter boas soluções para problemas complexos. Por outro lado, classificadores Pittsburgh são capazes de solucionar problemas mais complexos, ao custo de tempo de processamento e de uso de memória mais altos.

Por meio de conceitos de reputação e de formação de organizações adequadas, Jason Wilcox propôs um sistema classificador organizacional (OCS, do inglês *Organizational Classifier System*) que ajusta automaticamente o balanço entre comportamento individual e coletivo. Para atingir esse objetivo e criar um mecanismo autônomo para formação de organizações de regras, Wilcox usou estratégias de redução de custos em transações financeiras. R.H. Coase explica que a formação de organizações se dá devido à redução dos custos de transações associados a troca de bens e serviços.

Wilcox exemplifica custos de transações por meio de transações entre duas pessoas em um mercado. A primeira pessoa, o dono de uma manufatura, é responsável pela produção e venda de um determinado produto. A segunda pessoa, o trabalhor, é aquele que constrói o produto com a matéria prima. Uma vez que o dono da manufatura estabelece um contrato com o trabalhor, diz-se que houve uma transação. Para o dono da manufatura, o custo associado à transação pode ser definido pelo esforço em encontrar um trabalhor, o esforço em negociar o preço pago pelo trabalho e ainda o risco de contratar um trabalhador ruim. Por outro lado, o trabalhor também coloca esforço em encontrar um empregador e negociar o preço pelo seu trabalho, assim como deve enfrentar o risco de o empregador não cumprir com o pagamento. Wilcox explica que, similarmente às duas pessoas no mercado, duas regras também trocam bens e serviços. Isso em geral ocorre em forma de troca de informação para atualização da medida de qualidade (do inglês, *strength*) das regras por meio do algoritmo *Bucket Brigade*.

Além disso, uma porção dos custos envolvidos está associada ao ganho de conhecimento sobre o mercado. Isso quer dizer que quanto mais informação se acumula sobre os possíveis agentes com quem negociar, menor o risco enfrentado.

Acumular informação sobre reputação e formar organizações são duas técnicas fundamentais para a redução dos custos associados às transações. Fazer parte de uma organização provê proteção contra vários níveis de custos de transações porque algumas tarefas são atacadas em conjunto e assim amortizam-se os custos entre todos os membros da organização. Da mesma maneira, prestar atenção à reputação é um fator importante ao decidir com quem negociar, uma vez que quanto melhor a reputação, menor o risco associado na transação.

Wilcox aplica ambos os conceitos em seu sistema de classificação com o objetivo de diminuir os custos relacionados e encontrar uma melhor separação entre regras ótimas e sub-ótimas.

Mecanismos do OCS

A idéia principal por trás do OCS é separar adequadamente as regras que levam a decisões ótimas daquelas que levam a decisões sub-ótimas. Para alcançar esse objetivo, o OCS tenta criar organizações adequadas de regras baseando-se no conceito de reputação. O OCS possui três componentes principais:

- Sistema de produção de regras;
- Mecanismos de resolução de conflitos e alocação de crédito;
- Componente de crescimento organizacional.

O sistema de produção de regras contém uma população de organizações que, por sua vez, contém regras. As organizações podem possuir tamanhos variados e interagem entre si. Para cada exemplo de entrada, o sistema de produção é responsável por encontrar o respectivo *match-set* (conjunto de regras que casam com o exemplo de entrada) e, a partir desse, escolher uma ação e enviá-la ao ambiente. O ambiente, por sua vez, pode ou não enviar ao sistema de produção um sinal de recompensa, que indica a adequabilidade da ação escolhida.

As interações de regras e organizações são determinadas utilizando-se um mecanismo de resolução de conflitos baseado em valores de reputação, enquanto que o mecanismo de alocação de crédito fica responsável pela atualização desses valores toda vez que uma regra é escolhida para enviar sua ação ao ambiente. O OCS utiliza quatro valores de reputação para escolher organizações e regras. Cada regra, e cada organização, carrega dois valores que representam, respectivamente, a medida de reputação a longo prazo (LT) e a curto prazo (ST).

A reputação de uma regra é determinada pelo seu sucesso em obter recompensas do ambiente quando sua ação é selecionada. A performance mais recente de uma regra é medida por sua reputação ST. A reputação ST para a i-ésimo regra a ser acionada é calculada pela equação 3.4.

$$V_{STC}^{i} = R + b \sum \left\{ V_{LTC}^{i+1} \right\} \tag{3.4}$$

onde R é a recompensa recebida pelo ambiente, b é uma constante e V_{LTC}^{i+1} é uma fração da soma dos valores de reputação LT de todos os classificadores que são acionados após lerem a mensagem interna postada pela última regra que foi escolhida. O segundo termo está associado ao algoritmo *Bucket Brigade*.

A reputação LT de um classificador é calculada similarmente ao cálculo do valor de força em um classificador Michigan. A fórmula para cálculo da reputação LT de classificadores é dada pela equação 3.5.

$$V_{LTC}^{i} = (1-b)V_{LTC}^{i} + b\sum\left\{V_{LTC}^{i+1}\right\} + R \tag{3.5}$$

Os valores de reputação de uma organização refletem indiretamente o sucesso de seus classificadores. Assim, a reputação ST de uma organização é calculada como:

$$V_{STO}^{t+1} = (1-T_O)V_{STO}^{t} + \sum_{c\in O} T_c V_{LTC}^{c} - F \tag{3.6}$$

onde T_O é uma taxa do sistema, T_c é uma taxa sobre cada classificador c que pertence a organização O e F é uma taxa paga quando a organização é selecionada para afetar o ambiente. Todas as taxas são constantes. A reputação LT de uma organização mede o desempenho da organização durante seu tempo de vida e é calculada pela equação 3.7.

$$V_{LTO}^{i} = \frac{S_{sucesso}}{S_{tentativas}} \tag{3.7}$$

onde $S_{sucesso}$ é o número de vezes que a organização recebeu um sinal de recompensa quando foi selecionada para afetar o ambiente e $S_{tentativas}$ é o número de vezes que a organização foi selecionada para afetar o ambiente.

O mecanismo de resolução de conflitos do OCS utiliza reputação LT para selecionar uma organização e reputação ST para selecionar um classificador. Os outros dois valores de reputação são utilizados pelo componente de crescimento organizacional.

O último componente a ser discutido é o componente de crescimento organizacional, responsável por controlar como as organizações variam de tamanho. Esse componente é aplicado a cada k iterações e para cada organização presente na população seleciona um operador para ser aplicado. O OCS possui dois operadores, denominados *Grow* e *Shrink*. O operador *Grow* é responsável por aumentar o tamanho de uma organização por meio da inclusão de classificadores. Esse operador seleciona um classificador na população que não esteja contido na organização atualmente selecionada. Calcula-se então o salário das duas organizações (a atualmente selecionada e a organização que contém o classificador selecionado pelo operador) dividindo a reputação ST de cada uma pelo respectivo número de classificadores que essas possuem. A organização que possuir o maior salário receberá o classificador selecionado. Por outro lado, o operador *Shrink* reduz o tamanho de uma organização excluindo classificadores com baixo desempenho. Esse operador compara a reputação LT de cada classificador incluído na organização selecionada com um valor fixo e próximo a zero. Aqueles classificadores cujos valores de reputação LT estiverem abaixo desse valor fixo são removidos da organização. Os classificadores removidos não são retirados da população, mas sim colocados cada qual em uma nova organização.

3.5 Considerações Finais

Neste capítulo foram abordados os mecanismos básicos de um AG típico, introduzindo as características essenciais dos AGs, como os processos de seleção, recombinação e substituição de população. Esses conceitos serviram como base para a introdução aos LCS, que utilizam AGs como mecanismo de busca na tarefa de indução de regras de classificação. Outro conceito apresentado foi o de Aprendizado por Reforço, por estar fortemente relacionado com a teoria dos LCS. Uma das características dos LCS, a possível formação das chamadas hierarquias default, foi discutida e uma revisão do método proposto por Smith para sua formação, utilizando leilão de necessidade e diferentes fatores de prioridade, foi apresentado. Foram também discutidas as duas abordagens mais utilizadas, Michigan e Pittsburgh, assim como os principais trabalhos desenvolvidos seguindo cada uma delas. Uma terceira abordagem, utilizada pelo classificador OCS, foi discutida, assim como os principais mecanismos do OCS.

CAPÍTULO

4

OCS para Classificação Multirrótulo

4.1 Considerações Iniciais

Alguns problemas de classificação requerem que haja uma diferenciação hierárquica entre as regras presentes no conjunto solução. Para solucionar esses tipos de problemas, os sistemas de classificação baseados em regras devem, de alguma forma, encorajar a formação de conjuntos de regras hierárquicos. Nesse sentido, os LCS apresentam uma característica interessante, que é a possível formação adaptativa de hierarquias default. Como descrito em 3.4.2, na estrutura de uma hierarquia default, uma regra exceção deve ter prioridade sobre as regras mais gerais (regras *default*). Favorecer a formação de hierarquias default permite construir soluções elegantes para problemas hierárquicos, por meio do refinamento do conjunto de regras (Smith e Goldberg, 1992). É importante ressaltar que, nesse texto, problemas hierárquicos se referem aos problemas em que o conjunto de regras solução se arranja de maneira hierárquica e não à classe de problemas relacionados à classificação hierárquica, onde existe uma hierarquia de rótulos ou classes.

Para solucionar o problema de classificação multirrótulo, esse trabalho propõe a representação do problema por meio de hierarquias default. Dessa maneira, um problema multirrótulo é visto como um conjunto de regras em uma estrutura hierárquica em que as regras no topo da hierarquia (regras *default*) advocam apenas um rótulo ou classe, enquanto que as regras nos níveis mais baixos da hierarquia (regras exceção) advocam duas ou mais classes.

Num problema multirrótulo, como alguns exemplos podem pertencer a mais de uma classe simultaneamente é possível ter regras que associem apenas uma porção do conjunto de classes real de um exemplo. Essas regras são consideradas parcialmente corretas pois mesmo que elas associem algumas das classes que estejam presentes ao conjunto de classes do exemplo, elas ainda

deixam de associar uma ou mais classes. Ao contrário de um problema de classificação único-rótulo, agora é possível ter regras corretas, incorretas e parcialmente corretas (ou parcialmente incorretas).

Para tentar separar corretamente as regras corretas, parcialmente corretas e incorretas, esse trabalho propõe a utilização da teoria do OCS, pois um dos objetivos desse é atingir uma melhor separação entre regras ótimas e sub-ótimas.

Este capítulo descreve o algoritmo proposto neste trabalho, denominado pelos autores de *Multi-label OCS* (ML-OCS), para realização de classificação multirrótulo por meio da união das teorias de formação de hierarquias default e do OCS. O capítulo está organizado da seguinte maneira. Na Seção 4.2 é apresentada a estrutura do sistema de produção de regras do ML-OCS, descrevendo desde a criação do conjunto de regras inicial até a forma como se resolvem os conflitos entre regras, a atualização dos valores de parâmetros das regras, a forma como o ambiente envia *feedback* ao sistema em forma de recompensa e o critério de parada do algoritmo. Em seguida, na Seção 4.3, é apresentado o componente responsável pela interação entre as regras e consequente dimensionamento entre comportamento individual e coletivo. Na Seção 4.4, descreve-se o AG utilizado para criar e inserir novas regras no sistema, possibilitando a evolução do conjunto solução. O funcionamento do algoritmo na fase de teste é descrito na Seção 4.5. Finalmente, na Seção 4.6, apresentam-se as últimas considerações deste capítulo.

4.2 Sistema de Produção de Regras

O sistema de produção de regras do ML-OCS funciona de forma parecida com o do OCS. Assim como no OCS, o sistema de produção do ML-OCS possui uma população de organizações, cada qual podendo possuir no mínimo 1 e no máximo N_c regras, onde N_c é o número máximo de regras na população. As organizações interagem entre si trocando regras por meio de um componente especial de crescimento organizacional, que será explicado mais adiante. Também como no OCS, cada organização possui dois valores de reputação, um de longo prazo (LTR) e outro de curto prazo (STR). As regras, por sua vez, também possuem dois parâmetros, mas diferentemente do OCS, aqui não se utilizam reputações. Ao invés disso, utilizam-se um valor de estimativa de recompensa (RE, do inglês *Reward Estimate*) e um valor que é o fator de prioridade (PF, do inglês *Priority Factor*), assim como no trabalho de Smith sobre hierarquias default. As regras utilizam os mesmos parâmetros propostos no trabalho de Smith pois o objetivo é favorecer a formação de hierarquias default dentro das organizações.

A idéia principal é unir o OCS e a teoria para formação de hierarquias default de Smith para criar um sistema que separe classificadores ótimos de sub-ótimos e que, ao mesmo tempo, permita a formação de hierarquias entre regras dentro de suas organizações. Dessa maneira, procura-se contruir um sistema que seja capaz de construir uma solução para problemas de classificação multirrótulo que possua apenas as regras corretas arranjadas em uma hierarquia. O algoritmo

4.1 apresenta os principais componentes do sistema de produção, que é o programa principal do ML-OCS, assim como a forma como esse interage com o AG e o componente de crescimento organizacional.

4.2.1 Covering para Geração de Regras

A primeira versão do ML-OCS é capaz de processar apenas conjuntos de dados com exemplos representados de forma binária. O antecedente de uma regra de classificação é representado utilizando uma notação ternária composta dos valores 0, 1 e do símbolo $\#$ (*don't care*). A porção referente ao consequente da regra é representada de forma binária, possuindo tantas posições quanto o número de rótulos de classe no problema, sendo que um valor 1 em uma posição indica a presença da classe representada por essa posição do vetor. O Apêndice A contém um exemplo.

A primeira etapa do algoritmo é a geração da população de regras inicial. Assim como em outros LCSs, como é o caso do XCS, a população inicial é gerada de maneira a garantir o suprimento de *building blocks* na população. Para isso, utiliza-se uma técnica bastante simples, denominada *covering*. *Covering* pode ser utilizada de diversas maneiras distintas. Nesse trabalho, gera-se para cada exemplo de treinamento uma regra que case com esse exemplo, utilizando uma probabilidade P_{dc} de inserir um símbolo $\#$ por bit do antecedente da regra. A regra criada para um dado exemplo irá possuir os mesmos rótulos de classe que o exemplo possua. Garante-se assim que, no início do ciclo de treinamento, cada exemplo possuirá pelo menos uma regra cujo antecedente case com o exemplo.

O mecanismo de *covering* também é utilizado durante o ciclo de treinamento em toda ocasião em que não exista uma regra correspondente na população para um dado exemplo. Isso pode acontecer pois o AG retira regras da população para inserir as novas regras criadas. Nesse caso, cria-se uma regra para ser utilizada com esse exemplo da mesma maneira que se criam as regras iniciais.

4.2.2 Mecanismo de Resolução de Conflitos

O primeiro passo do sistema de produção de regras é receber um exemplo de entrada do ambiente, ou seja, um exemplo do conjunto de treinamento. Quando o ambiente envia um exemplo de entrada para o sistema de produção este deve procurar por todas as organizações que possuam regras cujo antecedente case com o exemplo. Quando mais de uma organização é encontrada, o sistema irá decidir por aquela cujo valor de LTR for mais alto. Caso haja empate entre duas ou mais organizações, o sistema escolhe aleatoriamente uma delas. Embora escolher a organização com mais alto LTR pareça suficiente, pois o LTR representa a qualidade dessa organização, é necessário garantir um certo nível de exploração de diferentes organizações. Assim, com uma probabilidade P_{org}, o sistema não irá escolher a organização com maior valor LTR, escolhendo aleatoriamente uma entre as organizações possíveis. O valor de P_{org} utilizado no restante desse trabalho é de 10%.

Tendo escolhido a organização, o sistema então escolherá uma regra dentro desta para classificar o exemplo de entrada. Como nada impede que existam várias regras dentro da organização cujo antecedente case com o exemplo, é necessário um critério para escolher uma delas. Para decidir qual regra será acionada, o sistema calcula os *bids* das regras por meio do produto entre seus valores de parâmetros, exatamente como proposto no trabalho de Smith, ou seja, o *bid* de uma regra é o produto de seu RE pelo seu PF. A regra com mais alto *bid* será escolhida e sua ação será enviada ao ambiente.

4.2.3 Mecanismo de Alocação de Crédito

Uma vez escolhida uma regra e enviada sua ação ao ambiente, este irá enviar em retorno um valor de recompensa que indica a adequabilidade da ação escolhida, ou seja, a adequabilidade da regra escolhida para classificar o exemplo de entrada. Esse valor de recompensa é utilizado pelo sistema de produção para atualizar os valores de parâmetros da organização e da regra selecionada. O procedimento de atualização de parâmetros, ou procedimento de alocação de crédito, atualiza os parâmetros da regra, RE e PF, exatamente como nas Equações 3.1 e 3.3 respectivamente. O valor de STR da organização é atualizado como proposto no OCS e apresentado na Equação 3.6. Já o valor de LTR da organização é atualizado de maneira distinta em relação ao proposto pelo OCS. O sistema armazena, para cada organização, os N últimos valores de recompensa recebidos e atualiza o LTR da organização calculando a média dos N valores.

No OCS, o valor LTR é atualizado utilizando o número de vezes em que a organização obteve recompensa máxima do ambiente durante todo o seu tempo de vida. Entretanto, o OCS não propunha um mecanismo de geração e inclusão de novas regras na população. Como será explicado na Seção 4.4, o ML-OCS inclui as novas regras geradas pelo AG em organizações recém criadas para recebê-las. Utilizar o mesmo mecanismo de atualização de LTR usado pelo OCS deixa de fazer sentido, uma vez que organizações recém criadas e organizações antigas teriam seus respectivos LTR modificados usando taxas distintas. A maneira como o valor de LTR é atualizado pelo ML-OCS garante que organizações antigas e organizações recém criadas tenham seu LTR modificado na mesma proporção.

4.2.4 Recompensas do Ambiente

Ao escolher uma regra, o sistema de produção envia a ação advocada por esta ao ambiente. O ambiente, por sua vez, deve possuir um critério para recompensar a regra em questão baseado na ação recebida. Neste trabalho, o critério de recompensa é baseado no fato de que o sistema possui regras corretas, parcialmente corretas e incorretas, desejando-se atribuir diferentes níveis de recompensa para diferentes situações. Para calcular a recompensa que uma regra deve receber, primeiramente calcula-se a diferença simétrica entre o conjunto de rótulos de classe advocados pela regra e o conjunto real de rótulos de classe do exemplo de entrada atual. A diferença simétrica

Entrada: Um exemplo de entrada
Gera população p utilizando covering;
p ← `GeraPopulação(`N_c, P_{dc}`)`;
enquanto *Critério de Parada não atendido* **faça**
  Constrói Match-Set de organizações;
  MatchSet ← `MatchingOrgs(`p, $Entrada$`)`;
  Se atingiu o tempo estabelecido, aplica o AG e desabilita a aplicação do OGC;
  se $SinalAG = Verdadeiro$ **então**
    `AplicarAG(`MatchSet`)`;
    SinalOGC ← Falso;
    SinalAG ← Falso;
  fim
  Seleciona org com maior LTR; Com probabilidade P_{org} seleciona org aleatória;
  org ← `SelecionaOrg(`MatchSet, P_{org}`)`;
  Seleciona regra com maior bid;
  regra ← `SelecionaRegra(`org, $Entrada$`)`;
  Envia ação ao ambiente, recebendo uma recompensa;
  r ← `EnviarAção(`regra[ação]`)`;
  Atualiza valores de parâmetros de org *e* regra ;
  `AlocarCrédito(`org, regra, r`)`;
  Se completou o conjunto de treinamento, calcular métricas para habilitação de OGC, AG e para verificação de critério de parada;
  se `CompletouConjuntoTrein()` $= Verdadeiro$ **então**
    precisão ← `CalcularPrecisão(`p, *Validação*`)`;
    acurácia ← `CalcularAcurácia(`p, *Validacao*`)`;
    se `AlcançouTimeA()` $= Verdadeiro$ **então**
      SinalOGC ← Verdadeiro;
    fim
    se $SinalOGC = Verdadeiro$ **então**
      se `VerificaCritérioParada(`acurácia`)` **então**
        Parar o treinamento;
        `CalcularMétricasAvaliação(`p, $Teste$`)`;
        Parar o algoritmo;
      senão se `AlcançouTimeB(`precisão `)` $= Verdadeiro$ **então**
        SinalAG ← Verdadeiro;
      fim
    fim
  fim
  Se atingiu o tempo estabelecido e estiver habilitado, aplica o OGC;
  se $TimeOGC$ **e** $SinalOGC = Verdadeiro$ **então**
    `AplicarOGC(`p`)`;
  fim
fim

Algoritmo 4.1: Sistema de Produção do ML-OCS e sua interação com OGC e AG

entre dois conjuntos A e B é o conjunto de todos os x tal que $x \in A$ ou $x \in B$, mas não a ambos. Em seguida, calcula-se o tamanho desse conjunto e divide-se o resultado pelo número de classes do problema. A recompensa é então calculada subtraindo esse resultado de 1. A Tabela 4.1 mostra um exemplo. Nesse exemplo, a primeira coluna representa o exemplo de entrada atual, a segunda coluna representa uma regra selecionada (na forma antecedente/consequente), a terceira coluna representa a ação correta para o exemplo de entrada e, por fim, a quarta coluna representa a recompensa recebida quando a regra da coluna 2 é selecionada para classificar o exemplo da coluna 1.

Exemplo de Entrada	Regra Selecionada	Rótulos Corretos	Recompensa
00	00/1;2	1;2	1,00
00	00/1;0	1;2	0,66
00	00/1;3	1;2	0,33
00	00/3;0	1;2	0,00

Tabela 4.1: Exemplo do esquema de recompensa utilizado

É interessante notar que o cálculo do valor de recompensa é baseado na medida *Hamming Loss* (HL) (Tsoumakas e Katakis, 2007), pois a recompensa R de uma regra é na verdade calculada como o oposto da HL entre o conjunto de classes do exemplo de entrada e o conjunto de classes da regra escolhida. A maneira como a recompensa é atribuída induz diferentes RE para cada tipo de regra. Isso é importante pois as regras proverão ao AG um guia de *fitness* (do inglês, *fitness guidance*), uma vez que, como será explicado na Seção 4.4, o AG utiliza o RE das regras como *fitness*. Não prover um guia de *fitness* ao AG reduz sua busca a uma busca cega entre regras ruins até encontrar a solução ótima, como num problema do tipo "agulha no palheiro".

4.2.5 Critério de Parada

Como o ML-OCS está continuamente tentando reorganizar suas regras entre as organizações, algumas vezes o sistema perde desempenho (em termos de acurácia do conjunto de regras) e precisa de algum tempo para retornar ao estado anterior. Portanto, é necessário escolher um momento apropriado para terminar a execução do algoritmo.

O critério de parada do ML-OCS é baseado na acurácia do conjunto de regras em um conjunto de validação. O algoritmo terminará quando obtiver no mínimo $x\%$ de acurácia no conjunto de validação por y vezes consecutivas. Uma vez que não se pode saber *a priori* se é realmente possível obter $x\%$ de acurácia em um determinado problema, é necessário introduzir um critério de parada extra ao algoritmo. Assim, o que se faz é fixar um valor N_{parada} de exemplos vistos pelo sistema e, ao atingir N_{parada}, parar o algoritmo. O Algoritmo 4.1 demonstra o funcionamento do sistema de produção, que é considerado o programa principal do ML-OCS.

4.3 Componente de Crescimento Organizacional

O componente de crescimento organizacional (OGC, do inglês *Organizational Growth Component*) é executado a cada k iterações e é responsável por rearranjar as organizações, favorecendo regras com desempenho similar a serem mantidos juntos em uma mesma organização. O OGC do ML-OCS é bastante parecido com o respectivo componente do OCS. Entretanto, algumas alterações foram realizadas visando obter um melhor resultado. No ML-OCS o OGC também utiliza os operadores *Grow* e *Shrink* para alterar os membros das organizações.

O operador *Grow*, assim como no OCS, é responsável por aumentar o tamanho das organizações. Ele funciona da mesma forma que no OCS, utilizando o STR das organizações para o cálculo de seus respectivos salários e, a partir desses valores, decidir qual organização ficará com a regra selecionada, se aquela que já possui a regra ou aquela que a regra deseja entrar. Além do teste do salário das organizações, o operador Grow do ML-OCS utiliza mais dois testes para decidir qual organização ficará com a regra. O primeiro teste compara o desempenho da regra com o desempenho da organização que esta deseja entrar. Somente é permitido à regra deixar sua organização atual para entrar em outra organização se o seu desempenho for no mínimo igual a uma porcentagem p do desempenho da organização. Esse teste foi adicionado para garantir que regras sabidamente ruins não entrem em organizações de boas regras. Considera-se o desempenho de uma regra como sendo o seu RE e o desempenho de uma organização como a média do desempenho de todas as suas regras, ou seja a média dos RE de todas as regras contidas na organização. Por exemplo, se o desempenho da organização selecionada para sofrer a aplicação do operador *Grow* for igual a $0,85$, então a regra selecionada deverá possuir, no mínimo, um RE igual a $p\%$ de $0,85$ para que esta possa deixar a sua organização atual e entrar na organização selecionada pelo operador. O segundo teste consiste em uma verificação de qual organização é selecionada mais vezes no conjunto de validação. A regra selecionada somente mudará de organização se a organização a qual ela pertence tiver sido selecionada menos vezes. O objetivo é tentar impedir trocas aleatórias de regras entre duas organizações consideradas boas e inserir uma pressão para que todas as boas regras sejam reunidas em apenas uma organização. Portanto, todos os três testes devem ser satisfeitos para que uma regra mude de organização.

Todos os três testes são realizados somente no caso de não existir uma regra igual na organização que a regra deseja entrar. Caso isso aconteça, o algoritmo decide imediatamente por não inserir duplicatas na organização e mantém a regra na organização em que ela se encontra.

O operador Shrink também funciona como no OCS, com duas pequenas alterações. Originalmente, o operador Shrink utilizado pelo OCS compara a medida de qualidade das regras de uma organização com um limiar próximo de zero. Todas as regras cuja medida de qualidade estiver abaixo desse limiar são retiradas da organização. No ML-OCS, compara-se a medida de qualidade de uma regra (seu RE) com uma porcentagem da média do desempenho da organização, medida da mesma maneira que no operador Grow. As regras que possuírem RE abaixo dessa porcentagem do desempenho médio são retiradas da organização e colocadas cada qual em uma organização va-

zia. Além disso, o operador Shrink também pode retirar regras de uma organização caso existam regras dentro dessa mesma organização que sejam mais gerais. Por exemplo, dadas duas regras $R1$ e $R2$ de uma mesma organização, se $R1$ e $R2$ possuem os mesmos rótulos de classe (consequentes iguais) e $R1$ possui antecedente mais geral que $R2$, então $R2$ será removida da organização. Nesse caso, a regra retirada também é colocada em um organização vazia. Essas organizações são criadas especificamente para receber as regras removidas pelo Shrink e necessitam de inicialização de seus parâmetros. O LTR dessa organização deverá ser setado com um valor baixo, uma vez que a regra que será inserida nessa organização foi considerada ruim, e portanto, não faz sentido dar muito crédito à organização. No decorrer deste trabalho, esse valor foi fixo em $0,6$. Para inicializar o STR de forma coerente, e impedir que essa regra retorne para a organização da qual acabou de sair, subtrai-se um pequeno valor do STR da organização antiga e utiliza-se esse valor como STR da organização recém criada. Nesse trabalho, esse valor foi fixo em $0,1$. Os parâmetros da regra que será inserida na nova organização não são alterados. O funcionamento geral do OGC é apresentado no Algoritmo 4.2.

4.4 Mecanismo para Descoberta de Novas Regras

O ML-OCS emprega um AG como mecanismo de descoberta de novas regras de classificação. O AG desenvolvido procura considerar uma característica particular de um problema de classificação multirrótulo: a possível existência de uma terceira categoria de regras que apresenta soluções parcialmente corretas. Quando uma regra é parcialmente correta, ela receberá do ambiente uma recompensa parcial, de acordo com a quantidade de rótulos preditos corretamente. Como essa regra sempre receberá recompensa parcial, seu RE terá um valor intermediário. Entretanto, existe a possibilidade de que uma regra possua os rótulos de classe corretos, mas cobre mais exemplos do que deveria cobrir e, portanto, precisa ser especializada. Nesse caso, a regra também terá RE intermediário, pois ela é penalizada por cobrir exemplos que não deveria. Uma vez que regras corretas e regras incorretas são facilmente detectadas por meio do RE e que, por outro lado, um valor de RE intermediário não é suficiente para induzir qual das situações anteriores se aplica, utiliza-se o desvio padrão das recompensas recebidas pela regra numa tentativa de prover conhecimento extra ao AG quando este se depara com uma regra com RE intermediário.

O AG é executado a cada $TimeGA$ iterações, selecionando duas regras pais do *Match-set* de regras(conjunto de todas as regras que cobrem o exemplo de entrada) atual utilizando seleção por roleta com o RE como *fitness*. É importante ressaltar que o AG considera apenas as regras do *Match-set*, não considerando suas respectivas organizações. O AG utiliza dois operadores de mutação e não utiliza operador de *crossover*. O primeiro operador (MA) é responsável por realizar uma mutação no antecedente da regra, alterando um bit da regra com probabilidade P_m. O segundo operador (MC) realiza uma mutação no consequente da regra, adicionando (ou removendo) uma classe aleatória ao conjunto de classes da regra.

Entrada: População de regras p
Saída: População atualizada

para cada Org *em* p **faça**
  operador ← `SelecionaOperador()`;
  se operador $= Grow$ **então**
    Seleciona uma regra não contida em Org;
    regra ← `SelecionaRegra(`p, Org`)`;
    orgRegra ← Organização correspondente a regra;
    Se Org já contém regra *não inserir cópia; Senão se salário de Org maior que salário de* orgRegra*, se regra possui mínima performance em relação a performance média de Org e se Org é selecionada mais vezes que* orgRegra *no conjunto de validação, troque a regra de organização*;
    se `ContémRegra(`Org, regra`)` $= Verdadeiro$ **então**
      Não insira regra *em Org*;
    senão se `MaiorSalário(`Org, orgRegra`)` **e**
    `MínimaPerformance(`Org, regra`)` **e**
    `MaisSelecionada(`Org, orgRegra, *Validação*`)` **então**
      Org ← `TrocarDeOrganização(`regra`)`;
    fim
  senão se operador $= Shrink$ **então**
    para cada regra *em* Org **faça**
      Se a regra está abaixo da performance média da organização ou se existe outra regra mais geral nessa organização, retire a regra;
      se `MínimaPerformance(`Org, regra`)` $= Falso$ **ou**
      `ExisteRegraMaisGeral(`Org, regra`)` $= Verdadeiro$ **então**
        `RetirarRegra(`Org`)`;
        p ← `CriarNovaOrganização(`regra`)`;
      fim
    fim
  fim
fim

Algoritmo 4.2: Componente de Crescimento Organizacional (OGC)

Para cada pai selecionado, o AG calcula o desvio padrão de suas M últimas recompensas recebidas. O desvio padrão é então utilizado para calcular a probabilidade P_{ma} de aplicação do operador de mutação MA. Para calcular essa probabilidade, utiliza-se uma interpolação linear entre dois pontos conhecidos (x_0, y_0) e (x_1, y_1), onde o eixo x representa o desvio padrão e o eixo y representa a probabilidade de se aplicar o operador MA. Neste trabalho, fixou-se $(x0, y0) = (0, 0)$ e $(x1, y1) = (0,5, 1)$. Caso o desvio padrão das recompensas recebidas pela regra seja elevado, conclui-se que a melhor opção para tentar obter um filho que seja melhor que a regra pai é realizar modificações no seu antecedente, uma vez que essa regra está recebendo valores de recompensas distintos ao ser ativada, indicando uma regra que cobre uma porção do espaço de soluções que não deveria. Caso o contrário ocorra, ou seja, o desvio padrão seja próximo de zero, conclui-se que a melhor alternativa é realizar uma modificação no consequente, criando uma regra com o mesmo antecedente que a regra pai, porém com um conjunto de rótulos de classes distinto.

Utilizando a probabilidade calculada, o algoritmo decide qual operador utilizar, MA ou MC, da seguinte maneira. Gera-se um valor aleatório no intervalo $[0, 1]$ utilizando uma distribuição uniforme. Se o valor gerado for inferior a P_{ma}, o algoritmo aplicará MA, caso contrário o algoritmo decidirá pela aplicação de MC.

Entretanto, existe ainda uma situação a ser considerada. Regras com alto valor de RE geralmente cobrem somente os exemplos que deveriam cobrir e possuem o conjunto correto de rótulos de classe. Segundo o esquema de escolha entre os operadores MA e MC, a probabilidade de aplicação de MA seria muito baixa, uma vez que o desvio padrão da regra seria muito próximo a zero, e portanto MC seria escolhido. Porém, escolher MC nessa situação não é a escolha mais interessante, uma vez que muito provavelmente essa regra já possui o conjunto de rótulos de classe adequado. Para evitar a aplicação de MC nessa situação, o AG primeiramente verifica o RE da regra, e caso considere que essa é uma regra muito recompensada, decidirá pela aplicação de MA. Aplicar MA ao invés de MC nesse caso é uma alternativa mais adequada, pois permite ao AG direcionar sua busca para regras mais gerais.

Os dois filhos gerados pelo AG serão introduzidos na população através de substituição de estado uniforme, ou seja, dois indivíduos serão selecionados para deixar a população e os dois novos indivíduos tomarão seus lugares. Os dois indivíduos que deixarão a população são aqueles que possuem os dois menores RE no *Match-set*, realizando assim uma espécie de reposição em nicho. Cada nova regra inserida na população será colocada em uma organização criada para recebê-la. As organizações criadas possuem LTR e STR inicializados com valor 1, com objetivo de dar-lhes chances de serem selecionadas e avaliadas. Para manter consistência com o valor de LTR, inicializam-se as N últimas recompensas recebidas também com o valor 1. O RE das novas regras é inicializado como 90% do RE de seus respectivos pais e seus fatores de prioridade são inicializados como 0. O algoritmo 4.3 ilustra os principais passos do AG utilizado.

Como mencionado anteriormente, o AG é executado a cada $TimeGA$ iterações do algoritmo. O tempo entre uma aplicação do AG e outra é estabelecido considerando-se quanto tempo o sistema necessita para se reorganizar após a inserção de novas regras. Mais especificamente, quanto tempo

```
Entrada: População de regras p, Match-Set atual
Saída: População atualizada

Selecionar duas regras pais utilizando seleção por roleta no Match-Set de regras atual;
pais ← SelecionarPais(MatchSet);
para cada pai em pais faça
    Calcular o desvio padrão das recompensas recebidas pela regra;
    dp ← CalcularDP(pai);
    Utilizando dp, calcular a probabilidade de aplicação do operador MA;
    pma ← ProbabilidadeMA(dp);
    Gerar um valor aleatório entre [0, 1] usando uma distribuição uniforme;
    v ← ValorAleatório();
    se v < pma então
        filho ← OperadorMA(pai);
    senão
        Verifique a recompensa da regra;
        se RegraAltaRecompensa(pai) então
            filho ← OperadorMA(pai);
        senão
            filho ← OperadorMC(pai);
        fim
    fim
fim
Inserir filhos gerados na população;
InserirFilhos(p, filhos);
```

Algoritmo 4.3: Procedimento do Algoritmo Genético

o algoritmo levará para se recuperar caso regras ruins tenham sido inseridas na população pelo AG. No ML-OCS, estabelecer esse tempo é muito importante pois regras estão em uma constante mudança de organizações. Uma regra recém criada não deveria sofrer a aplicação de nenhum operador do componente de crescimento organizacional, simplesmente porque essa regra não foi suficientemente avaliada. Para calcular $TimeGA$, divide-se o problema em duas partes: $TimeA$ e $TimeB$.

Inicialmente, calcula-se quanto tempo é necessário ao sistema para selecionar a nova organização, utilizar a regra dentro dela e perceber que trata-se de uma regra ruim, retornando à solução anterior ($TimeA$). Durante $TimeA$ não será permitido ao sistema aplicar o OGC. Em seguida, estabalece-se um tempo extra em que permite-se ao sistema voltar a aplicar este componente ($TimeB$). Somente ao término de $TimeB$ será permitido reaplicar o AG.

Para calcular $TimeA$, calcula-se o número de vezes m que a nova organização precisa ser selecionada até que seu LTR decaia para um valor α. Considerando que a nova organização é inserida na população com LTR igual a 1 e N últimas recompensas também iguais a 1, calcula-se m como na equação 4.1.

$$\frac{N - m + cm}{N} = \alpha \tag{4.1}$$

A qual pode ser transformada como

$$m = \left\lceil \frac{(1-\alpha)N}{(1-c)} \right\rceil \tag{4.2}$$

onde c é uma constante que indica qual tipo de regra se está considerando. Quanto menor o valor de c, pior a regra.

Ao invés de contar o número de vezes que a organização foi selecionada, o ML-OCS estima essa quantidade baseando-se em considerações conservadoras de generalidade. Assume-se que cada regra cobre pelo menos 5% do espaço de características. Consequentemente, o número de exemplos no conjunto de treinamento cobertos por uma regra é pelo menos igual a 5% do tamanho do conjunto de treinamento. Assim, estima-se o número de repetições n do conjunto de treinamento necessárias para selecionar a nova organização pelo menos m vezes, como na Equação 4.3.

$$0.05 * Nexemplos * n = m \tag{4.3}$$

Que pode ser transformada em

$$n = \left\lceil \frac{20m}{Nexemplos} \right\rceil \tag{4.4}$$

onde $Nexemplos$ é o número de exemplos no conjunto de treinamento. Ainda é preciso considerar que, com uma certa porcentagem, o sistema escolherá uma organização aleatória entre todas

as que possuem regras que cobrem o exemplo de entrada. Uma vez que nesse trabalho essa porcentagem é fixa e igual a 10%, calcula-se o novo número de repetições do conjunto de treinamento (n') como

$$n' = \lceil 1,11n \rceil \tag{4.5}$$

Todos os cálculos acima consideram uma regra inserida pelo AG. Como o AG insere não apenas uma, mas duas novas regras por vez, é necessário dobrar o valor de n', pois os pais são selecionados do mesmo *Match-set* e, portanto, é esperado que os filhos também pertençam ao mesmo *Match-set*, dependendo é claro do operador MA. Se os dois filhos fazem parte do mesmo *Match-set*, eles irão competir entre si todas as vezes que cobrirem um exemplo. Portanto,

$$TimeA = 2n' \tag{4.6}$$

Em seguida, calcula-se $TimeB$ considerando a precisão do sistema no conjunto de validação. Se o sistema mantém a mesma precisão no conjunto de validação por y repetições do conjunto de treinamento, ou se o sistema atinge um número fixo z de repetições do conjunto de treinamento, é permitido aplicar o AG novamente. Note que a precisão do conjunto de regras é calculada em um conjunto de validação apenas após a apresentação de todos os exemplos de treinamento. Por exemplo, $y = 10$ e $z = 30$ significa que o AG será aplicado novamente se por 10 repetições completas do conjunto de treinamento consecutivas a precisão no conjunto de validação permanecer a mesma, ou se esse fato não acontecer em 30 repetições do conjunto de treinamento. É importante ressaltar que, como mencionado anteriormente, durante $TimeB$ o sistema aplicará o OGC.

4.5 Funcionamento na Fase de Teste

Ao atingir o critério de parada, o algoritmo possui um conjunto de organizações contendo regras, no qual está representado o conjunto de regras solução para o problema de classificação multirrótulo. Entretanto, nem todas as organizações são úteis ao término do algoritmo. Assim, na fase de teste, ao receber um exemplo de entrada a ser classificado, o algoritmo aplica o mesmo mecanismo básico da fase de treinamento, ou seja, os mesmos critérios para escolha da organização a ser utilizada e da regra dentro desta. Existem, entretanto duas diferenças básicas. A primeira delas é que a ação correspondente a regra escolhida, ou seja, seu conjunto de rótulos, não é enviada ao ambiente com o objetivo de receber uma recompensa em resposta. O conjunto de rótulos é utilizado em comparação com o conjunto real de rótulos de classe do exemplo para cálculo das métricas de avaliação do classificador. Nesse trabalho utilizam-se quatro medidas de avaliação, como explicadas no Capítulo 2: Hamming Loss, Acurácia, Precisão e Revocação.

A outra diferença diz respeito ao critério seguido para selecionar qual organização utilizar. Ao final do treinamento pode acontecer de algumas organizações que fazem parte do mesmo *Match-set*

para um dado exemplo possuírem o mesmo valor de LTR. Ou seja, ao procurar pela organização com máximo LTR, acontece um empate entre duas ou mais organizações. Na fase de treinamento, caso isso aconteça, simplesmente escolhe-se aleatoriamente uma das organizações, pois todas aparentam ser igualmente boas. Entretanto, na fase de teste, isso é feito de maneira diferente para garantir a maior generalização possível. Caso haja empate entre duas ou mais organizações, verifica-se qual regra seria escolhida em cada uma delas. Se as regras advocarem a mesma ação, será escolhida a organização daquela que possuir um antecedente mais geral, e consequentemente, essa regra será selecionada para classificar o exemplo. Essa forma de escolher a organização ajuda a minimizar o tamanho do conjunto final de regras que formam o conjunto solução para o problema.

4.6 Considerações Finais

Nesse capítulo foi apresentado o algoritmo para classificação multirrótulo desenvolvido nesse trabalho, o ML-OCS. O ML-OCS utiliza conceitos do OCS e da teoria para favorecimento de formação de hierarquias default, unindo-os em um algoritmo para classificação multirrótulo que representa esse tipo de problema por meio de um conjunto de regras hierárquico. Com o uso do OGC, o algoritmo rearranja suas regras buscando colocar regras com RE parecido em uma mesma organização. Esse componente foi baseado no componente de mesmo nome do algoritmo OCS, porém com algumas modificações propostas para melhorar o desempenho geral do sistema por meio de trocas menos aleatórias de regras entre organizações. Por meio do AG, o ML-OCS tem a oportunidade de procurar por novas soluções baseando-se nas regras atuais. O AG do algoritmo foi proposto de maneira a se adaptar ao problema multirrótulo, aplicando operadores genéticos mais apropriados a cada situação.

Assim, o objetivo do algoritmo é conseguir produzir um conjunto de regras adequado ao problema, por meio da indução de regras corretas a partir de regras parcialmente corretas e da separação entre os tipos de regras em diferentes organizações.

CAPÍTULO

5

Experimentos

5.1 Considerações Iniciais

Neste capítulo, o algoritmo proposto no Capítulo 4 é avaliado por meio de experimentos envolvendo a classificação de três conjuntos de dados, um deles artificial e dois reais. Todos os conjuntos de dados representam seus atributos e classes de maneira binária, onde um valor 1 em um atributo (ou classe) representa a presença desse no exemplo e um valor 0 representa a ausência. O ML-OCS foi treinado e testado em cada conjunto seguindo uma metodologia de validação cruzada, para possibilitar uma análise de resultados confiável. Para avaliar a qualidade dos resultados obtidos pelo ML-OCS em cada conjunto de dados, outros três métodos de classificação multirrótulo existentes na literatura foram aplicados aos mesmos conjuntos e os resultados entre o ML-OCS e cada um desses métodos foi comparado por meio de testes estatísticos. Os métodos utilizadas foram Binary Relevance, Label Power-Set e ML-KNN.

Além disso, uma análise do conjunto de regras obtido pelo ML-OCS é realizada com o objetivo de fornecer um panorama inicial de como o algoritmo organiza seu conhecimento.

O capítulo está organizado da seguinte maneira. Na Seção 5.2, apresentam-se as principais características de cada conjunto de dados utilizado nos experimentos. Em seguida, a Seção 5.3 descreve a metodologia utilizada para condução dos experimentos e para análise dos resultados. A Seção 5.4 apresenta os resultados obtidos separadamente para cada conjunto de dados, juntamente com uma análise inicial do conjunto de regras geradas pelo ML-OCS. Por fim, na Seção 5.5, são discutidos os resultados dos experimentos realizados e apresentadas as principais conclusões.

5.2 Conjuntos de Dados Utilizados

Para a condução dos experimentos foram utilizados três conjuntos de dados binários distintos, uma vez que a primeira implementação do ML-OCS não é capaz de tratar dados que não sejam binários. O primeiro deles é um conjunto de dados artificial que foi gerado especialmente para avaliar o desempenho do ML-OCS em relação a outros classificadores multirrótulo. Esse conjunto é formado por todos os exemplos gerados a partir das seguintes regras de classificação.

1. $0\#\#\#\#\# \Rightarrow 1,0$
2. $1\#\#\#\#\# \Rightarrow 0,1$
3. $00\#\#\#\# \Rightarrow 1,1$

Como pode-se perceber, no conjunto de regras apresentado existe uma hierarquia default entre a primeira e a terceira regra e o número de rótulos de classe existentes é igual a dois.

Também foram utilizados dois conjuntos de dados originários do conjunto Uniprot [1], que é um dos maiores conjuntos de dados de bioinformática. Os conjuntos foram criados e utilizados para avaliação do algoritmo Mulan (Chan e Freitas, 2006) e foram gentilmente cedidos para utilização nesse trabalho. Os criadores dos conjuntos de dados criaram os conjuntos a partir de referência cruzada entre o conjunto Uniprot e o conjunto PROSITE [2], que é um conjunto de dados que armazena famílias e domínios de proteínas. No conjunto Uniprot, cada registro representa uma proteína, e contém uma referência para cada padrão PROSITE presente naquela proteína. Os criadores utilizaram cada padrão PROSITE como um atributo binário, indicando a presença ou ausência desse padrão na proteína. O número de rótulos de classe existentes também é igual a dois em ambos os conjuntos.

A Tabela 5.1 ilustra as principais características dos conjuntos de dados utilizados.

O conjunto de dados Anti-oncogene Apoptosis originalmente possuía 540 exemplos. Entretanto, notou-se que, desses exemplos, apenas 136 eram exemplos distintos. O motivo para a repetição de exemplos é que a representação utilizando os padrões PROSITE gera um conjunto de dados bastante esparso devido a pequena quantidade de padrões presentes em cada proteína. Em consequência, muitos exemplos acabam sendo idênticos nessa representação em alto nível de abstração, embora as proteínas que geraram os exemplos sejam distintas. O mesmo ocorre no conjunto Cell-cycle Cell-division. Para utilização nesse trabalho, os conjuntos de dados foram reduzidos em relação a seu tamanho original eliminando as repetições.

[1] Universal Protein Resource; http://www.uniprot.org. Acesso em 10/01/2009

[2] Prosite - Database of protein domains, families and functional sites; http://www.expasy.org/prosite. Acesso em 10/01/2009

	Artificial	Anti-oncogene Apoptosis	Cell-cycle Cell-division
Número de exemplos	52	136	144
Número de atributos	6	153	156
Número de classes	2	2	2
Num. exemplos com classe 1	11	35	73
Num. exemplos com classe 2	27	95	43
Num. exemplos com classes 1 e 2	14	6	28
Cardinalidade	1,27	1,04	1,19
Densidade	0,63	0,52	0,60

Tabela 5.1: Características dos conjuntos de dados com 2 rótulos de classe

5.3 Metodologia

Nessa seção é descrita a metodologia utilizada para a condução dos experimentos. São apresentadas as ferramentas de software utilizadas, a forma de avaliação dos classificadores gerados e os testes estatísticos utilizados para comparação de resultados. A motivação para uma descrição detalhada da metodologia utilizada é facilitar a análise dos resultados e possibilitar a repetição dos experimentos realizados.

5.3.1 Ferramentas de Software Utilizadas

Para implementação e teste do ML-OCS foi utilizada a linguagem funcional e ambiente de software R (R Development Core Team, 2008). O R é uma ferramenta livre para computação estatística e pode ser instalado em uma grande variedade de plataformas e arquiteturas de computadores. Além disso, utilizou-se a linguagem Python para geração de *scripts* para pré-processamento dos conjuntos de dados.

O pacote Mulan (Tsoumakas et al., 2007) foi utilizado para conduzir os experimentos com os três demais métodos de classificação, Binary Relevance, Label Power-Set e ML-KNN. Mulan é um conjunto de classes escritas em Java que funciona sobre o software Weka (Witten e Frank, 2005) e contém vários métodos para classificação multirrótulo implementados. O pacote também contém um *framework* de avaliação com várias medidas de avaliação para classificadores multirrótulo, além de uma classe que fornece estatísticas sobre os conjuntos de dados utilizados.

5.3.2 Avaliação de Resultados

Todos os algoritmos utilizados nesse trabalho foram avaliados segundo quatro métricas de avaliação. Essas métricas são: Hamming Loss, Acurácia, Precisão e Revocação. As fórmulas para o cálculo de cada uma das quatro métricas são as mesmas apresentadas na Seção 2.4.4. Buscando garantir maior confiabilidade nos resultados, utilizou-se o método de amostragem conhecido como *k-fold cross-validation* estratificado. Nesse método, divide-se o conjunto de exemplos em k *folds*

ou partições. Em cada iteração do algoritmo, um *fold* é utilizado como conjunto de teste e os demais formam o conjunto de treinamento. Esse processo é repetido um número de vezes igual ao número de *folds*, de modo que cada *fold* é utilizado uma vez como conjunto de teste. O erro, ou qualquer métrica que se deseje, é medido como a média dos erros em cada um dos subconjuntos de treinamento e teste. A diferença desse método para o *k-fold cross-validation* tradicional é que, ao formar as partições mantêm-se a proporção dos exemplos presentes em cada uma das classes do conjunto completo.

No caso específico do ML-OCS, um dos *folds* é utilizado como conjunto de validação em determinadas partes do algoritmo, como explicado no Capítulo 4. Assim, tem-se $k-2$ *folds* utilizados no treinamento, 1 *fold* para teste e 1 *fold* para validação.

Em todos os conjuntos de dados utilizados, o número de exemplos que possuem ambos os rótulos de classe, ou seja, pertencem as classes 1 e 2, é relativamente menor que o número de exemplos pertencentes a apenas uma classe. Esse fato foi considerado para a escolha do número de *folds* em que os conjuntos originais foram divididos, de maneira a haver pelo menos um exemplo com mais de um rótulo de classe em cada *fold*.

5.3.3 Testes Estatísticos

Os testes estatísticos são uma estratégia para validação de resultados e comparação de experimentos que vêm recebendo cada vez mais atenção da comunidade de AM. A idéia principal é comparar os resultados obtidos em um conjunto de dados por diferentes algoritmos de modo a verificar se esses diferem uns dos outros com significância estatística. Neste trabalho, utilizou-se o teste estatístico t-student para dados pareados corrigido (Nadeau e Bengio, 2003) para comparar o desempenho do ML-OCS em relação aos métodos Binary Relevance, Label Power-Set e ML-KNN.

Segundo esse teste, para comparar dois algoritmos A e B, basta comparar o desempenho preditivo do classificador gerado por A (C_A) com o do classificador gerado por B (C_B). Dados k *folds* gerados pelo método *k-fold cross-validation*, calcula-se a diferença entre os classificadores gerados por A e B para cada *fold* de teste i, como na Equação 5.1.

$$dif_i = R_i(C_A) - R_i(C_B) \tag{5.1}$$

onde R representa o resultado obtido por um classificador e pode ser qualquer medida de avaliação utilizada para avaliar o desempenho preditivo de classificadores.

Em seguida, calcula-se a média M_d e o desvio padrão S_d das diferenças obtidas, como nas equações 5.2 e 5.3.

$$M_d = \frac{1}{k}\sum_{i=1}^{k} dif_i \tag{5.2}$$

$$S_d = \sqrt{\frac{1}{k-1}\sum_{i=1}^{k}(dif_i - M_d)^2} \tag{5.3}$$

Com os valores de média e desvio padrão, calcula-se o valor de t observado segundo a Equação 5.4.

$$t_{obs} = \frac{|M_d|}{\sqrt{(\frac{1}{k} + \frac{1}{k-1})S_d^2}} \tag{5.4}$$

A hipótese de que não há significância estatística entre as diferenças nos resultados de A e B, ou hipótese nula, é rejeitada se $t_{obs} > t_{(k-1,\frac{\alpha}{2})}$, que corresponde ao valor de t na distribuição t-student com $k-1$ graus de liberdade e nível de significância α. Caso a hipótese nula seja rejeitada, pode-se afirmar que existe diferença estatística entre os classificadores gerados por A e B com um nível de significância no teste igual a α.

Esse procedimento é adequado para realizar a comparação de dois algoritmos. Quando múltiplos algoritmos são comparados, entretanto, é necessário realizar uma correção no nível de significância do teste para garantir a sua validade. Isso é necessário porque ao se comparar diversos pares de algoritmos, a chance de ocorrer uma significância ao acaso é proporcional ao número de comparações que são realizadas. Assim, ajusta-se o nível de significância do teste em relação ao número de comparações realizadas entre algoritmos. Um ajuste que pode ser utilizado é a Correção de Bonferroni, que consiste em calcular um novo nível de significância α^*, utilizando o nível de significância original α e o número de comparações n, como pode ser observado na Equação 5.5.

$$\alpha = 1 - (1 - \alpha^*)^n \tag{5.5}$$

5.4 Resultados

Nessa seção são apresentados os principais resultados observados nos experimentos realizados. No A podem ser encontrados resultados preliminares obtidos durante a construção do ML-OCS. No Apêndice B, os resultados descritos nesta seção são apresentados de maneira detalhada.

5.4.1 Conjunto de Dados Artificial

Como mencionado anteriormente, foram conduzidos experimentos utilizando o ML-OCS e outros três métodos existentes na literatura de classificação multirrótulo. Para cada método, foram realizados 10 experimentos, cada qual consistindo na aplicação da técnica *k-fold cross-validation* estratificado com $k = 5$. O número de *folds* foi escolhido baseado no tamanho do conjunto de dados e na quantidade de exemplos pertencentes às classes 1 e 2 simultaneamente.

Os métodos *Binary Relevance* e *Label Power-Set* consistem em transformar o problema multirrótulo como explicado em 2.4.1. O algoritmo aplicado por ambos os métodos após a transformação do problema foi o C4.5 (Quinlan, 1993), com os parâmetros *default* definidos pelo Weka. Para o ML-KNN utilizou-se número de vizinhos igual a 3, uma vez que o conjunto de dados é pequeno.

Com relação ao ML-OCS, gerou-se uma população inicial contendo 100 regras, com probabilidade de inserção do símbolo "don't care"igual a 0.7 por atributo. O intuito foi o de gerar uma população inicial bastante geral. Cada regra é colocada em sua própria organização com LTR e STR inicializados como 1, RE como $0,5$ e PF como 0. O número N de últimas recompensas armazenadas para cada organização foi fixado como 40 e todos os N valores inicializados com 1. Utilizaram-se os seguintes valores para as constantes de atualização de classificadores e organizações: $C = 0,02$, $M = 1,15$, $F = 0,01$, $T_O = 0,01$ e $T_c = 0,01$.

No que diz respeito ao OGC, o tempo entre aplicações desse componente foi fixado em 3 vezes o tamanho do conjunto de treinamento. A porcentagem da performance média de uma organização utilizada pelos operadores Grow e Shrink foi de 85%. O operador Shrink retira classificadores de suas organizações colocando-os em organizações recém criadas seguindo a mesma inicialização de parâmetros utilizada na geração da população inicial, com a exceção do LTR que é inicializado como 0.6, pois uma vez que esse classificador deixou sua organização por ter tido desempenho abaixo da média não faz sentido inicializar o LTR de sua nova organização como se essa fosse uma organização boa. Para manter a consistência, inicializam-se as N últimas recompensas dessa nova organização também com valor $0,6$ cada. O RE do classificador é alterado para $0,5$ e seu PF para 0.

Para que o AG efetue o cálculo do desvio padrão de cada regra, o algoritmo armazenou as 30 últimas recompensas recebidas por cada uma das regras na população. O RE da regra é considerado no cálculo do desvio padrão como o valor de média dessas últimas recompensas. No caso do algoritmo selecionar o operador MA a probabilidade de ocorrência de mutação é de $0,3$ por atributo. Caso o contrário ocorra, ou seja, o AG selecione o operador MC, verifica-se o RE da regra antes da aplicação do operador, como explicado na Seção 4.4. Nesse caso, uma regra é considerada altamente recompensada caso seu RE seja no mínimo $0,9$. No que diz respeito ao tempo de aplicação do AG, calculou-se $TimeA$ utilizando as fórmulas apresentadas na Seção 4.4, com $\alpha = 0,7$, $c = 0,3$ e $Nexemplos = 30$. A variável $Nexemplos$ foi inicializada com o valor aproximado do tamanho do conjunto de treinamento, uma vez que esse valor pode sofrer pequenas alterações dependendo da maneira como o conjunto for particionado em *folds*. Para $TimeB$ utilizou-se $y = 15$ e $z = 40$.

O critério de parada foi estabelecido como atingir acurácia no mínimo 1 no conjunto de validação por 10 vezes consecutivas, ou atingir 350000 iterações. O mínimo valor de acurácia foi estabelecido baseando-se na acurácia máxima obtida pelos três outros algoritmos nesse mesmo conjunto de dados.

Os resultados dos experimentos podem ser observados na Tabela 5.2. Os valores entre parênteses representam o desvio padrão.

Método	Hamming Loss	Acurácia	Precisão	Revocação
BR	0,000 (0,000)	1,000 (0,000)	1,000 (0,000)	1,000 (0,000)
LP	0,000 (0,000)	1,000 (0,000)	1,000 (0,000)	1,000 (0,000)
ML-KNN	0,053 (0,020)	0,947 (0,020)	0,948 (0,018)	0,990 (0,015)
ML-OCS	0,010 (0,013)	0,991 (0,013)	0,995 (0,008)	0,991 (0,014)

Tabela 5.2: Médias e desvios padrão de 10 experimentos utilizando diferentes métodos de classificação

Para avaliar a relevância dos resultados obtidos, aplicou-se o teste estatístico t-student para dados pareados corrigido. Para todos os testes, utilizou-se um nível de confiança de 5% e a hipótese nula de que não existe diferença estatística entre os resultados obtidos pelos métodos comparados. O teste comparando ML-OCS e BR resultou em aceitação da hipótese nula para todas as quatro métricas, demonstrando não haver diferença estatística entre os dois métodos. Como os resultados obtidos por BR e por LP foram exatamente iguais, a mesma conclusão se aplica para o teste que compara ML-OCS e LP. Em seguida, comparou-se o ML-OCS com ML-KNN. O teste rejeitou a hipótese nula para as métricas Hamming Loss, Acurácia e Precisão. Isso significa que existe diferença estatística entre os dois métodos nessas três métricas, com melhor desempenho do ML-OCS. Para a métrica Revocação, a hipótese nula foi aceita, demonstrando que os dois algoritmos alcançaram desempenho similar nessa métrica.

5.4.2 Conjunto de Dados Anti-oncogene Apoptosis

Para o conjunto Anti-oncogene Apoptosis também foram conduzidos 10 experimentos para cada um dos 4 métodos comparados. Foram utilizados 6 folds para a realização do *cross-validation* em cada um dos experimentos. A configuração dos métodos BR, LP e ML-KNN utilizadas foram as mesmas do experimento com a base artificial.

Todos os parâmetros do algoritmo ML-OCS foram mantidos em relação aos experimentos com a base artificial com exceção dos seguintes:

- Utilizou-se uma população inicial contendo 400 regras;
- O tempo entre aplicações do OGC foi fixado em 2 vezes o tamanho do conjunto de treinamento, uma vez que esse é maior em relação ao conjunto artificial;
- O tempo entre aplicações do AG também sofreu alterações. Para o cálculo de $TimeA$ utilizou-se $Nexemplos = 91$ e para $TimeB$ utilizou-se $y = 7$ e $z = 20$;
- O critério de parada foi estabelecido em um mínimo de acurácia de 70% no conjunto de validação por 10 vezes consecutivas, ou atingir 350000 iterações. O valor de 70% foi fixo após realizar os experimentos com os três outros algoritmos e obter o desempenho máximo desses no mesmo critério.

A Tabela 5.3 apresenta os resultados dos experimentos.

Método	Hamming Loss	Acurácia	Precisão	Revocação
BR	0,290 (0,004)	0,710 (0,004)	0,732 (0,004)	0,710 (0,004)
LP	0,290 (0,004)	0,710 (0,004)	0,732 (0,004)	0,710 (0,004)
ML-KNN	0,286 (0,008)	0,711 (0,015)	0,732 (0,015)	0,712 (0,015)
ML-OCS	0,317 (0,024)	0,686 (0,025)	0,705 (0,025)	0,690 (0,024)

Tabela 5.3: Médias e desvios padrão de 10 experimentos utilizando diferentes métodos de classificação

Aplicando-se o teste estatístico com nível de confiança 5% e hipótese nula de que não existe diferença estatística entre os resultados comparados, obtiveram-se os seguintes resultados. O teste comparando BR e ML-OCS resultou em aceitação da hipótese nula em todas as métricas, indicando não haver diferença estatística entre os dois métodos. O teste também não encontrou diferenças estatísticas entre os métodos LP e ML-OCS. A comparação entre ML-KNN e ML-OCS resultou em aceitação da hipótese nula para as métricas Acurácia, Precisão e Revocação. O teste rejeitou a hipótese nula para a métrica Hamming Loss, indicando diferença estatística entre os resultados dos dois métodos, com melhor desempenho do ML-KNN.

5.4.3 Conjunto de Dados Cell-cycle Cell-division

Para o conjunto de dados Cell-cycle Cell-division foram conduzidos 10 experimentos para cada um dos 4 métodos de classificação utilizados, utilizando o *k-fold cross validation* estratificado com $k = 6$. As configurações dos métodos BR, LP e ML-KNN foram as mesmas utilizadas para os dois conjuntos de dados anteriores.

Para o ML-OCS, foram mantidos os mesmos parâmetros utilizados nos experimentos com a base artificial, com exceção dos seguintes:

- Utilizou-se uma população inicial contendo 500 regras;
- O tempo entre aplicações do OGC foi fixado em 2 vezes o tamanho do conjunto de treinamento, assim como nos experimentos utilizando a base Anti-oncogene Apoptosis;
- Com relação aos parâmetros do AG, fixou-se $N_{exemplos} = 96$ para o cálculo de $TimeA$ e para calcular $TimeB$ utilizaram-se $y = 10$ e $z = 20$;
- O critério de parada estabelecido foi o de alcançar no mínimo 60% de acurácia no conjunto de validação por 10 vezes consecutivas, ou atingir 350000 iterações. Assim como nos experimentos com os conjuntos de dados anteriores, o valor 60% foi estabelecido após a realização dos experimentos com os três outros algoritmos para esse conjunto de dados.

Método	Hamming Loss	Acurácia	Precisão	Revocação
BR	0,432 (0,008)	0,567 (0,008)	0,641 (0,015)	0,706 (0,054)
LP	0,407 (0,012)	0,593 (0,0125)	0,686 (0,011)	0,604 (0,009)
ML-KNN	0,412 (0,022)	0,587 (0,022)	0,640 (0,020)	0,745 (0,036)
ML-OCS	0,420 (0,022)	0,573 (0,020)	0,641 (0,023)	0,713 (0,025)

Tabela 5.4: Médias e desvios padrão de 10 experimentos utilizando diferentes métodos de classificação

Os resultados dos experimentos podem ser observados na Tabela 5.4.

Assim como para os dois conjuntos de dados anteriores, aplicou-se o teste estatístico t-student para dados pareados corrigido, com nível de confiança 5%, para comparar os resultados obtidos por ML-OCS em relação a BR, LP e ML-KNN. A hipótese nula utilizada é a de que não existe diferença estatística entre os métodos comparados. O teste comparando ML-OCS e BR resultou em aceitação da hipótese nula para todas as métricas utilizadas, demonstrando não haver diferença estatística entre os resultados obtidos pelos dois métodos. O resultado do teste entre ML-OCS e LP indicou haver diferença estatística entre os dois métodos para as métricas Precisão e Revocação. Na métrica Precisão, o método LP alcançou melhores resultados que o ML-OCS, enquanto que na métrica Revocação, o inverso ocorre. Por fim, o teste comparando ML-OCS e ML-KNN demonstrou não haver diferença estatística significativa entre os dois métodos, com aceitação da hipótese nula em todas as métricas utilizadas.

5.4.4 Análise de Regras Geradas

Uma vez que o ML-OCS é um classificador baseado em regras, é interessante analisar o conjunto de regras gerado pelo algoritmo ao final de sua execução. Nessa sub-seção, realiza-se uma análise inicial do conjunto solução gerado pelo ML-OCS utilizando o conjunto de dados artificial. A escolha do conjunto de dados se deu por este possuir menor número de atributos em seu antecedente, e portanto, as regras para seus exemplos são sintaticamente menores e mais fáceis de serem analisadas por um ser humano.

São apresentados aqui os conjuntos solução obtidos pelo ML-OCS em cada um dos folds de um experimento que realiza *5-fold cross-validation*. Todos os parâmetros do algoritmo são idênticos aos dos experimentos apresentados em 5.4.1.

As Tabelas 5.5, 5.6, 5.7, 5.8 e 5.9 apresentam as organizações, e suas respectivas regras, utilizadas para classificar respectivamente o primeiro, segundo, terceiro, quarto e quinto *fold* de teste.

Na situação representada na Tabela 5.5 nota-se que o algoritmo obteve um conjunto de regras bastante próximo de um conjunto ideal. Para a obtenção do conjunto ideal, basta que haja a generalização do terceiro bit da terceira regra da organização 1. Caso isso acontecesse, a quarta regra dessa organização não seria mais necessária e logo deixaria de ser utilizada. A organização 2 possui apenas uma regra, que é uma cópia de uma regra da organização 1. Casos como esse são

Organização	Regra	RE	PF	Bid	LTR
1	1,#,#,#,#,# -> 0;1	0,999	1,508	1,508	1,000
1	#,1,#,#,#,# -> 1;0	0,999	1,505	1,505	1,000
1	0,0,1,#,#,# -> 1;1	0,999	1,913	1,913	1,000
1	#,0,#,#,0,# -> 1;1	0,999	0,9601	0,960	1,000
2	1,#,#,#,#,# -> 0;1	0,999	0,000	0,000	1,000

Tabela 5.5: Organizações e regras obtidas - Primeiro fold

esperados uma vez que o LTR das organizações é igual, e, obviamente, nenhuma é mais geral que a outra.

Organização	Regra	RE	PF	Bid	LTR
1	1,#,#,#,#,# -> 0;1	0,999	0,132	0,132	1,000
1	#,1,#,#,#,# -> 1;0	0,999	0,000	0,000	1,000
1	0,0,#,#,1,# -> 1;1	0,999	0,384	0,384	1,000
1	#,0,#,#,#,# -> 1;1	0,999	0,000	0,000	1,000

Tabela 5.6: Organizações e regras obtidas - Segundo fold

Para o conjunto de regras gerado usando o segundo *fold* como *fold* de teste, apresentado na Tabela 5.6, nota-se que a solução criada foi parecida com a obtida no primeiro *fold*. Observe que, novamente, é preciso que ocorra a generalização de um bit da terceira regra para que a quarta regra deixe de ser necessária.

Organização	Regra	RE	PF	Bid	LTR
1	0,1,#,#,#,# -> 1;0	0,999	0,000	0,000	1,000
2	1,#,#,#,#,# -> 0;1	0,999	0,000	0,000	1,000
2	0,0,#,#,#,# -> 1;1	0,999	0,110	0,110	1,000

Tabela 5.7: Organizações e regras obtidas - Terceiro fold

Nas terceira e quarta execuções do algoritmo, utilizando respectivamente o terceiro e o quarto *fold* como *fold* de teste, obteve-se um conjunto de regras com o menor número de regras possível, como pode ser observado nas Tabelas 5.7 e 5.8.

Organização	Regra	RE	PF	Bid	LTR
1	1,#,#,#,#,# -> 0;1	0,999	0,010	0,010	1,000
1	0,1,#,#,#,# -> 1;0	0,999	0,029	0,029	1,000
1	0,0,#,#,#,# -> 1;1	0,999	0,000	0,000	1,000

Tabela 5.8: Organizações e regras obtidas - Quarto fold

A Tabela 5.9 apresenta as regras obtidas ao utilizar o quinto *fold* na fase de teste.

Organização	Regra	RE	PF	Bid	LTR
1	#,#,#,#,#,# -> 0;1	0,999	-1,318	-1,320	1,000
1	0,#,#,#,#,# -> 1;0	0,999	-0,321	-0,321	1,000
1	1,#,#,#,#,1 -> 0;1	0,999	0,000	0,000	1,000
1	0,0,#,#,#,# -> 1;1	0,999	0,029	0,029	1,000

Tabela 5.9: Organizações e regras obtidas - Quinto fold

O algoritmo demonstrou ser capaz de construir um conjunto solução com o menor número de regras possível e, quando não é capaz, aproxima-se satisfatoriamente do menor número de regras. Essa característica observada é bastante interessante pois indica que, pelo menos para um problema de baixa dificuldade, o ML-OCS produz conjuntos de regras de fácil leitura, com pequeno número de regras que, por sua vez, possuem poucas condições em seu antecedente.

5.5 Considerações Finais

Nesse capítulo foram apresentados a metodologia para condução dos experimentos envolvendo o método proposto para classificação multirrótulo e os resultados obtidos nos experimentos realizados. Foram utilizados três conjuntos de dados, um artificial e dois reais, para comparar o desempenho obtido pelo ML-OCS com outros três métodos de classificação multirrótulo existentes na literatura. Os resultados obtidos por meio da realização de experimentos com todos os algoritmos foram em seguida comparados utilizando um teste estatístico. Além disso, uma pequena análise das regras geradas pelo ML-OCS foi realizada utilizando o conjunto de dados artificial.

Os experimentos comparando o ML-OCS com os outros algoritmos demonstraram o potencial do método proposto. Os resultados do teste estatístico demonstraram que para o conjunto de dados artificial não existe diferença entre os resultados obtidos pelo ML-OCS e pelos métodos BR e LP, sendo esses últimos os que alcançaram melhor desempenho em todas as métricas utilizadas. O teste também mostrou que o ML-OCS obteve melhor desempenho que o ML-KNN em três métricas (Acurácia, Precisão e Revocação) e, na métrica Hamming Loss, os algoritmos alcançaram resultados similares.

Para o conjunto de dados Anti-oncogene Apoptosis, o teste demonstrou não haver diferença entre ML-OCS e os métodos BR e LP em todas as quatro métricas. No comparação do ML-OCS com o ML-KNN, o teste indicou diferença estatística com melhor desempenho do ML-KNN em apenas uma métrica (Hamming Loss).

Finalmente, para o conjunto de dados Cell-cycle Cell-division, o teste estatístico demonstrou que não existe diferença entre os resultados obtidos pelo ML-OCS e os métodos BR e ML-KNN em todas as métricas utilizadas. A comparação entre ML-OCS e LP indicou a existência de diferença estatística significativa apenas para a métrica Revocação, com melhor resultado obtido pelo ML-OCS.

Esses resultados demonstram o potencial do ML-OCS em alcançar o seu objetivo, que é obter desempenho no mínimo similar a outros métodos de classificação multirrótulo. O ML-OCS não apenas alcançou desempenho similar ao de outros três métodos como em um dos conjuntos de dados obteve melhor desempenho que um deles em três métricas distintas.

Por fim, realizou-se uma análise inicial do conjunto de regras gerado pelo ML-OCS. Para isso, utilizou-se o conjunto de dados artificial por ser o conjunto com menor número de atributos. Os conjuntos de regras gerados para esse conjunto mostraram que o ML-OCS tem potencial não somente para alcançar desempenho similar a outros métodos, mas também para gerar conjuntos solução de fácil análise e interpretação.

CAPÍTULO

6

Conclusão

Essa dissertação apresentou um estudo sobre LCS para classificação multirrótulo. Classificação multirrótulo é um tipo de classificação em que os exemplos podem possuir não apenas um rótulo de classe, mas um conjunto de rótulos. A possível existência de mais de um rótulo de classe para cada exemplo torna a tarefa de classificação mais complexa. A principal motivação para os problemas multirrótulo surgiu em problemas de categorização de textos e, atualmente, as técnicas de classificação multirrótulo vem sendo cada vez mais solicitadas por aplicações novas, como categorização de músicas, classificação de funções de proteínas e classificação semântica de cenas.

Os métodos existentes para classificação multirrótulo podem ser agrupados em duas categorias principais: métodos independentes de algoritmo, também chamados de métodos de transformação do problema, e métodos dependentes de algoritmo, também chamados de métodos de adaptação de algoritmos.

Para tentar solucionar problemas de classificação multirrótulo, utilizou-se nessa dissertação a teoria dos LCS. LCS são sistemas de aprendizado adaptativo que consistem em um sistema de produção de regras e um mecanismo de descoberta de novas regras. O sistema de produção de um LCS é um sistema baseado em regras que realiza inferências sobre o problema que se deseja solucionar. Já o mecanismo de descoberta de novas regras utiliza um AG para buscar por novas regras baseado nas regras atuais. Desde a sua proposta em 1971 por Holland, muitas pesquisas foram realizadas na área dos LCS com aplicações tanto em classificação de dados quanto em tarefas de aprendizado por reforço. Os LCS representam seu conhecimento por meio de regras de decisão. Baseado no tipo de representação de um indivíduo, existem duas categorias principais em que se dividem os LCS: Michigan e Pittsburgh. Os LCS que seguem a abordagem Michigan representam um indivíduo como um único classificador, enquanto que os que seguem a abordagem

Pittsburgh representam um indivíduo como um conjunto de regras completo. Ambas as abordagens possuem vantagens e desvantagens.

Embora Michigan e Pittsburgh sejam as abordagens mais conhecidas e utilizadas, existe ainda uma terceira abordagem proposta por Wilcox. Wilcox propôs um conceito para um sistema classificador capaz de exibir comportamento coletivo e individual ao mesmo tempo. O classificador criado por Wilcox recebeu a denominação de Organizational Classifier System (OCS). O OCS evolui grupos de classificadores, em que esses grupos podem variar em tamanho e podem também interagir entre si trocando classificadores.

Para alcançar o objetivo de realizar classificação multirrótulo utilizando um LCS, esse trabalho uniu a teoria do OCS com a teoria das hierarquias default. Hierarquias default são conjuntos de regras organizadas em camadas em que regras padrão cobrem uma parte das respostas do sistema enquanto que as regras mais específicas são acionadas quando as regras padrão são incorretas. Para favorecer a formação adaptativa das hierarquias default, utilizou-se o método proposto por Smith, baseado em leilão de necessidades e diferentes fatores de prioridade.

O algoritmo desenvolvido, denominado de Multi-Label OCS, trabalha sobre o conceito de formação de organizações proposto pelo OCS e sobre a representação de problemas multirrótulo como uma hierarquia default. A diferença básica entre o sistema de produção de regras do ML-OCS e do OCS é que o ML-OCS utiliza medidas de qualidade de suas regras de modo a favorecer a formação das hierarquias default. O algoritmo procura unir os classificadores que apresentam desempenho similar durante o treinamento em uma mesma organização, de modo a separar regras corretas de parcialmente corretas e incorretas. Para realizar essa separação de classificadores em diferentes organizações, o algoritmo utiliza um Mecanismo de Crescimento Organizacional, que foi baseado no respectivo componente do OCS. Esse mecanismo utiliza os operadores Grow e Shrink de maneira aleatória para respectivamente, adicionar um classificador em uma organização e retirar classificadores de uma organização. Embora esse mecanismo seja fortemente baseado no mecanismo do OCS, várias modificações foram realizadas de maneira a adaptar os operadores para a realização de trocas menos aleatórias de classificadores entre organizações.

Para a inserção de novas regras no sistema, o algoritmo conta com o auxílio de um AG como mecanismo de busca por soluções. O AG do ML-OCS foi criado visando a particularidade do problema de classificação multirrótulo, que é a existência de uma terceira categoria de regras, as parcialmente corretas. Essas regras recebem recompensas parciais do sistema, porém somente o seu RE não é suficiente para determinar se realmente se trata de uma regra parcialmente correta, ou seja, que necessita de alteração em seu consequente. Regras que precisam ser especializadas, possuindo o consequente correto mas cobrindo uma região do espaço de soluções maior do que deveriam, também recebem recompensas parciais. Por meio do desvio padrão das recompensas recebidas pela regra, o AG tenta inferir quais os tipos das regras que foram selecionadas por seu mecanismo de seleção, escolhendo então qual alternativa seguir, se uma mutação no antecedente da regra ou em seu consequente. As novas regras criadas são inseridas na população no lugar de duas regras consideradas ruins pertencentes ao Match-set atual.

Para a fase de teste, o ML-OCS utiliza o mecanismo do sistema de produção de regras responsável pela resolução de conflitos para escolher a regra que classificará o exemplo de entrada. A principal diferença com relação a fase de treinamento é que, no teste, o algoritmo não escolhe aleatoriamente entre duas organizações na ocorrência de empate entre uma ou mais. Ao ocorrer empate, o ML-OCS escolherá a organização que possua a regra mais geral para classificação do exemplo, em uma tentativa de aumentar a capacidade de generalização do algoritmo. É importante ressaltar que, nesse trabalho, o ML-OCS não conta, na fase de teste, com um mecanismo que considere mais de uma regra para a classificação de um exemplo, ou seja, para um exemplo com mais de um rótulo de classe, se o algoritmo não possuir uma regra que contenha todas as classes em conjunto, ele não será capaz de inferir o conjunto total de regras do exemplo de entrada a partir da união de várias regras que possuam um subconjunto do conjunto total de rótulos.

O ML-OCS foi testado em três conjuntos de dados binários distintos. O primeiro deles foi gerado a partir de um conjunto de regras, os dois seguintes são conjuntos de dados de Bioinformática. Para comparar o desempenho do ML-OCS, foram utilizados os métodos para classificação multirrótulo independentes de algoritmo Binary Relevance e Label Power-Set, e o método dependente de algoritmo ML-KNN. Todos os algoritmos foram executados 10 vezes para cada conjunto de dados, sendo que cada execução consiste em realizar uma validação cruzada por meio do método *k-fold cross validation* estratificado. Quatro métricas foram consideradas para avaliar o desempenho dos algoritmos: Hamming Loss, Acurácia, Precisão e Revocação. Após a realização dos experimentos, os resultados obtidos foram comparados por meio do teste estatístico t-student para dados pareados corrigido. Os resultados demonstraram que o ML-OCS alcançou, na maior parte das vezes, desempenho similar aos outros métodos.

Com relação ao ML-OCS, além dos testes comparativos, foi realizada uma pequena análise de seu conjunto solução. Essa análise inicial em um conjunto de dados pequeno revelou que o ML-OCS é capaz de organizar as regras que obteve durante o treinamento em um número reduzido de organizações, quando não em apenas uma organização. Além disso, o número de regras no conjunto solução foi sempre muito próximo do número mínimo de regras para solucionar o problema de classificação.

A principal contribuição desse trabalho foi a proposta de um LCS para classificação multirrótulo. Os LCS tem sido bastante utilizados e seus resultados são promissores, entretanto não se tem conhecimento de trabalhos prévios utilizando LCS para problemas multirrótulo. Os resultados alcançados foram bastante promissores, indicando que o método criado foi competitivo com demais métodos existentes na literatura. Além disso, a primeira etapa de criação do novo algoritmo gerou uma publicação (Vallim et al., 2008) que contém os resultados preliminares obtidos e que estão também reportados no Apêndice A dessa dissertação. Os resultados reportados nessa dissertação também serão publicados nos anais da *Genetic and Evolutionary Computation Conferecence 2009* (GECCO, 2009) (Vallim et al., 2009), como artigo completo na área de LCS.

Uma outra contribuição importante foi a pesquisa bibliográfica realizada, principalmente na área de LCS, reunindo os conceitos principais e os algoritmos mais utilizados.

Muito pode ser realizado em pesquisas futuras considerando o ML-OCS. Em sua versão original, o algoritmo é capaz de trabalhar apenas com conjuntos de dados binários. Assim, uma primeira direção para trabalhos futuros seria a adaptação do algoritmo para trabalhar com diferentes representações de regras, como, por exemplo, aceitando valores de atributos inteiros e reais. Uma análise sobre o tempo entre aplicações do componente de crescimento organizacional e a maneira como esse afeta o sistema também pode ser útil. Além disso, é interessante realizar um estudo sobre os operadores que alteram as organizações, de modo a desenvolver operadores mais eficazes. Ainda sobre o componente de crescimento organizacional, esse está diretamente ligado ao tamanho da população, ou seja, quanto maior a população, mais tempo o algoritmo levará para aplicar esse componente. Esse tempo, contudo, tende a diminuir durante a execução, uma vez que o número de organizações começa a diminuir com a inclusão de vários classificadores em uma mesma organização. Assim, uma possível direção futura é a realização de um re-desenho do componente de crescimento organizacional de modo a reduzir sua complexidade.

Uma vez que o OCS não contém um mecanismo de descoberta de novas regras, o AG criado para o ML-OCS é um dos pontos que podem ser abordados em trabalhos futuros. Seria interessante a inserção de um operador de *crossover* no algoritmo. Outro fator que merece estudos mais detalhados é a forma como se calcula o tempo entre aplicações do AG. Neste trabalho, esse tempo é calculado de maneira bastante conservadora, considerando sempre o pior caso que é o do AG inserir um filho ruim na população resultando em uma queda temporária de desempenho. Uma melhor análise de quanto tempo o ML-OCS necessita para alocar esses novos classificadores em organizações adequadas (o que está diretamente ligado ao componente de crescimento organizacional) pode levar a redução do tempo entre aplicações do AG, e consequente redução na complexidade do algoritmo.

Como mencionado, o ML-OCS não é capaz de unir regras para classificar um novo exemplo na fase de teste. Alterar a maneira como o algoritmo se comporta na fase de teste para incluir essa possibilidade poderia também ser objeto de estudos futuros, podendo inclusive trazer uma melhora no desempenho preditivo do modelo de classificação gerado.

É importante ressaltar que, embora o ML-OCS tenha alcançado resultados bastante promissores, se aproximando dos melhores resultados obtidos por outros métodos, o método proposto nesse trabalho ainda sofre com a quantidade de parâmetros que devem ser inicializados e com a complexidade do sistema como um todo. Os métodos utilizados para comparação possuem bem menos parâmetros e terminam sua execução em menor tempo que o ML-OCS. A principal dificuldade do ML-OCS é a interação adequada entre o AG e o OGC, uma vez que novas regras inseridas na população pelo AG precisam ser suficientemente avaliadas antes de sofrerem a aplicação dos operadores do OGC. Observou-se que não respeitar essa condição leva ao consequente colapso da população de regras, não chegando a uma solução satisfatória.

Referências Bibliográficas

AIOLLI, F.; PORTERA, F.; SPERDUTI, A. Speeding up the solution of multilabel problems with Support Vector Machines. In: *Supplementary Proc. of the Joint 13th International Conference on Artificial Neural Networks and 10th International Conference on Neural Information Processing*, 2003, p. 118–121.

BACARDIT, J.; BUTZ, M. V. Data mining in learning classifier systems: Comparing xcs with gassist. In: *Seventh International Workshop on Learning Classifier Systems (IWLCS- 2004)*, 2004.

BACARDIT, J.; GARRELL, J. M. Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. *Lecture Notes in Computer Science: Learning Classifier Systems*, v. 4399, p. 59–79, 2007.

BACK, T. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

BACK, T.; FOGEL, D.; MICHALEWICZ, Z. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, 2000.

BAECK, T.; FOGEL, D.; MICHALEWICZ, Z. *Evolutionary Computation 2: Advanced Algorithms and Operators*. Institute of Physics Publishing, 2000.

BEASLEY, D. Possible applications of evolutionary computation. In: *Evolutionary Computation 1: Basic Algorithms and Operators*, Berlin, Heidelberg, and New York: Institute of Physics Publishing, p. 4–18, 2000.

BERNADÓ-MANSILLA, E.; GARRELL, J. M. MOLeCS: Using Multiobjective Evolutionary Algorithms for Learning. *Lecture Notes in Computer Science*, v. 1993, p. 696–710, 2001.

BERNADÓ-MANSILLA, E.; GARRELL, J. M. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation*, v. 11, n. 3, p. 209–238, 2003.

BERNADÓ-MANSILLA, E.; LLORÁ, X.; TRAUS, I. *Multiobjective Learning Classifier Systems: An Overview*. Relatório Técnico 2005020, university of Illinois at Urbana Champaign, Urbana, IL, 2005.

BONELLI, P.; PARODI, A.; SEN, S.; WILSON, S. W. NEWBOOLE: A fast GBML System. In: *Seventh International Conference on Machine Learning*, Morgan Kaufmann, 1990, p. 153–159.

BOUTELL, M.; LUO, J.; SHEN, X.; BROWN, C. Learning multi-label scene classification. *Pattern Recognition*, v. 37, n. 9, p. 1757–1771, 2004.

BUTZ, M. V. *Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction*. Tese de Doutoramento, Graduate College of the University of Illinois at Urbana-Champaign, Urbana, IL, 2004.

CASTRO, L. N.; TIMMIS, J. I. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer-Verlag, 2002.

CHAN, A.; FREITAS, A. A. A New Ant Colony Algorithm for Multi-Label Classification with Applications in Bioinformatics. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, 2006, p. 27–34.

CLARE, A.; KING, R. Knowledge discovery in multi-label phenotype data. In: *Proc. of the European Conf. on Principles of Data Mining and Knowledge Discovery*, 2001, p. 42–53.

CLARK, P.; BOSWELL, R. Rule induction with cn2: some recent improvements. In: *EWSL-91: Proc. of the Working Session on Learning on Machine Learning*, 1991.

COHEN, W. Fast effective rule induction. In: *Proc. of the 12th International Conference on Machine Learning*, 1995.

COSTA, E. P. *Investigação de Ténicas de Classificação Hierárquica para Problemas de Bioinformática*. Relatório Técnico, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2008.

DARWIN, C. *On the Origin of Species by Means of Natural Selection*. 1859.

DAVIS, L. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

DORIGO, M.; STÜTZLE, T. *Ant Colony Optimization*. MIT Press, 2004.

DUDA, R.; HART, P.; STORK, D. *Pattern Classification*. Wiley-Interscience, 2000.

ELISSEEFF, A.; WESTON, J. *Kernel methods for multi-labelled classification and categorical regression problems*. Relatório Técnico, BIOwulf Technologies, 2001.

FLOCKHART, I. W.; RADCLIFFE, N. J. *GA-MINER: parallel data mining with hierarchical genetic algorithms - final report*. Relatório Técnico EPCCAIKMS, University of Edinburgh, UK, 1995.

FREITAS, A. A. On objective measures of rule surprisingness. In: *Lecture Notes in Artificial Intelligence 1510: Principles of Data Mining and Knowledge Discovery (Proc. 2nd European Symp., PKDD´98, Nantes, France)*, Springer-Verlag, 1998, p. 1–9.

FREITAS, A. A. On Rule Interestingness Measures. *Knowledge-Based Systems*, v. 12, n. 5-6, p. 309–315, 1999.

FREITAS, A. A. A survey of evolutionary algorithms for data mining and knowledge discovery. *Advances in Evolutionary Computation*, p. 819–845, 2002.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, v. 55, p. 119–139, 1997.

GIORDANA, A.; NERI, F. Search-intensive concept induction. *Evolutionary Computation*, v. 3, n. 4, p. 375–416, 1995.

GODBOLE, S.; SARAWAGI, S. Discriminative Methods for Multi-labeled Classification. In: *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2004.

GOLDBERG, D.; DEB, K. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In: *Foundations of Genetic Algorithms*, 1991, p. 69–93.

GOLDBERG, D.; RICHARDSON, J. Genetic algorithms with sharing for multimodal function optimization. In: *Proceedings of the Second International Conference on Genetic Algorithms*, 1987, p. 41–49.

GOLDBERG, D.; SASTRY, K. A practical schema theorem for genetic algorithm design and tuning. In: *Proceedings of the Third Genetic and Evolutionary Computation Conference*, 2001, p. 328–335.

GOLDBERG, D. E. *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

GOLDBERG, D. E. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 2002.

GREENE, D. P.; SMITH, S. F. Competition-based induction of decision models from examples. *Machine Learning*, v. 13, p. 229–257, 1993.

HARIK, G. *Finding multiple solutions in problems of bounded dificulty*. Relatório Técnico 94002, University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory, Urbana, IL, 1994.

HAYKIN, S. *Neural Networks - A Compreensive Foundation*. New Jersey: Prentice-Hall, 1999.

HOLDEN, N.; A., A. A hybrid PSO/ACO algorithm for classification. In: *Proc. of the GECCO-2007 Workshop on Particle Swarms: The Second Decade*, ACM Press, 2007, p. 2745–2750.

HOLLAND, J. H. Processing and processors for schemata. *Associative information processing*, p. 127–146, 1971.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. The MIT Press, 1975.

HOLLAND, J. H.; REITMAN, J. S. Cognitive systems based on adaptive algorithms. *Pattern directed inference systems*, p. 313–329, 1978.

HOLMES, J. H. Discovering Risk of Disease with a Learning Classifier System. In: *Proceedings of the Seventh International Conference of Genetic Algorithms (ICGA97)*, Morgan Kaufmann, 1997, p. 426–433.

HORN, J.; GOLDBERG, D.; DEB, K. Implicit niching in a learning classifier system: Nature's way. *Evolutionary Computation*, v. 2, n. 1, p. 37–66, 1994.

HSU, C.-W.; LIN, C.-J. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, v. 13, n. 2, p. 415–425, 2002.

JANIKOW, C. *Inductive Learning of Decision Rules in Attribute-Based Examples: a Knowledge-Intensive Genetic Algorithm Approach*. Tese de Doutoramento, University of North Carolina at Chapel Hill, 1991.

JANIKOW, C. Z. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, v. 13, p. 189–228, 1993.

JONG, K. A. D. *An analysis of the behavior of a class of genetic adaptive systems*. Tese de Doutoramento, University of Michigan, Ann Arbor, 1975.

JONG, K. A. D.; SPEARS, W. M. Learning Concept Classification Rules Using Genetic Algorithms. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, 1991, p. 651–656.

JONG, K. A. D.; SPEARS, W. M.; GORDON, D. F. Using genetic algorithms for concept learning. *Machine Learning*, v. 13, p. 161–188, 1993.

JONG, K. D. *Evolutionary Computation: A Unified Approach*. The MIT Press, 2006.

KARALIC, A.; PIRNAT, V. Significance level based multiple tree classification. *Informatica*, v. 15, n. 5, 1991.

LACERDA, E. G. M.; CARVALHO, A. C. P. L. F.; LUDERMIR, T. B. Um Tutorial sobre Algoritmos Geneticos. *RITA*, v. 4, p. 110–139, 1999.

LANGDON, W. B.; POLI, R. Fitness causes bloat: Mutation. In: *Genetic Programming: First European Conference*, 1998, p. 37–48.

LIU, B.; HSU, W.; CHEN, S. Using general impressions to analyze discovered classification rules. In: *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining*, AAAI Press, 1997, p. 31–36.

LLORÀ, X. *Genetic Based Machine Learning using Fine-grained Parallelism for Data Mining*. Tese de Doutoramento, Enginyeria i Arquitectura La Salle. Ramon Llull University, Barcelona, Catalonia, European Union, 2002.

LLORÀ, X.; GOLDBERG, D. E.; TRAUS, I.; BERNADÓ-MANSILLA, E. Accuracy, Parsimony, and Generality in Evolutionary Learning Systems via Multiobjective Selection. *Lecture Notes in Artificial Intelligence*, v. 2661, p. 118–142, 2003.

LORENA, A. C.; CARVALHO, A. C. P. L. F. *Introdução às Máquinas de Vetores Suporte*. Relatório Técnico 192, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, SP, 2003.

MAYR, E. *Toward a New Philosophy of Biology: Observations of an Evolutionist*. Harvard University Press, 1987.

MENDEL, G. Experiments in Plant Hybridization. In: *Meetings of the Brünn Natural History Society*, 1865.

MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1994.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre Aprendizado de Máquina. In: *Sistemas Inteligentes: Fundamentos e Aplicações*, 2003.

NADEAU, C.; BENGIO, Y. Inference for the Generalization Error. *Machine Learning*, v. 52, p. 239–281, 2003.

PAPPA, G. L.; FREITAS, A. A. Automatically Evolving Rule Induction Algorithms. In: *Proceedings of 17th European Conference on Machine Learning*, 2006, p. 341–352.

PARPINELLI, R. S.; LOPES, H. S.; FREITAS, A. A. Data Mining with an Ant Colony Optimization Algorithm. *IEEE Trans. On Evolutionary Computation*, v. 6, n. 4, p. 321–332, 2002.

QUINLAN, J. R. Induction of decision trees. In: SHAVLIK, J. W.; DIETTERICH, T. G., eds. *Readings in Machine Learning*, Morgan Kaufmann, originally published in *Machine Learning* 1:81–106, 1986, 1990.

QUINLAN, J. R. C4.5: Programs for Machine Learning. *Machine Learning*, v. 16, p. 235–240, 1993.

R DEVELOPMENT CORE TEAM *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0, 2008.
DisponÌvel em: `http://www.R-project.org`

RISSANEN, J. Modeling by shortest data description. *Automatica*, v. 14, p. 465–471, 1993.

SCHAPIRE, R.; SINGER, Y. Boostexter: A boosting-based system for text categorization. *Machine Learning*, v. 39, n. 2-3, p. 135–168, 2000.

SHEN, X.; BOUTELL, M.; LUO, J.; BROWN, C. Multi-label machine learning and its application to semantic scene classification. In: *Proceedings of the SPIE*, 2003, p. 188–199.

SMALDON, J.; FREITAS, A. A. A new version of the Ant-Miner algorithm discovering unordered rule sets. In: *Proc. Genetic and Evolutionary Computation Conference (GECCO-2006)*, ACM Press, 2006, p. 43–50.

SMITH, R. E.; GOLDBERG, D. E. Reinforcement Learning with Classifier Systems: Adaptive Default Hierarchy Formation. *Applied Artificial Intelligence*, v. 6, p. 79–102, 1992.

SMITH, S. F. Flexible Learning of Problem Solving Heuristics through Adaptive Search. In: *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 1983, p. 422–425.

SMOLA, A. J.; BARTLETT, P. L.; SCHOLKOPF, B.; SCHUURMANS, D. *Advances in Large Margin Classifiers*. MIT Press, 1999.

TACKETT, W. A. Recombination, selection, and the genetic construction of computer programs, unpublished doctoral dissertation, University of Southern California, 1994.

TROHIDIS, K.; TSOUMAKAS, G.; KALLIRIS, G.; VLAHAVAS, I. Multilabel Classification of Music into Emotions. In: *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, Philadelphia, PA, USA, 2008.

TSOUMAKAS, G.; FRIBERG, R.; SPYROMITROS-XIOUFIS, E.; KATAKIS, I.; VILCEK, J. Multilabel Classification. 2007.
DisponÌvel em: `http://mlkd.csd.auth.gr/multilabel.html#Software`

TSOUMAKAS, G.; KATAKIS, I. Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining*, v. 3, p. 1–13, 2007.

VALLIM, R.; DUQUE, T.; GOLDBERG, D.; CARVALHO, A. The Multi-label OCS with a Genetic Algorithm for Rule Discovery: Implementation and First Results. In: *To appear at the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2009.

VALLIM, R.; GOLDBERG, D.; LLORA, X.; DUQUE, T.; CARVALHO, A. A New Approach for Multi-label Classification based on Default Hierarchies and Organizational Learning. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Workshop Session: Learning Classifier Systems*, 2008, p. 2017–2022.

WILCOX, J. R. *Organizational Learning Within a Learning Classifier System*. Relatório Técnico, Department of Computer Science, University of Illinois at Urbana-Champaign, illiGAL Report No, 95003, 1995.

WILSON, S. W. *Classifier System Learning of a Boolean Function*. Relatório Técnico 27r, The Rowland Institute for Science, 1986.

WILSON, S. W. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, v. 3, n. 2, p. 149–175, 1995.

WITTEN, I. H.; FRANK, E. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

ZHANG, M.; ZHOU, Z. ML-kNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, v. 40, n. 7, p. 2038–2048, 2007.

ZHANG, M.-L.; ZHOU, Z.-H. A k-nearest neighbor based algorithm for multi-label classification. In: *Proceedings of the 1st IEEE International Conference on Granular Computing (GrC'05))*, 2005, p. 718–721.

APÊNDICE

A

Testes Preliminares

Neste apêndice apresentam-se os testes preliminares para verificação da abordagem utilizada pelo algoritmo proposto. Estes testes foram realizados antes da inclusão do AG e portanto serviram para testar a capacidade do algoritmo em organizar as regras que possui.

Para a realização desse teste, utilizou-se um problema multirrótulo bastante simples definido na Figura A.1.

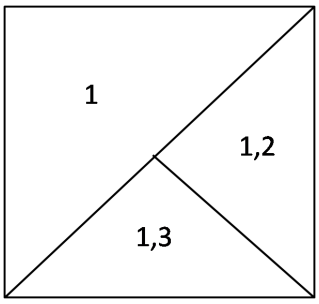

Figura A.1: Um problema multirrótulo simples

Nesse problema, alguns exemplos pertencem apenas a classe 1, enquanto outros exemplos pertencem as classes 1 e 2 ou 1 e 3 simultaneamente. As três regras a seguir, formando uma hierarquia default, são suficientes para representar o problema:

Antecedente/Consequente

```
00/1;2
01/1;3
##/1
```

As regras foram então codificadas de maneira ternária como representado a seguir.

Antecedente/Consequente

```
00/110
01/101
##/100
```

O consequente das regras possui um bit para cada classe do problema, com um valor 1 indicando a presença da classe correspondente e um valor 0 indicando a sua ausência.

O sistema foi provido das três regras corretas que representam o problema e mais quatro outras regras. Entre essas quatro regras, duas são erradas e duas são parcialmente corretas. As quatro regras extras apresentadas ao sistema são apresentadas a seguir.

Antecedente/Consequente

```
00/3
01/2
00/1;2;3
00/1
```

Como mencionado, o AG não foi utilizado nesse experimento de modo que não são adicionadas novas regras ao sistema e, portanto, o objetivo do sistema é aprender e organizar corretamente suas regras iniciais.

Inicialmente, cada regra foi colocada em sua própria organização, com $LT = 0,95$, $ST = 1$, $RE = 0,5$ e $PF = 0$. Todas as vezes que uma organização nova é criada pelo operador Shrink, os valores de LT e ST são inicializados com os mesmo valores das organizações iniciais, isso é, $0,95$ e 1, respectivamente. A porcentagem da performance média utilizada pelo operador Grow foi fixa em 85%. As constantes foram setadas em $C = 0,2$, $T_O = 0,01$, $F = 0,1$, $M = 1,5$. O tempo entre aplicações do componente de crescimento organizacional foi estabelecido em 30 ciclos, em que um ciclo é definido pela apresentação de todos os exemplos de treinamento ao sistema.

O algoritmo foi executado 30 vezes, cada vez com 20000 ciclos e, em 99% das vezes, terminou a execução com uma configuração similar a apresentada na Tabela A.1. A Tabela A.1 mostra os classificadores, as organizações em que foram alocados e os valores de RE e PF dos classificadores.

Como pode ser observado, o sistema separou os classificadores em 4 organizações. Os três classificadores corretos foram colocados juntos em uma mesma organização, os dois classificadores parcialmente corretos foram colocados em outra organização e os dois classificadores incorretos foram deixados sozinhos em suas próprias organizações. A Tabela A.2 apresenta os valores de LT de cada organização.

Pode-se observar que o sistema aprendeu a selecionar corretamente a melhor organização quando mais de uma pertence ao Math-set de organizações. Além disso, o sistema construiu a hierarquia default correta entre as regras contidas na organização ótima. Esses resultados demonstram que o sistema foi capaz de aprender o conjunto de regras ideal para resolver o problema multirrótulo e também aprendeu a estrutura hierárquica desse conjunto.

Classificador	Organização	Reward est.	Priority factor
00/001	1	0,000066	-0,131043
01/010	2	0,000083	-0,200000
00/110	3	0.999998	2.951449
01/101	3	0,999998	2,940476
##/100	3	0,999998	1,408944
00/111	4	0,665129	-0,066666
00/100	4	0,666665	-0,133332

Tabela A.1: Configuração do conjunto de regras ao final da execução

Organização	Reputação Long-Term
1	0,802632
2	0,500000
3	0,999987
4	0,833595

Tabela A.2: LTR de todas as organizações no final da execução

APÊNDICE

B

Resultados Detalhados

Neste apêndice, os resultados do Capítulo 5 são apresentados de maneira mais detalhada. Para cada conjunto de dados, são apresentados os resultados obtidos por cada um dos 10 experimentos realizados. Os resultados aqui apresentados consistem na média, para cada experimento, dos resultados obtidos segundo o processo *k-fold cross-validation* estratificado. Para cada conjunto de dados foi utilizado um número k de *folds* distinto. Para o conjunto artificial utilizou-se $k = 5$. Já para os conjuntos Anti-oncogene Apoptosis e Cell-Cycle Cell-division utilizou-se $k = 6$.

As Tabelas B.1 , B.2, B.3 e B.4 apresentam os resultados obtidos por cada algoritmo de aprendizado utilizado para o conjunto de dados artificial.

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,000	1,000	1,000	1,000
2	0,000	1,000	1,000	1,000
3	0,000	1,000	1,000	1,000
4	0,000	1,000	1,000	1,000
5	0,000	1,000	1,000	1,000
6	0,000	1,000	1,000	1,000
7	0,000	1,000	1,000	1,000
8	0,000	1,000	1,000	1,000
9	0,000	1,000	1,000	1,000
10	0,000	1,000	1,000	1,000

Tabela B.1: Resultados obtidos pelo método Binary-Relevance no conjunto de dados artificial

As Tabelas B.5, B.6, B.7 e B.8 apresentam os resultados obtidos pelos algoritmos no conjunto de dados Anti-oncogene Apoptosis.

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,000	1,000	1,000	1,000
2	0,000	1,000	1,000	1,000
3	0,000	1,000	1,000	1,000
4	0,000	1,000	1,000	1,000
5	0,000	1,000	1,000	1,000
6	0,000	1,000	1,000	1,000
7	0,000	1,000	1,000	1,000
8	0,000	1,000	1,000	1,000
9	0,000	1,000	1,000	1,000
10	0,000	1,000	1,000	1,000

Tabela B.2: Resultados obtidos pelo método Label Power-Set no conjunto de dados artificial

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,039	0,961	0,961	1,000
2	0,038	0,962	0,962	1,000
3	0,060	0,940	0,940	1,000
4	0,077	0,923	0,923	0,980
5	0,093	0,907	0,916	0,954
6	0,046	0,954	0,954	0,982
7	0,060	0,940	0,940	0,980
8	0,028	0,972	0,972	1,000
9	0,039	0,961	0,961	1,000
10	0,048	0,952	0,952	1,000

Tabela B.3: Resultados obtidos pelo método ML-KNN no conjunto de dados artificial

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,000	1,000	1,000	1,000
2	0,020	0,980	0,980	0,980
3	0,000	1,000	1,000	1,000
4	0,000	1,000	1,000	1,000
5	0,009	1,000	0,991	1,000
6	0,010	0,990	1,000	0,990
7	0,040	0,960	1,000	0,960
8	0,000	1,000	1,000	0,980
9	0,000	1,000	1,000	1,000
10	0,018	0,982	0,982	1,000

Tabela B.4: Resultados obtidos pelo método ML-OCS no conjunto de dados artificial

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,288	0,712	0,734	0,712
2	0,286	0,714	0,736	0,714
3	0,295	0,705	0,727	0,705
4	0,287	0,713	0,735	0,713
5	0,295	0,705	0,727	0,705
6	0,288	0,712	0,734	0,712
7	0,286	0,714	0,736	0,714
8	0,287	0,713	0,735	0,713
9	0,294	0,706	0,728	0,706
10	0,294	0,706	0,728	0,706

Tabela B.5: Resultados obtidos pelo método Binary-Relevance no conjunto de dados Anti-oncogene Apoptosis

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,288	0,712	0,734	0,712
2	0,286	0,714	0,736	0,714
3	0,295	0,705	0,727	0,705
4	0,287	0,713	0,735	0,713
5	0,295	0,705	0,727	0,705
6	0,288	0,712	0,734	0,712
7	0,286	0,714	0,736	0,714
8	0,287	0,713	0,735	0,713
9	0,294	0,706	0,728	0,706
10	0,294	0,706	0,728	0,706

Tabela B.6: Resultados obtidos pelo método Label Power-Set no conjunto de dados Anti-oncogene Apoptosis

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,291	0,708	0,731	0,712
2	0,271	0,729	0,750	0,729
3	0,295	0,694	0,712	0,694
4	0,280	0,720	0,742	0,720
5	0,280	0,719	0,742	0,719
6	0,280	0,719	0,741	0,719
7	0,286	0,714	0,736	0,721
8	0,283	0,713	0,735	0,713
9	0,301	0,677	0,699	0,677
10	0,287	0,713	0,735	0,720

Tabela B.7: Resultados obtidos pelo método ML-KNN no conjunto de dados Anti-oncogene Apoptosis

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,338	0,662	0,684	0,662
2	0,360	0,655	0,670	0,676
3	0,290	0,720	0,735	0,713
4	0,298	0,702	0,713	0,695
5	0,308	0,691	0,710	0,695
6	0,335	0,665	0,684	0,665
7	0,340	0,652	0,674	0,656
8	0,300	0,670	0,729	0,721
9	0,304	0,710	0,725	0,706
10	0,297	0,703	0,725	0,714

Tabela B.8: Resultados obtidos pelo método ML-OCS no conjunto de dados Anti-oncogene Apoptosis

As Tabelas B.9, B.10, B.11 e B.12 apresentam os resultados obtidos pelos algoritmos no conjunto de dados Cell-cycle Cell-division.

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,442	0,558	0,634	0,697
2	0,431	0,569	0,648	0,666
3	0,420	0,580	0,666	0,673
4	0,4314	0,569	0,637	0,711
5	0,427	0,573	0,653	0,719
6	0,441	0,559	0,642	0,656
7	0,420	0,579	0,642	0,718
8	0,438	0,562	0,614	0,844
9	0,434	0,565	0,649	0,660
10	0,440	0,560	0,622	0,719

Tabela B.9: Resultados obtidos pelo método Binary Relevance no conjunto de dados Cell-cycle Cell-division

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,407	0,593	0,686	0,596
2	0,386	0,614	0,704	0,621
3	0,413	0,587	0,680	0,597
4	0,404	0,596	0,689	0,610
5	0,400	0,600	0,697	0,607
6	0,434	0,566	0,663	0,594
7	0,410	0,589	0,686	0,600
8	0,397	0,603	0,693	0,614
9	0,411	0,589	0,679	0,599
10	0,409	0,590	0,680	0,601

Tabela B.10: Resultados obtidos pelo método Label Power-Set no conjunto de dados Cell-cycle Cell-division

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,431	0,569	0,637	0,745
2	0,448	0,552	0,603	0,722
3	0,430	0,570	0,639	0,704
4	0,407	0,593	0,627	0,828
5	0,434	0,566	0,621	0,744
6	0,382	0,618	0,666	0,753
7	0,406	0,583	0,649	0,719
8	0,403	0,596	0,652	0,717
9	0,383	0,616	0,669	0,740
10	0,396	0,604	0,635	0,777

Tabela B.11: Resultados obtidos pelo método ML-KNN no conjunto de dados Cell-cycle Cell-division

Experimento	Hamming Loss	Acurácia	Precisão	Revocação
1	0,403	0,576	0,656	0,757
2	0,424	0,573	0,628	0,722
3	0,417	0,569	0,639	0,691
4	0,430	0,555	0,642	0,691
5	0,427	0,549	0,646	0,694
6	0,410	0,597	0,667	0,712
7	0,430	0,569	0,635	0,719
8	0,413	0,587	0,660	0,687
9	0,382	0,608	0,653	0,753
10	0,465	0,549	0,583	0,701

Tabela B.12: Resultados obtidos pelo método ML-OCS no conjunto de dados Cell-cycle Cell-division