UM MÉTODO EVOLUCIONÁRIO PARA CLASSIFICAÇÃO HIERÁRQUICA

RAFAEL VIEIRA CARVALHO

UM MÉTODO EVOLUCIONÁRIO PARA CLASSIFICAÇÃO HIERÁRQUICA

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

ORIENTADOR: GISELE L. PAPPA

Belo Horizonte

Abril de 2011

Carvalho, Rafael Vieira
C331m Um Método Evolucionário para Classificação Hierárquica / Rafael Vieira Carvalho. — Belo Horizonte, 2011
xx, 82 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Gisele L. Pappa

1. Computação - Teses. 2. Classificação (Computadores) - Teses. 3. Classificação - Genética - Teses. 4. Algoritmos genéticos - Teses. I. Título.

CDU 519.6*72 (043)

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

FOLHA DE APROVAÇÃO

Um método evolucionário para classificação hierárquica

RAFAEL VIEIRA CARVALHO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

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

PROF. ALEXANDRE PLASTINO DE CARVALHO
Instituto de Computação - UFF

PROF. MARCOS ANDRÉ GONÇALVES
Departamento de Ciência da Computação - UFMG

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 13 de maio de 2011.

"Sábio é aquele que conhece os limites da própria ignorância."
(Sócrates)

Resumo

A tarefa de classificação tradicional (plana) vem sendo extensivamente explorada nas mais diversas áreas de conhecimento. Na última década, os esforços de investigação nestas áreas foram voltados para a tarefa de classificação hierárquica, uma especialização da tarefa anterior. A principal diferença entre as duas é que na classificação hierárquica os exemplos devem ser associados a classes organizadas em uma hierarquia de classes pré-definida, enquanto na classificação plana, nenhuma ordem é imposta para as classes.

Atualmente, existem duas abordagens principais para tratar do problema de classificação hierárquica: (i) utilizar uma abordagem local, também conhecida como *top-down*, ou (ii) utilizar uma abordagem global, também chamada de *big-bang*. A primeira estratégia utiliza classificadores planos para gerar modelos para cada nó ou nó pai da hierarquia, e leva em consideração informações locais da hierarquia na construção dos modelos, terminando com um conjunto de modelos. Para cada novo exemplo, esses modelos são usados para prever suas classes mais genéricas e depois as mais específicas. A abordagem global, por sua vez, normalmente modifica um classificador plano a fim de acrescentar informações sobre a hierarquia na construção do modelo de classificação.

Muitos algoritmos estão sendo utilizados com sucesso na classificação tradicional. Vários desses métodos são derivados de algoritmos já consolidados como o SVM ou Naive Bayes, enquanto novas propostas de algoritmos ainda não foram muito explorados nessa área. Aqui exploramos algoritmos evolucionários para classificação hierárquica. Algoritmos evolucionários são métodos baseados nos princípios da evolução de Darwin e sobrevivência do indivíduo mais forte, e são muito conhecidos por seu poderoso mecanismo de busca global e capacidade de adaptação.

Esse trabalho propõe HCGA (*Hierarchical Classification Genetic Algorithm*), um algoritmo evolucionário para classificação hierárquica que tem como principal contribuição unir ambas informações local e global da hierarquia utilizando uma abordagem *top-down* tanto para criação do modelo de cada classe (treino) quanto para classificação de novos exemplos (teste). Isto representa uma grande inovação, já que os métodos *top-*

down atuais utilizam apenas informação local no treino, perdendo valiosa informação sobre a estrutura da hierarquia de classes.

Como problemas de classificação hierárquica vêm sendo explorados principalmente nas áreas de classificação de texto e bioinformática, o método proposto é aplicado a quatro bases de dados de previsão de proteínas e a "20 Newsgroups", uma base clássica da área de classificação de texto. Os resultados obtidos foram comparados com cinco outros algoritmos comumente utilizados nessa tarefa, obtendo resultados promissores principalmente nos níveis mais baixos da hierarquia.

Abstract

The traditional (flat) classification task has been extensively explored in several areas of knowledge. In the last decade, however, research efforts turn to the task of hierarchical classification, a specialization of the previous task. The main difference between the two is that in hierarchical classification examples are associated with classes organized into a predefined class hierarchy, whereas in flat classification no class order is enforced.

Currently, two main approaches are used to generate hierarchical classification models: (i) the local approach, also known as top-down, and (ii) the global approach, also called *big-bang*. The first strategy uses flat classifiers to generate models for each node or parent node of the hierarchy, and take into account local information of the hierarchy when developing these models, ending with a set of models. For each new example, these models are used to predict their most generic classes, and then the most specific ones. The global approach, in contrast, typically modifies a flat classifier to add information about the hierarchy for building the classification model.

Many traditional classification algorithms are being successfully adapted for hierarchical classification, such as SVM and Naive Bayes, while new algorithms proposals are yet underexplored. In this direction, this work proposes HCGA (Hierarchical Classification Genetic Algorithm), an evolutionary method for hierarchical classification whose main contribution is to merge both local and global information of the hierarchy, using a top-down approach for both creating the class models (training) and classifying new examples (test). This represents a major innovation, since the current top-down methods use only local information in training, and are not aware of the hierarchy structure when building models.

Hierarchical problems have been exploited mainly in the areas of text classification and bioinformatics. Hence, the proposed method was applied to four datasets regarding protein function prediction and a well-known text classification dataset. The results obtained were compared to five other algorithms commonly used in hierarchical classification, obtaining promising results specially in the deeper levels of the hierarchy.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

A tarefa de classificação vêm sendo extensivamente explorada nas mais diversas áreas do conhecimento devido à facilidade do ser humano em modelar os mais variados problemas como uma tarefa de classificação. Na tarefa de classificação tradicional (ou plana), dado um objeto e um conjunto de características (atributos), o objetivo principal é encontrar relações entre os atributos desse objeto e um conjunto de classes pré-definido. Assim, se quisermos prever se o conteúdo de uma determinada página Web encontra-se na categoria "Esporte" ou não, temos que explorar atributos como, por exemplo, a lista de palavras que descrevem a página, ou seu autor ou fonte [37].

Este trabalho, porém, foca em uma especialização da tarefa de classificação plana: a classificação hierárquica (CH). A diferença mais importante entre os problemas de classificação plana e os de classificação hierárquica é que, enquanto na primeira um exemplo deve ser associado a apenas uma classe, na última a associação deve ser feita várias vezes, de acordo com uma hierarquia de classes existente. A Figura 1.1 mostra uma hierarquia de classes de dois níveis para um problema de classificação de documentos, onde um exemplo deve ser associado à classe "Esporte" e depois a "Vôlei" ou a "Futebol".

Problemas hierárquicos têm sido explorados principalmente nas áreas de bioinformática [10] e classificação de texto [67], contextos onde o conhecimento em si é naturalmente organizado de forma hierárquica. Inicialmente, a Web atraiu muitos pesquisadores com seu acelerado crescimento e facilidade de organização de seus documentos eletrônicos em hierarquias de diretórios [56]. Ultimamente, o foco das pesquisas em classificação hierárquica está mudando para a área de bioinformática, principalmente para o cenário de previsão de funções de proteínas. Uma motivação para isso é a importância médica dessas descobertas. Citamos como exemplo a identificação de proteínas GPCR (*G protein-coupled receptor*) e suas famílias, utilizadas nesse trabalho.

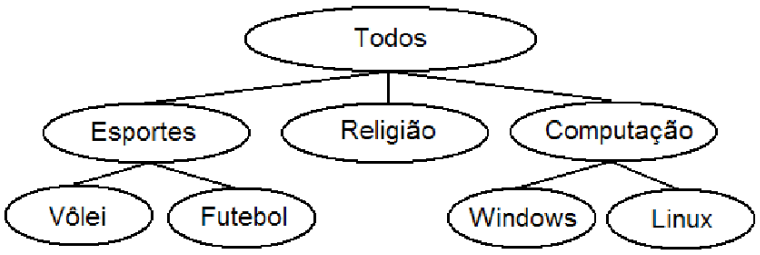


Figura 1.1. Estrutura de classes para a classificação hierárquica. Na classificação plana, normalmente apenas o primeiro nível da hierarquia é considerado

Acredita-se que 40 a 50% dos medicamentos atuais atuam nas atividades desse tipo de proteína [49], e conhecer sua função é essencial para o desenvolvimento de novas drogas.

Entre os grandes desafios da tarefa de classificação, estão hoje o grande número de classes e conjuntos de dados muito desbalanceados. Muitas vezes os exemplos não estão igualmente distribuídos entre as classes, estando concentrados em apenas uma parte delas. Ao tratar do problema de classificação hierárquica, esses desafios se tornam ainda mais difíceis, pois existe um número reduzido de exemplos encontrados nos níveis mais baixos da hierarquia, o que diminui a confiança estatística das previsões de classificação. Quanto mais profundo o nível da hierarquia, mais complexa se torna a criação de modelos de classificação.

Centenas de algoritmos foram propostos para resolver o problema de classificação nas últimas três décadas [99]. Entre os algoritmos explorados na literatura para construção de modelos de classificação plana, destacamos os baseados em algoritmos evolucionários [26, 59, 38, 84]. Algoritmos evolucionários (AE) são métodos estocásticos inspirados nos conceitos Darwinianos de evolução e seleção do indivíduo mais apto [40]. Em um AE, um indivíduo representa uma solução candidata para o problema em questão. Os indivíduos são avaliados de acordo com uma função de *fitness*, e os melhores selecionados para operações como cruzamento e mutação, resultando em uma nova geração formada por indivíduos que são, em teoria, mais adaptados ao ambiente que os indivíduos da geração anterior. Esse processo é repetido até que uma condição de parada seja satisfeita.

Em problemas mais complexos, a solução pode ser um conjunto de indivíduos. Nesse caso, os indivíduos podem ser separados em espécies, onde cada uma representa uma parte do problema a ser abordado, como soluções locais em um problema que admite várias soluções.

Embora AEs já tenham sido anteriormente modelados com sucesso para a tarefa de classificação plana, seu desempenho na resolução de problemas de classificação hi-

erárquica ainda não foi testado. A busca global realizada por AEs associada ao seu poder de representação tornam-no um método atrativo para buscar por modelos eficazes de classificação hierárquica. Além disso, são sutis as diferenças de modelagem entre as tarefas de classificação plana e hierárquica. Finalmente, a evolução simulada pelos AEs permitirá a construção de modelos que "evoluem" conforme a classificação alcança os níveis mais baixos da hierarquia, como descrevemos a seguir.

Assim, o principal objetivo desse trabalho é responder à seguinte pergunta: um algoritmo evolucionário é capaz de gerar um modelo eficaz para a tarefa de classificação hierárquica? Visando investigar esta área de pesquisa ainda inexplorada, esta dissertação propõe um algoritmo evolucionário para encontrar regras de indução para o problema de classificação hierárquica.

Como mostraremos no decorrer dessa dissertação, a maioria dos métodos de classificação hierárquica segue uma de duas abordagens: a abordagem global ou a abordagem local [81]. Aqui, os termos local e global referem-se a como a informação da hierarquia de classes é explorada.

Enquanto métodos globais normalmente se referem a modificações de algoritmos clássicos de classificação plana [62], métodos locais utilizam algoritmos de classificação plana para cada nó, nível ou subárvore da hierarquia [56]. Depois de criados, quando um novo exemplo de teste é apresentado, o método aplica esses modelos de forma *top-down* (do primeiro nível para os níveis mais baixos). Porém, independente do método utilizado, não há um consenso sobre qual dos dois métodos é o melhor, já que uma única implementação não consegue sucesso em toda a área de classificação hirárquica.

Em contraste com essas abordagens, o método proposto nesse trabalho utiliza tanto a informação local quanto a global ao criar modelos. Ele é *top-down* tanto na criação (treino) quanto na classificação de novos exemplos (teste). Conseguimos isso explorando a forma como o método trabalha.

O método constrói modelos para os primeiros níveis da hierarquia. Esses modelos são então utilizados como "sementes" para criação dos modelos do segundo nível, e assim sucessivamente. Dessa forma, a busca nos níveis mais baixos da hierarquia é influenciada pelos modelos encontrados nos níveis mais rasos. Além disso, o método utiliza um mesmo conjunto de atributos para toda a hierarquia (abordagem global), mas pondera-os na especialização das classes dada a importância do atributo na discriminação daquela classe dentre as demais (abordagem local). Assim, criamos uma forma de construção de modelos locais utilizando informação global para cada uma das classes da hierarquia, visando usar o melhor das duas abordagens com esse método inovador.

O método proposto foi aplicado a cinco bases binárias: quatro bases da área de

bioinformática, conhecidas como GPCR, e à "20 Newsgroups", da área de classificação de texto. Essas bases foram escolhidas devido à sua importância na área médica e dificuldade de classificação na área de texto e também por apresentarem características de domínio bem diferentes. Resultados promissores foram obtidos em comparação a algoritmos do estado da arte de abordagem *top-down*. Apesar do foco inicial estar na classificação proteínas e documentos, o método proposto pode ser facilmente adaptado para outros cenários.

1.1 Contribuições

A principal contribuição desse trabalho é um algoritmo evolucionário para tratar problemas de classificação hierárquica. Isso inclui a criação de uma abordagem *top-down* que utiliza informação local e global da hierarquia de classes tanto na criação de modelos de classificação e na classificação de novos exemplos. Além disso, esse trabalho deixa as seguintes contribuições:

1. Realização de um levantamento bibliográfico relacionado a área de classificação hierárquica, com ênfase na classificação de proteínas e de documentos;
2. Realização de um levantamento bibliográfico relacionado a algoritmos evolucionários para classificação tradicional;
3. Proposta e adaptação de operadores genéticos para problemas de classificação hierárquica;
4. Resultados do método aplicados a bases de proteínas e documentos;
5. Publicação dos resultados da aplicação na área de bioinformática na conferência *2011 IEEE Congress on Evolutionary Computation* (2011 IEEE CEC), avaliada como nível A1 pela Capes, que ocorre nos dias 5 e 8 de Junho de 2011 em New Orleans, Louisiana, EUA, com o artigo "HCGA: A Genetic Algorithm for Hierarchical Classification".

1.2 Organização do texto

Dividimos esse texto de forma simples, primeiro delimitando o problema abordado para depois demonstrar o método proposto e seus resultados. Primeiramente, descrevemos melhor o que é a classificação hierárquica no Capítulo 2. Em seguida, falamos sobre algoritmos evolucionários e sua aplicação para o problema de classificação

tradicional no Capítulo 3. Depois desse levantamento bibliográfico, apresentamos o método proposto neste trabalho no Capítulo 4 e descrevemos os experimentos feitos no Capítulo 5. Ao fim, concluímos a dissertação no Capítulo 6, enfatizando os objetivos alcançados e propostas para trabalhos futuros.

Capítulo 2

Classificação Hierárquica

Na tarefa de classificação, dado um objeto e um conjunto de características (atributos), o objetivo principal é encontrar relações entre os atributos desse objeto e um conjunto de classes pré-definido.

A grande maioria dos problemas de classificação descritos na literatura, e aqui nos referimos a eles como problemas de classificação tradicional (ou plana) [38], associa cada exemplo a apenas uma classe pertencente a um conjunto finito de classes. Porém, existe um grande número de problemas em que as classes podem estar relacionadas entre si, possibilitando novos tipos de soluções para a categorização.

Um desses casos ocorre quando uma ou mais classes podem ser divididas em subclasses ou agrupadas em superclasses, ou seja, as classes podem ser dispostas em uma estrutura hierárquica, tal como uma árvore ou um grafo acíclico direcionado (DAG). A problemas de classificação que trabalham com esse tipo de estrutura de classes, denominamos problema de classificação hierárquica.

Assim, definimos a classificação hierárquica (CH) como uma especialização da tarefa de classificação plana. A diferença mais importante entre os dois problemas é que, enquanto na classificação plana um exemplo deve ser associado a apenas uma classe, na CH a associação deve ser feita várias vezes (ex.: por nível, por classe pai), de acordo com uma hierarquia de classes existente.

Na CH, uma previsão de uma classe para um exemplo é dado por um ou mais caminhos de classificação. Um caminho é composto pela sequência possível de classes, uma para cada nível da hierarquia, que liga a raiz da hierarquia à classe prevista mais específica.

A modelagem de problemas de classificação hierárquica depende de duas características relacionadas ao tipo do problema sendo resolvido [85, 38]: (i) a representação da hierarquia das classes, que pode ser feita através de uma árvore ou um grafo ací-

clico direto; e (ii) o resultado desejado em relação à classificação dos exemplos de teste. Quando a hierarquia é representada por uma árvore, cada nó no nível n da árvore pode ser associado a apenas um nó no nível *n-1* (cada classe-filha está associada a uma única classe antecessora, ou nó pai). No caso de hierarquias representadas por grafos (DAG), cada classe-filha pode estar associada a uma ou mais classes antecessoras (classificação multi-rótulo).

Na estrutura de hierarquia em árvore, quanto mais profundo é o nível, geralmente mais difícil é a previsão da classe correta. Isso ocorre porque classes nos níveis mais profundos representam informações mais específicas e são produzidas por modelos induzidos a partir de um número menor de exemplos de treinamento. Para a estrutura DAG, onde um nó pode ter mais de um pai, modelos nos níveis mais profundos da hierarquia podem ser induzidos com um número maior de exemplos de treinamento do que seus nós pais. Apesar disso, na prática, mesmo para DAGs a precisão da previsão decresce com o aumento da profundidade.

Além da representação da hierarquia de classes, deve-se considerar que, em alguns problemas de classificação hierárquica, todos os exemplos de entrada devem ser associados a classes representadas pelos nós-folha (classes no final de cada caminho de classificação). Esses problemas são chamados de problemas de "previsão obrigatória em nós-folha" ou também de "previsão por caminhos completos". Quando essa obrigação não ocorre, o problema de classificação é chamado de problema de "previsão opcional em nós-folha" ou "previsão por profundidade parcial".

Em relação ao resultado desejado, o sistema pode ser desenvolvido de forma que, em situações em que não exista uma quantidade suficiente de informação disponível para tomada de decisões nos níveis mais específicos da hierarquia, o exemplo não seja classificado em todos os níveis. Ao mesmo tempo, o classificador pode ser implementado de maneira que um exemplo seja sempre classificado em todas as classes da hierarquia, independente da "confiança" do classificador em prever a classe para o exemplo. Nos experimentos deste trabalho, o algoritmo implementado suporta hierarquias estruturadas como árvores com ambas previsão parcial ou completa.

É interessante observar que, em um recente trabalho, Silla e Freitas (2011) [81] criam definições para facilitar a categorização dos problemas e algoritmos propostos na área de classificação hierárquica. Além disso, compara os vários métodos propostos na literatura para esse problema, e identifica os que se aplicam a cada uma dessas categorizações propostas. A seguir, sumarizamos essas taxonomias, que serão posteriormente utilizadas para categorização do nosso método e bases de dados. No entanto, note que a caracterização das bases e métodos de alguma forma se sobrepõe, uma vez que o objetivo final é poder selecionar bases e algoritmos compatíveis. Essa caracte-

rização é seguida da descrição das principais abordagens para se tratar problemas de classificação hierárquica e de métricas para avaliação da qualidade dessas abordagens.

2.1 Categorização de problemas de classificação hierárquica

De acordo com [81], um problema de classificação hierárquica pode ser descrito como uma tupla de três atributos $\langle \Upsilon, \Psi, \Phi \rangle$, onde:

- Υ especifica o tipo de grafo representando as classes hierárquicas (nós no grafo) e suas inter-relações (arestas no grafo). Os valores possíveis para este atributo são:
  - T (*tree*), indicando que as classes a serem previstas são organizados em uma estrutura de árvore;
  - D (*Directed Acyclic Graph*), indicando que as classes a serem previstas são organizados em um DAG (Grafo Acíclico Direto).
- Ψ indica se é permitido a um exemplo ter classes associadas a um único ou múltiplos caminhos na hierarquia de classe. Ou seja, na árvore da Fig. 1.1, se houver um exemplo cujas classes mais específicas sejam tanto "Esportes.Vôlei" quanto "Esportes.Futebol", esse exemplo tem vários caminhos de classificação (MPL). Este caminho (classe) pode assumir dois valores, como segue (os nomes dos valores são auto-explicativos):
  - SPL (*Single path of labels*). Esta definição é equivalente a "único caminho por nível de classe".
  - MPL (*Multiple paths of labels*). Esta definição é equivalente ao termo "hierarquicamente multi-rótulo" (ou multi-classe).
- Φ descreve a profundidade das classes dos exemplos da base, e assume dois valores:
  - O valor FD (*full depth labeling* – previsão por caminhos completos) indica que cada exemplo é rotulado em classes de todos os níveis, desde o primeiro nível até o nível das folhas.
  - O valor do PD (*partial depth labeling* – previsão por profundidade parcial) indica que pelo menos um exemplo tem profundidade parcial de classe, ou seja, o valor de classe, em algum nível do caminho (tipicamente o nível folha) é desconhecida.

2.2 Categorização de algoritmos de classificação hierárquica

Também de acordo com [81], um algoritmo de classificação hierárquica é descrito como uma tupla de quatro atributos $\langle \Delta, \Xi, \Omega, \Theta \rangle$, onde:

- Δ indica se o algoritmo pode prever rótulos em apenas um ou em vários (mais de um) caminhos diferentes na hierarquia. Por exemplo, na árvore de hierarquia de classes da Fig. 1.1, se o algoritmo pode prever tanto a classe "Todos.Esportes.Vôlei" quanto a "Todos.Esportes.Futebol" para um dado exemplo, então o algoritmo é capaz de prever múltiplos caminhos (classes). Esse atributo pode assumir dois valores, como segue:

  - SPP (*single path prediction*) ou previsão de caminho único, indica que o algoritmo pode atribuir a cada um dos caminhos do exemplo, no máximo, uma das classes previstas.
  - MPP (*multiple path prediction*) ou previsão de vários caminhos, indica que o algoritmo pode potencialmente atribuir a cada exemplo vários caminhos de classes previstas.

  Note que este atributo é conceitualmente semelhante ao atributo Ψ utilizado para descrever problemas de classificação hierárquica, mas eles se referem a diferentes entidades (problemas e algoritmos). Se o problema alvo é um caminho de SPL (caminho único de classes), seria mais natural usar um algoritmo de SPP (previsão de único caminho), uma vez que um algoritmo de MPP (previsão de vários caminhos) teria "muita flexibilidade" para o problema-alvo e poderia produzir classificações inválidas, erroneamente atribuindo MPL para alguns casos. Se o problema-alvo é uma MPL (caminhos múltiplos de classes), então deve-se usar um algoritmo de MPP, uma vez que algoritmo SPP teria claramente "pouca flexibilidade" para o problema alvo, não prevendo classes extras para alguns casos. Na prática, a fim de evitar a complexidade associada com algoritmos MPP, alguns métodos simplesmente transformam o problema original MPL em um problema de SPL mais simples, e depois aplicam um algoritmo SPP aos dados simplificados.

- Ξ é a profundidade de previsão do algoritmo. Esse atributo pode ter dois valores:

  - MLNP (*mandatory leaf-node prediction*), ou previsão nó folha obrigatória, significa que o algoritmo sempre deve atribuir classe(s) folha(s) a um exemplo.

  - NMLNP (*non-mandatory leaf-node prediction*), ou previsão nó folha não obrigatória, significa que o algoritmo pode atribuir classes em qualquer nível (incluindo as classes folha).

  Novamente, há uma relação natural entre esse atributo para descrever algoritmos e seu atributo correspondente para descrever os problemas. Se o alvo problema é FD (profundidade total de classificação), deve-se, naturalmente, utilizar um algoritmo MLNP, uma vez que um algoritmo NMLNP teria "muita flexibilidade" e faria "subclassificações" em alguns casos. Se o problema-alvo é um PD (classificação parcial), deve-se utilizar um algoritmo NMLNP, uma vez que algoritmo MLNP teria "pouca flexibilidade" e "classificaria excessivamente" em alguns casos.

- Ω é a estrutura da taxonomia que o algoritmo pode manipular, e assume os mesmos valores que Υ: T (*Tree*) ou DAG (*Directed Acyclic Graph*).

  Em princípio, um algoritmo criado para lidar com DAGs pode ser aplicado diretamente (sem modificação) a árvores. No entanto, o inverso não é verdadeiro.

- Θ é a categorização do algoritmo sob a taxonomia proposta e tem os valores:

  - LCN (*local classifier per node*) – classificador local por classe. Dentro desta categoria, há também um outro argumento que deve ser especificado, que é a estratégia utilizada para a seleção de exemplos negativos e positivos. Ela pode assumir sete valores, como detalhado na Seção 2.3.2.1.
  - LCL (*local classifier per level*) – classificador local por nível.
  - LCPN (*local classifier per parent node*) – classificador local por classe pai.
  - GC (*global classifier*) – classificador Global.

  Essas categorias de algoritmos são detalhadas na próxima seção.

2.3 Abordagens utilizadas em problemas hierárquicos

Essa seção detalha a categorização dos algoritmos (conforme descrito pelo parâmetro Θ) e mostra exemplos de métodos seguindo essas abordagens. Além das abordagens naturalmente hierárquicas, é interessante ressaltar que muitos métodos ainda ignoram a hierarquia das classes, e transformam o problema hierárquico em um ou mais problemas de classificação plana, e utilizam um classificador plano na classificação. Porém, considerando métodos intrinsecamente hierárquicos, note que eles

Tabela 2.1. Classificadores globais e algoritmo de origem, adaptado de [81]

Algoritmo de origem	Abordagem Global
Ant-Miner	Otero et al. (2009) [66]
Association rule-based classifier	Wang et al. (2001) [96]
C4.5	Clare e King (2003) [22]
Naive Bayes	Silla Jr e Freitas (2009) [82]
Predictive clustering trees	Blockeel et al. (2006) [44] e Vens et al. (2008) [94]
Ripper	Sasaki e Kita (1998) [77]
Kernel machines	Cai e Hofmann (2004, 2007) [12, 13], Dekel et al. (2004a,b) [30, 31], Rousu et al. (2005, 2006) [74, 75], Seeger (2008) [80], Qiu et al. (2009) [71] e Wang et al. (2009) [95]

exploram a hierarquia de duas formas: local (LCN, LCL e LCPN de Θ) ou global (GC). As próximas subseções detalham esses métodos.

2.3.1 Métodos Globais (*Big-Bang*)

Visando criar métodos que utilizem a informação da hierarquia para construção de um único classificador hierárquico, surgiu a abordagem global (*big-bang*) [62]. O ponto comum entre os métodos seguindo essa abordagem é considerar a informação disponível através da hierarquia de classes globalmente. Porém, a maneira como esses métodos alcançam esse objetivo difere de acordo com as técnicas em que eles são baseados.

A maioria dos trabalhos que implementam métodos globais seguem uma abordagem estatística. Métodos Bayesianos [62] e SVMs (*Support Vector Machines*) [51, 97, 20, 65] estão entre os métodos mais populares, principalmente no contexto de classificação de documentos.

Além de métodos Bayesianos e baseados em SVMs, algumas extensões de métodos de indução de regras anteriormente criados para tarefa de classificação plana, como C4.5 [22] e Ripper [77], também foram criadas. Novas abordagens também foram propostas no contexto de regras de associação [96]. A Tabela 2.1, adaptado de [81], apresenta trabalhos que utilizam a abordagem global a partir de alterações de algum algoritmo tradicional de origem.

Apesar de normalmente não obterem taxas de acerto tão boas quanto os métodos *top-down*, a abordagem global tem a vantagem de produzir um único modelo de classificação, enquanto abordagens locais produzem vários.

É interessante ressaltar que o número de trabalhos que utilizam uma abordagem global é pequeno se comparado ao número de métodos seguindo as abordagens locais, que dividem o problema de classificação hierárquica em vários problemas de classificação plana.

2.3.2 Métodos Locais (*Top-Down*)

Em contraste com a abordagem global, os métodos locais consideram a hierarquia de classes sob uma perspectiva de informação local. Em outras palavras, essa abordagem divide o problema de classificação hierárquica em um conjunto de problemas de classificação plana.

O primeiro tipo de classificador local (também conhecida como abordagem *top-down* na literatura) foi proposto em [56]. A partir deste trabalho, vários autores utilizaram diferentes versões desta abordagem para lidar com problemas de classificação hierárquica. Essas diferentes versões podem ser agrupadas com base em como elas usam essa informação local e como constroem seus classificadores em torno dela. Mais precisamente, existem três formas e padrões comuns de uso da informação local [81]: um classificador local por nó (LCN), um classificador local por nó pai (LCPN) e um classificador por nível local (LCL). As subseções seguintes discutem cada uma delas, e trabalhos baseados em abordagens locais são citados na Tabela 2.2, com ênfase em aplicações de classificação de texto e bioinformática.

Deve-se notar que, embora os três tipos de algoritmos de classificação hierárquica local supracitados diferem significativamente em sua fase de treinamento, **elas compartilham uma abordagem de teste denominada *top-down***. Em essência, nesta abordagem, para cada novo exemplo no conjunto de teste, o primeiro modelo prevê seu primeiro nível (mais genérico) de classe. Ele então usa essa classe prevista para restringir as escolhas das classes a serem previstas no segundo nível (as classes candidatas de segundo nível só são válidas se são filhas da classe prevista no primeiro nível), e assim por diante, recursivamente, até que a previsão mais específica seja feita.

Porém, a abordagem top-down é essencialmente um método para evitar inconsistências nas previsões de classe em níveis diferentes na hierarquia. Como resultado, uma desvantagem dessa abordagem é que um erro em um nível vai ser propagado para os próximos níveis da hierarquia, a menos que algum procedimento para evitar este problema seja utilizado. Se o problema for previsão não-obrigatória de nós folha (PD), uma abordagem de bloqueio (onde um exemplo é passado para o nível imediatamente inferior somente se a confiança na previsão ao nível atual é superior a um limiar) pode evitar que erros de classificação sejam propagados, em detrimento de fornecer ao usuá-

Tabela 2.2. Resumo da literatura sobre classificação hierárquica de acordo com o domínio de aplicação e o tipo de abordagem Θ, adaptado de [81]

Tipo de aplicação	Abordagem de CH (Θ)	Lista de trabalhos
Texto	LCN	D'Alessio et al. (2000) [24], Sun e Lim (2001) [87], Mladenic e Grobelnik (2003) [64], Sun et al. (2003, 2004) [85, 86], Wu et al. (2005) [101], Cesa-Bianchi et al. (2006,2006a) [18, 17], Esuli et al. (2008) [34], Jin et al. (2008) [50], Punera e Ghosh (2008) [69], Xue et al. (2008) [102] e Bennett e Nguyen (2009) [9]
	LCPN	Koller e Sahami (1997) [56], Chakrabarti et al. (1998) [19], McCallum et al. (1998) [62], Weigend et al. (1999) [98], D'Alessio et al. (2000) [24], Dumais e Chen (2000) [32], Ruiz e Srinivasan (2002) [76], Tikk e Biró (2003) [89], Tikk et al. (2003, 2007) [90, 88], Kriegel et al. (2004) [57] e Gauch et al. (2009) [39]
Funções de proteínas	LCN	Wu et al. (2005) [101], Barutcuoglu et al. (2006) [8], Guan et al. (2008) [42], Valentini (2009) [92] e Cesa-Bianchi e Valentini (2009) [16]
	LCPN	Holden e Freitas (2005, 2006, 2008, 2009) [46, 47, 49, 48] Secker et al. (2007, 2010) [79, 78], Costa et al. (2008) [23] e Kriegel et al. (2004) [57]
	LCL	Clare e King (2003) [22]

rio previsões menos específicas (menos úteis) de classe. Esse problema é discutido na Seção 2.3.3.

2.3.2.1 Classificador local por nó (LCN)

A abordagem LCN consiste em treinar um classificador binário para cada nó da hierarquia de classes (exceto o nó raiz), e é, de longe, a abordagem mais utilizada na literatura [38].

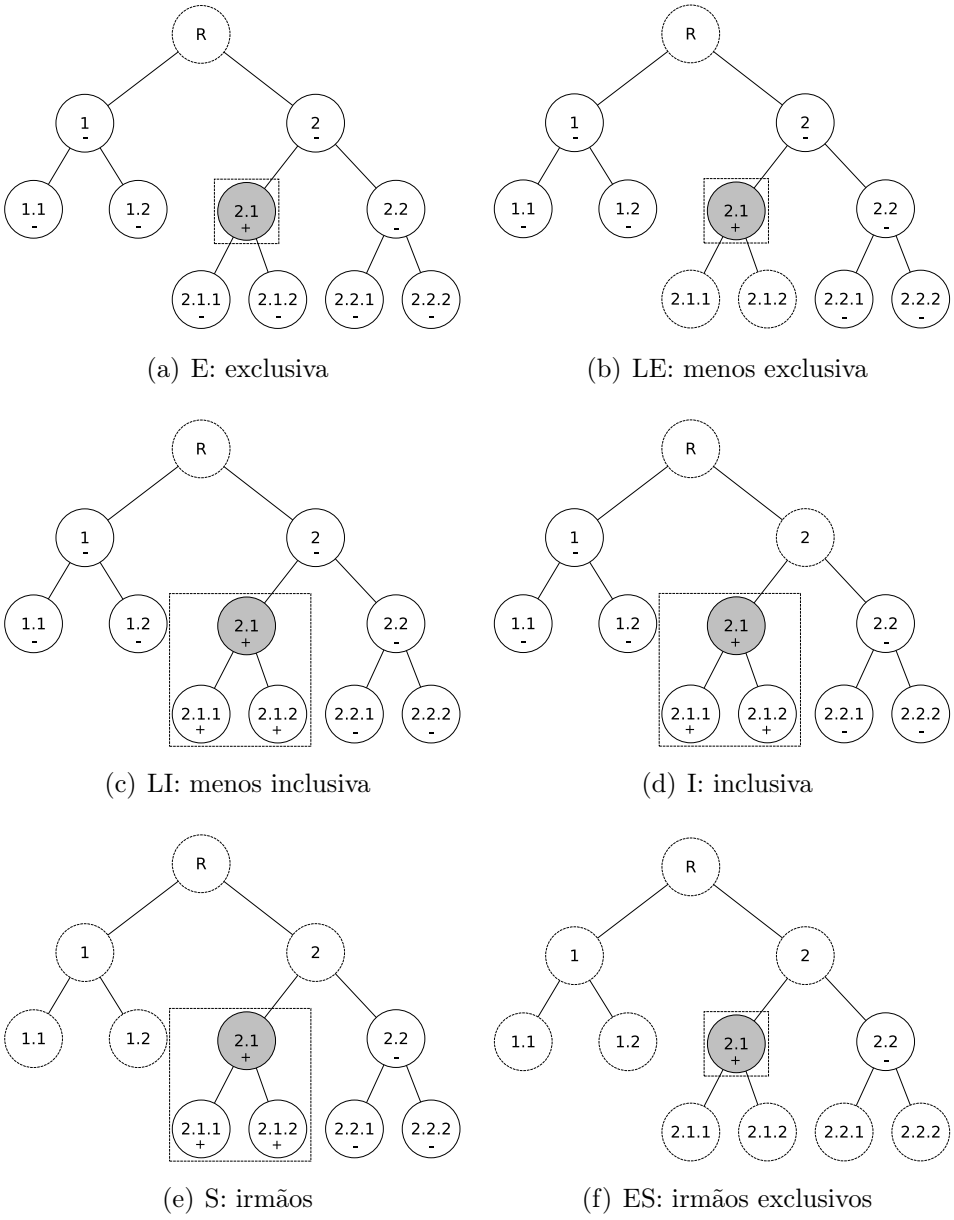
Existem diferentes maneiras de definir o conjunto de exemplos a serem utilizados no treinamento dos classificadores binários de cada nó (classe). Para a criação de um classificador para uma classe, um conjunto de exemplos deve ser considerado como pertencente à classe (positivos) e outro conjunto distinto deve ser considerado como não pertencente à classe (negativos). A escolha desses exemplos é na maioria das vezes realizada antes do treinamento, ou escolhido dinamicamente de acordo com a necessidade [35].

Na literatura, vários trabalhos costumam usar apenas uma forma de escolher o conjunto positivo e negativo de exemplos [87, 85, 86]. Exceções são os trabalhos de [33, 35]. No trabalho de [33] os autores identificam quatro diferentes políticas para essa escolha. Em [35] os autores focam na seleção dos exemplos negativos e empiricamente comparam quatro políticas (duas padrões comparadas com duas outras novas). No entanto, as novas abordagens são limitadas a problemas de categorização de texto e obtiveram resultados semelhantes com as abordagens padrões [81].

Abaixo citamos políticas padrão definidas na literatura para escolha dos conjuntos de exemplos positivos Tr^+ e negativos Tr^-, que são descritas e analisadas em [81]. Em cada padrão, consideramos a construção do modelo para uma classe 2.1 da hierarquia exemplificada na Figura 2.1.

- **E** (*exclusive*): política "exclusiva" [33]. Apenas exemplos pertencentes à classe 2.1 são selecionados para $Tr^+_{2.1}$ do treinamento para a classe 2.1, enquanto todo o resto é usado como exemplos negativos $Tr^-_{2.1}$. Esta abordagem tem alguns problemas. Primeiro, ela não considera a hierarquia para criar os conjuntos de treinamento local. Em segundo lugar, ela está limitada a problemas onde há exemplos com classificação de profundidade parcial, i.e., exemplos cuja classe é conhecida apenas para níveis mais rasos. Em terceiro lugar, usar os nós descendentes de 2.1 como exemplos negativos parece contra-intuitivo, considerando que os exemplos que pertencem aos descendentes da classe 2.1 implicitamente também pertencem à classe 2.1.

- **LE** (*less exclusive*): política "menos exclusiva" [33]. Neste caso, $Tr^+_{2.1}$ consiste no conjunto de exemplos da classe 2.1 e seus descendentes 2.1.1 e 2.1.2, enquanto $Tr^-_{2.1}$ consiste no conjunto de exemplos cuja classe é qualquer classe diferente de



(a) E: exclusiva (b) LE: menos exclusiva

(c) LI: menos inclusiva (d) I: inclusiva

(e) S: irmãos (f) ES: irmãos exclusivos

Figura 2.1. Representação das classes presentes nos conjuntos de exemplos positivos (+) e negativos (-) para construção do classificador para a classe 2.1. Os exemplos pertencentes às classes tracejadas não são considerados pela política correspondente

2.1 e seus descendentes. Essa abordagem evita o primeiro e terceiro problemas acima mencionados da política exclusiva, mas ainda é limitada a problemas onde classificação em profundidade parcial é necessária.

- **LI** (*less inclusive*): política "menos inclusiva" [33] (ou política "ALL" [35]). Neste caso, $Tr^{+}_{2.1}$ consiste no conjunto de exemplos da classe 2.1 ou de seus descendentes,

e $Tr^{-}_{2.1}$ consiste no conjunto de exemplos restantes de toda a hierarquia.

- **I** (*inclusive*): política "inclusiva" [33]. $Tr^{+}_{2.1}$ é o conjunto de exemplos da classe 2.1 ou seus descendentes e $Tr^{-}_{2.1}$ consiste no conjunto de exemplos restantes menos os exemplos cuja de classes antecessoras a 2.1.

- **S** (*siblings*): política "irmãos" [35] (ou "conjunto de treinamento hierárquico" [15]). Neste caso, $Tr^{+}_{2.1}$ consiste no conjunto de exemplos da classe 2.1 e seus descendentes, e $Tr^{-}_{2.1}$ consiste no conjunto de exemplos cuja classe tem nível igual ou maior (mais específico) que o nível de 2.1. Ou seja, desconsidera para $Tr^{-}_{2.1}$ os exemplos de classes mais específicas.

- **ES** (*exclusive siblings*): política "irmãos exclusivo" [15] (ou "conjunto de treino próprio"). Neste caso, $Tr^{+}_{2.1}$ consiste no conjunto de exemplos da classe 2.1 e $Tr^{-}_{2.1}$ consiste no conjunto de exemplos de classes irmãs de 2.1, ou seja, as classes de mesmo nível que 2.1, mas que tenham o mesmo nó pai (antecessor).

- D (*Dynamic*): política dinâmica [35]. Neste caso, os exemplos positivos e negativos são selecionados de forma dinâmica, de acordo com alguma outra abordagem de seleção de exemplos. Em [35], uma métrica de distância é criada para identificar exemplos próximos. Assim, [35] escolhe $Tr^{+}_{2.1}$ seguindo uma política LI, mas, para $Tr^{-}_{2.1}$, uma nova política BestLocal é proposta. BestLocal escolhe, de todos os exemplos negativos da abordagem LI, apenas aqueles próximos ao centróide do conjunto $Tr^{+}_{2.1}$. Com isso, ela reduz o número de exemplos utilizados durante o treinamento, mas torna o cálculo dessa distância complexo em bases que possuem classes não linearmente separáveis.

A Tabela 2.2 lista 24 trabalhos na área de classificação de proteínas e 15 na área de classificação de texto, onde as diferenças experimentais no uso de cada uma das categorias de exemplos acima descritos podem ser observadas.

2.3.2.2 Classificador local por nó pai (LCPN)

Outro tipo de informação local que pode ser usado é a abordagem onde, para cada nó pai da hierarquia de classes, um classificador multi-classe é treinado para distinguir entre seus nós filhos [81]. Esses métodos podem ser treinados usando as abordagens S (*siblings*) e ES (*exclusive siblings*) descritos na seção anterior.

Normalmente, na abordagem LCPN o mesmo algoritmo de classificação é utilizado em toda a hierarquia de classes. Uma extensão deste tipo de abordagem local, conhecido como "classificação seletiva", é proposto em [79] na tentativa de melhorar a

precisão da previsão da abordagem LCPN usando algoritmos de classificação diferentes em diferentes nós pais da hierarquia de classe. Para determinar qual classificador deve ser utilizado em cada nó da hierarquia de classe, durante a fase de construção do modelo, o conjunto de treinamento é dividido em um conjunto de sub-treinamento e um conjunto de validação, com exemplos distribuídos aleatoriamente. Diferentes classificadores são treinados com o sub-conjunto de treinamento e depois são avaliados no conjunto de validação. O classificador escolhido para cada nó da classe pai é aquele com a maior precisão de classificação sobre o conjunto de validação.

Uma melhoria em relação à abordagem de classificação seletiva foi proposta em [49], onde um algoritmo de otimização de inteligência coletiva (*swarm intelligence*) foi utilizada para realizar a seleção de classificadores em uma abordagem NMLNP (*non-mandatory leaf node prediction*). A motivação por trás dessa abordagem é que a classificação seletiva original usa um método de pesquisa local, que tem uma visão limitada do conjunto de dados de treinamento, enquanto o algoritmo de inteligência coletiva realiza uma pesquisa global que considera toda a árvore de classificadores (tendo uma visão completa dos dados de treinamento) de uma vez.

Outra melhoria em relação à abordagem de classificação seletiva foi proposto em [82], onde ambos o melhor classificador e o melhor tipo de representação de exemplos (envolvendo diferentes subconjuntos de atributos previsores) são selecionados para cada classificador nó pai. Além disso, [78] estende a sua abordagem de seleção de classificação anterior, a fim de selecionar os classificadores e atributos em cada nó de classificador.

Além desses métodos, vários outros nas áreas de classificação de texto e proteínas são listados na Tabela 2.2. Note que aqui discutimos apenas a abordagem LCPN no contexto de um problema onde cada exemplo é associado a apenas uma classe (*uni-label*) de uma hierarquia de classes estruturada em árvore. Para detalhes em métodos para multi-rótulo (*multi-label*), consulte [82].

2.3.2.3 Classificador local por nível (LCL)

Este é o tipo de abordagem "local" de classificação menos utilizada na literatura [81] até o momento. O classificador local por nível consiste em treinar um classificador multi-classe para cada nível da hierarquia de classes (dependendo se o problema é de rótulo único ou multi-rótulo). A criação do conjunto de treinamento aqui é implementada da mesma forma como no classificador local por abordagem nó pai, ou seja, utilizando S (*siblings*) ou ES (*exclusive siblings*).

Uma forma simples de classificar exemplos de teste usando classificadores treinados por esta abordagem é a seguinte: Quando um novo exemplo de teste é apresentado

ao classificador, pegue a saída de todos os classificadores (um classificador por nível) e use essas informações como a classificação final. A principal desvantagem desta abordagem de previsão de classes é estar propensa a inconsistência de associação de classes. Por exemplo, é possível ter resultados como classe 2, no primeiro nível, classe 1.2 no segundo nível e classe 2.2.1 no terceiro nível. Portanto, se essa abordagem é usada, ela deve ser complementada por um método de pós-processamento, que tenta corrigir a incoerência de previsão.

Para evitar esse problema, a abordagem *top-down* poderia ser utilizada. Neste contexto, a classificação de um novo exemplo de teste restringiria a saída de classificação possível em um determinado nível apenas para os nós filhos do nó de classe previsto no nível anterior. Apesar de profundidade ser normalmente um conceito de árvore, essa abordagem também pode ser aplicada a DAGs, mas de forma muito mais complexa. Observe que apenas um exemplo desse tipo de classificação aparece na Tabela 2.2.

2.3.3 Previsão não-obrigatória em nós folha (NMLNP)

Essa seção discute o problema da previsão não-obrigatória em nós folha (NMLNP – *non-mandatory leaf node prediction*). Ele ocorre quando a classe mais específica atribuída a um exemplo pode estar em qualquer nível da hierarquia, e não necessariamente nos nós folhas (nível mais profundo) [87]. A este problema damos o nome de bloqueio (*blocking*) [86].

O bloqueio ocorre quando, durante o processo *top-down* da classificação de um exemplo de teste, o classificador em um determinado nível na hierarquia de classe prevê que o exemplo em questão não tem a classe associada ao classificador. Neste caso, a classificação do exemplo será "bloqueada", ou seja, o exemplo não será passado para os descendentes desse classificador.

Uma maneira de implementar o bloqueio é associar um limiar a cada nó classe, e se o valor de confiança ou probabilidade posterior a classificação em um nó de uma determinada classe em nível mais profundo que a atual – para um exemplo de teste em questão – é inferior a este limiar, a classificação pára para esse exemplo. Outras estratégias são discutidos em maiores detalhes em [86].

2.4 Avaliação de métodos hierárquicos

Uma das grandes dificuldades na classificação hierárquica está na avaliação dos resultados. Como em qualquer problema de classificação, uma previsão de classe para

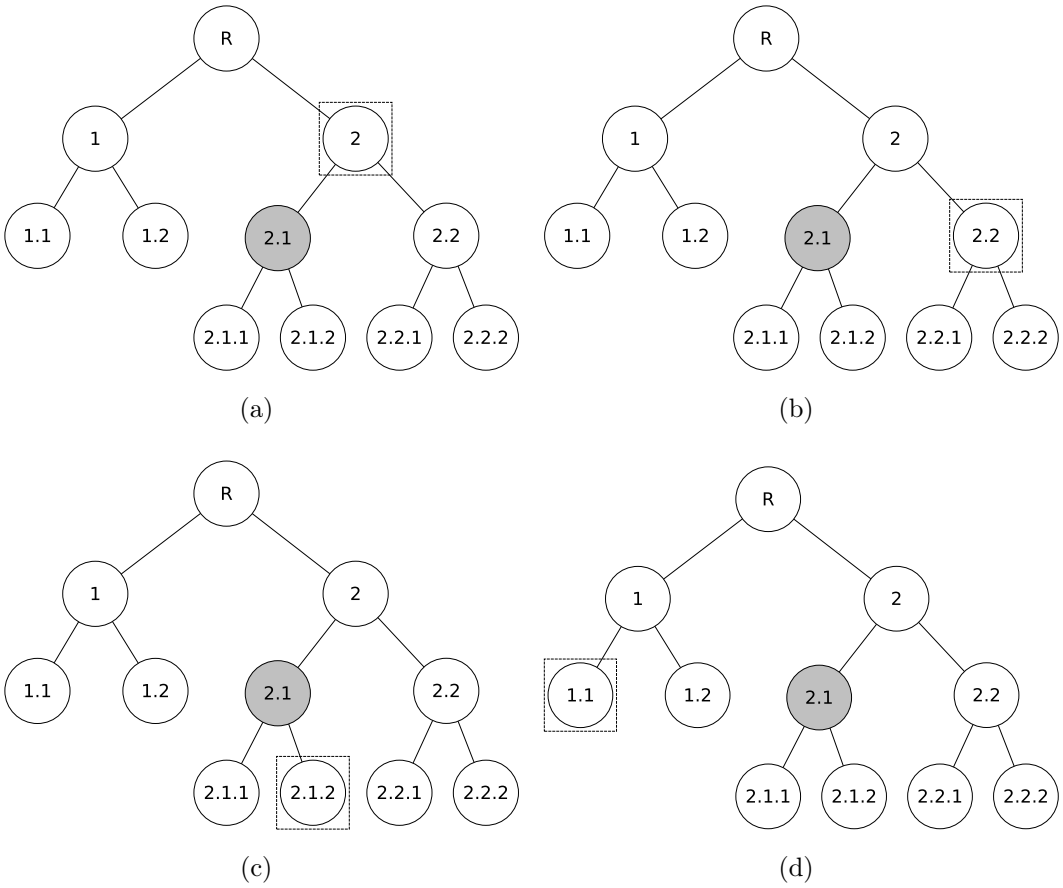


Figura 2.2. Exemplos de erros de classificação em uma hierarquia. Na classificação de um exemplo da classe 2.1, qualquer outra classe prevista é errada. O problema da hierarquia é priorizar os erros mais aceitáveis.

um exemplo deve acertar a classe desse exemplo e todas as previsões diferentes são consideradas igualmente erradas.

Porém, como um modelo de classificação hierárquico pode prever qualquer classe de uma hierarquia, vários tipos de erros podem ser encontrados. Na Figura 2.2 podemos identificar quatro erros de previsão. Nela, são previstas as classes 2, 2.2, 2.1.2 e 1.1 para um exemplo da classe 2.1. Ao classificar o exemplo como 2 (a), o intuitivo é consideramos que obtemos uma classificação parcialmente correta, ou seja, a previsão acertou até onde pôde e parou. Caso este classificador tentasse prever uma classe para o próximo nível, poderia errar, prevendo a 2.2 (b). Vemos que prever a classe 2 é melhor que prever a classe 2.2, já que a segunda adiciona uma especialização errada. Porém, a previsão de 2.1.2 (c) ainda demonstra outra característica de erro: classificar um nível a mais que a classe certa. Cabe ao usuário definir se essa especialização além da classificação correta é interessante ou não para a aplicação sendo tratada. Uma

classificação completamente errada é prever a classe 1, mas prever a classe 1.1 (d) pode ainda ser considerado pior, pois houve propagação do erro de classificação identificado no primeiro nível para o segundo nível. Esses e outros problemas são detalhadamente analisados em [54].

Assim, muitas vezes é difícil quantificar se um método é melhor ou pior do que o outro, ou definir como penalizar eficientemente uma classificação para que classificadores distintos possam ser comparados independente da aplicação. Esses problemas são identificados tanto nas abordagens locais tanto nas globais. Cabe às métricas de avaliação priorizar classificadores a partir de suas previsões. Porém, essa avaliação fica difícil quando um erro pode não ser igual a outro, como ocorre no problema de classificação hierárquica, independente da abordagem utilizada para tratá-lo.

Várias propostas de avaliação da hierarquia por completo já foram feitas, como métricas que permitem penalizar classificações parciais (quando uma classificação erra ou termina antes de classificar a classe final) [85, 18, 55]. Porém, ainda não existe um consenso sobre qual a melhor forma de avaliação a ser utilizada na classificação hierárquica.

Na avaliação de abordagens *top-down*, métricas de classificação plana, quando analisadas sob uma ótica de hierarquias, podem deixar de lado algumas informações que podem ser úteis para avaliação da previsão realizada, mas são muito eficazes quando utilizadas para identificar a qualidade das classificações para cada nível da hierarquia. Exemplos de utilização dessas métricas podem ser encontrados em vários trabalhos [49, 67, 78, 103].

Capítulo 3

Algoritmos Evolucionários para Classificação Tradicional

Algoritmos evolucionários (AE) são métodos de busca inspirados nas ideias da evolução biológica Darwiniana, que prega que indivíduos mais adaptados ao ambiente possuem mais chance de se reproduzir e dar continuação à sua espécie. Apesar de existirem vários tipos de AE, todos eles possuem alguns elementos básicos em comum [5, 37]:

1. AEs tipicamente trabalham com uma população de indivíduos (soluções candidatas para o problema sendo atacado), ao invés de apenas uma solução candidata.

2. Eles utilizam um método de seleção com viés de sobrevivência na qualidade da solução para resolver o problema (*fitness*), dando maior chance aos indivíduos com melhor *fitness*. A seleção é o método que determina quais indivíduos terão partes de seu código genético passado para as próximas gerações.

3. Eles geram novos indivíduos através de um mecanismo de herança de código genético dos indivíduos, utilizando operações estocásticas para determinar quais partes desse código serão mantidas na próxima geração e quais serão mutadas.

Dentre os tipos de AE, destacamos quatro principais abordagens [37]: estratégias evolucionárias, programação evolucionária, algoritmos genéticos e programação genética. Estratégias evolucionárias (ES) e programação evolucionária (EP) são abordagens similares, pois evoluem indivíduos representados por um vetor de valores reais, e priorizam a manutenção do código genético durante a evolução utilizando principalmente a operação de mutação. A mutação pode alterar grande parte do indivíduo ou focar

Algoritmo 3.1 AGPadrão

Parâmetro: P // População inicial

while critério de parada não satisfeito **faça**
  $P^* \leftarrow \emptyset$
  while P^* não completa **faça**
    se probabilidade de cruzamento satisfeita **então**
      $ind1 \leftarrow selecao(P)$
      $ind2 \leftarrow selecao(P)$
      $(filho1, filho2) \leftarrow cruzamento(ind1, ind2)$
      $P^* \leftarrow P^* \bigcup \{filho1, filho2\}$
    fim se
    se probabilidade de mutação satisfeita **então**
      $ind3 \leftarrow selecao(P)$
      $filho3 \leftarrow mutacao(ind3)$
      $P^* \leftarrow P^* \bigcup \{filho3\}$
    fim se
  fim while
  $P \leftarrow P^*$
fim while
retorne $melhorIndividuo(P)$

em uma parte menor, de acordo com um parâmetro fornecido pelo usuário. A operação de cruzamento também pode ser utilizada por esses métodos, mas tem menor probabilidade.

Algoritmos genéticos (AG) e a programação genética (PG) procuram simular de forma mais fiel o comportamento da evolução natural, priorizando o cruzamento como o principal operador para busca no espaço de soluções, e utilizando a operação de mutação com menor probabilidade. Porém, enquanto o AG utiliza, na maioria das vezes, indivíduos representados por vetores binários ou reais, um PG é normalmente utilizado para evoluir indivíduos mais complexos, representados por árvores de decisão, que podem ser programas, ou até funções (ou operações).

Devido à sua simplicidade e seu reconhecimento na área de classificação tradicional [100, 91, 73, 67, 60, 11, 41], este trabalho propõe um AG implementado especificamente para o problema de classificação hierárquica (descrito na Seção 4.1). Assim, o restante dessa seção focará em AGs.

Baseado na teoria da evolução das espécies, o objetivo de um AG é evoluir uma população inicial P através de operações de cruzamento e mutação até que algum critério de parada seja satisfeito. O Algoritmo 3.1 apresenta um pseudocódigo de um algoritmo genético padrão.

Dada uma população P de entrada e uma população auxiliar P^* inicialmente vazia (linha 2), indivíduos são escolhidos de P (linhas 5, 6 e 11) por um critério de seleção (descrito na Seção 3.4). Cada indivíduo é variado através de operações de cruzamento e mutação (linhas 7 e 12). Os indivíduos resultantes dessas operações são inseridos em P^* (linhas 8 e 13). A probabilidade de ocorrer qualquer dessas operações é definida pelo usuário (linhas 4 e 10). Chamamos uma sequência de operações como esta (linhas 3-15) de iteração. São executadas tantas iterações quanto necessário até que P^* esteja completa com novos indivíduos. Quando isto ocorre, a população P é atualizada com os indivíduos de P^*, formando a nova geração de P (linha 16). São criadas novas gerações até que um critério de parada seja satisfeito (linha 1). Esse critério pode ser um número máximo de gerações ou a obtenção da solução ótima, que normalmente ocorre quando há convergência de todos os indivíduos para uma única solução. Ao fim da evolução, o melhor indivíduo da população P é retornado (linha 18).

Uma das grandes vantagens dos AGs é que eles podem ser facilmente adaptados para resolver problemas nas mais diversas áreas de aplicação simplesmente escolhendo uma representação de indivíduo e uma *fitness* apropriadas. Neste trabalho, tratamos o problema de classificação hierárquica. Apesar de AGs ainda não terem sido utilizados para resolver esse tipo de problema, diversos AGs já foram propostos para a tarefa de classificação tradicional.

Assim, nas seções seguintes, descreveremos como as principais características de um AG (veja Algoritmo 3.1) podem ser implementadas em problemas de classificação tradicional. A representação de um indivíduo é descrita na Seção 3.1, enquanto as formas de se inicializar uma população e selecionar seus indivíduos são descritas na Seção 3.2. Métricas de avaliação utilizadas como *fitness* seguem na Seção 3.3. A seleção de indivíduos para a nova geração é descrita na Seção 3.4. Os tipos de operações mais comuns em AG e os específicos para classificação são descritos respectivamente na Seção 3.5. Por fim, na Seção 3.6, são descritos AGs utilizados na literatura para classificação tradicional.

3.1 Representação dos indivíduos

Em um problema de classificação, cada indivíduo representa um classificador. Porém, existem vários tipos de modelos de classificadores ou parte dele. A maioria dos trabalhos que utilizam AGs para classificação utilizam como modelo para os classificadores funções matemáticas ou regras de decisão. Quando os indivíduos são re-

presentados como funções matemáticas explorando relações dos atributos da base, um limiar é utilizado para determinar a classe do exemplo de acordo com o valor retornado pela função. No caso de regras, elas são modelos do tipo ***se*** *<antecedente>* ***então*** *<consequente>*. Por exemplo, ***se*** *contém atributo "bola"* ***ou*** *"placar"* ***então*** *"Esporte"* caracteriza uma regra que atribui a classe "Esporte" aos documentos que possuem o termo "bola" ou "placar".

AGs para classificação se baseiam em duas abordagens [27, 37]: a abordagem *Michigan* e a abordagem *Pittsburg*. Os sistemas que utilizam a abordagem de Michigan mantêm uma população de regras individuais que competem entre si por *espaço* e *prioridade* na população. Um classificador com essa abordagem deve ser formado pelo conjunto de regras distintas que melhor solucione o problema. Assim, cada indivíduo é uma regra e o AG deve retornar um conjunto de indivíduos (regras).

Em contraste, os sistemas que utilizam a abordagem de Pittsburgh mantêm uma população onde indivíduos representam conjuntos de regras de comprimento variável, e competem entre si com relação ao desempenho no domínio da tarefa. Assim, regras com *fitness* semelhantes podem cobrir o mesmo espaço de soluções. Um classificador com essa abordagem precisa apenas do melhor indivíduo para realizar a classificação.

Dentre as vantagens e desvantagens de utilização de cada uma dessas abordagens, há muito o que ser estudado para identificar qual é a mais eficaz, uma vez que o problema de interação entre as regras durante a evolução [37] é difícil de ser avaliado.

3.2 Inicialização da População

Na maioria das aplicações de AG, a população é gerada aleatoriamente. Porém, no contexto de classificação, essa abordagem tem um grande problema [37]: é possível que as regras criadas não cubram nenhum dos exemplos da base de treinamento, obtendo uma qualidade (*fitness*) muito baixa. Por cobrir entendemos que um exemplo da base deve possuir os valores especificados pelo antecedente da regra. Uma simples solução para esse problema é inicializar a população aleatoriamente utilizando regras mais gerais, com um número pequeno de condições.

Uma forma um pouco mais inteligente de solucionar esse problema é inicializar a população com regras que garantem acerto de pelo menos uma das instâncias da base [61]. Para isso, basta escolher aleatoriamente um exemplo da base de treinamento que pertença à classe prevista pela regra (também chamado de semente), e gerar condições válidas baseada neste exemplo. Assim, qualquer que seja a regra criada, pelo menos o exemplo escolhido será coberto por ela.

Com maior conhecimento do problema, outras formas de inicialização podem ser propostas. Durante a evolução do AG, por exemplo, podemos manter as melhores soluções encontrados em uma geração para que elas não se percam durante as operações de cruzamento e mutação. Essa abordagem é chamada de *elitismo*. No Algoritmo 3.1, a população P^* seria inicializada com a elite de P ao invés de vazia, com exceção da primeira geração.

3.3 Avaliação de *Fitness*

Uma das questões mais importantes no desenvolvimento de um AG é a função de *fitness*. Essa função determina a qualidade de uma solução candidata ao problema, e por isso é utilizada como função de otimização pelo AG. Idealmente, a *fitness* deve medir a qualidade de um indivíduo o mais precisamente possível, sujeita a restrições computacionais, conhecimento sobre o problema sendo atacado e os requisitos do usuário. No problema de classificação, onde um indivíduo normalmente representa um modelo de classificação, a *fitness* representa a qualidade do indivíduo para classificar um determinado conjunto de exemplos de uma base de dados.

Em classificação, existem vários critérios de avaliação de qualidade de uma regra. Algumas delas são [37]: precisão (confiança), revocação (sensitividade), taxa de acerto positivo (completude), taxa de acerto negativo (especifidade), acurácia e curvas ROC. Qualquer combinação entre essas métricas pode ser utilizada tentando sempre manter o equilíbrio entre corretude e cobertura. Avaliações que privilegiam a precisão podem ser muito específicas e cobrirem poucos exemplos, enquanto as que privilegiam revocação podem ser muito genéticas e acertarem muito pouco. As métricas podem sempre ser adaptadas à necessidade do problema, mas podem também se tornar muito complexas para serem utilizadas na prática. Neste trabalho, utilizamos a métrica $F1$ para *fitness* do indivíduo, que é a média harmônica da precisão e revocação. Mais detalhes sobre essas métricas estão na Seção 4.5.

Ainda, se a base de dados utilizada é muito grande, a *fitness* dos indivíduos pode ser computada por conjuntos menores da mesma base, economizando tempo de processamento. Outra característica importante desta métrica é ter um alto fator de granularidade, caso contrário, o AG terá pouca informação sobre a qualidade dos indivíduos para guiar a busca. Um caso extremo é utilizar uma *fitness* que retorne apenas os valores 1 ou 0, correspondente a um indivíduo "bom" ou "ruim". Neste caso, o algoritmo seria incapaz de selecionar uma melhor solução entre várias boas retornadas.

3.4 Métodos de Seleção

A seleção é responsável por retornar um indivíduo de uma população para o propósito de reprodução. Essa escolha deve dar oportunidade a todos os indivíduos de serem escolhidos, mas privilegiar indivíduos com melhor *fitness*. Assim, indivíduos melhores terão maior probabilidade de terem suas informações passadas para a próxima geração.

Existem vários métodos de seleção [6], mas aqui destacamos três [37]: a seleção proporcional, a seleção por *ranking* e por torneio. A seleção proporcional privilegia a escolha dos melhores indivíduos dando a eles um peso proporcional à sua *fitness* em uma *roleta-russa* [40]. Assim, cada indivíduo tem uma posição na roleta com viés proporcional à sua qualidade na população. A simulação de um giro na roleta é realizada escolhendo um número aleatório sobre a soma total das *fitness* de todos os indivíduos da população (tamanho da roleta). Apesar de sua simplicidade, esse método tem algumas desvantagens [28]. Primeiro, ele só considera indivíduos com *fitness* positiva. Além disso, ele exige o cálculo da soma da qualidade de todos os indivíduos, penalizando o tempo de execução do algoritmo e dificultando a sua paralelização. Por último, AGs com esse tipo de seleção tendem a convergir muito rápido, já que os melhores indivíduos são muito privilegiados.

A seleção por *ranking* consiste em dois passos. Primeiramente, todos os indivíduos da população são ranqueados de acordo com a sua *fitness*. O *ranking* é montado de forma ascendente se o objetivo é minimizar a *fitness* ou decrescente se queremos maximizá-la. A seleção do indivíduo é realizada como a seleção proporcional, mas baseada no *ranking* criado ao invés do valor absoluto da *fitness*. Com isso, diferenças de magnitude de *fitness* são intencionalmente descartadas.

Por último, a seleção por torneio consiste em escolher randomicamente k indivíduos de uma população – onde k é um parâmetro chamado tamanho do torneio – que competem entre si. Esse torneio pode ser determinístico, quando o indivíduo com melhor *fitness* ganha, ou probabilístico, quando uma roleta-russa é executada apenas para esses k indivíduos.

Ambas as seleções por *ranking* e por torneio evitam algumas desvantagens da seleção proporcional. Dependendo dos parâmetros utilizados, essas duas seleções podem até ser aproximadamente equivalentes. Porém, o *ranking* ainda precisa ser executado sobre todos os indivíduos da população, uma operação de grande custo computacional. Ela pode ser evitada no torneio, onde k é relativamente pequeno em comparação ao tamanho da população. Por essas razões, a seleção por torneio é muito utilizada na literatura, e também será utilizada neste trabalho.

3.5 Operadores

As operações genéticas são as responsáveis pela evolução dos indivíduos de uma população entre uma geração e a próxima. Os indivíduos podem ser alterados através de operações visando explorar o espaço de busca a partir de soluções viáveis já encontradas.

São vários os tipos de operadores que podem ser implementados para um AG. Neste trabalho, focamos nos tradicionais operadores de cruzamento uniforme e de k pontos, e mutação de k pontos, além de dois operadores específicos para classificação, chamados de generalização e especialização [37].

3.5.1 Cruzamento

Baseando-se na reprodução sexuada dos seres, a operação de cruzamento realiza uma troca de informações entre os cromossomos de dois indivíduos (pais), resultando em outros cromossomos (filhos) com partes das características de cada um dos originais. Existem várias formas de cruzamento na literatura, entre eles o cruzamento de k pontos e o uniforme. Por padrão, dois pais distintos geram dois filhos distintos entre si.

O cruzamento de k pontos escolhe aleatoriamente k pontos nos indivíduos, formando $k-1$ sequências de genes (bits). Cada sequência não consecutiva entre os pontos é cruzada entre os indivíduos. O resultado são dois novos indivíduos que mantém partes inteiras de um cromossomo de indivíduos pais. Um exemplo de cruzamento de 2 pontos 2 e 4 (indicados por ||) pode ser visto na Figura 3.5.1.

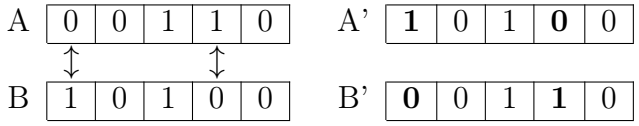
A	0	0	1	1	0	A'	0	**0**	**1**	**0**	0
B	1	0	1	0	0	B'	1	**0**	**1**	**1**	0

(a) Pais A e B (b) Filhos A' e B'

Figura 3.1. Exemplo de cruzamento entre os pontos 2 e 4 em um AG

O cruzamento de k pontos parece bem natural (ex: $k = 1$ é utilizado no cruzamento humano). Caso uma informação esteja codificada em uma sequência de bits do cromossomo, ele tem alta probabilidade de manter esses blocos de bits. Porém, quando os bits são independentes, cada bit deve poder ser variado e essas sequências deixam de ser importantes. Para isso, foi criada a forma de cruzamento chamada uniforme. O termo uniforme significa que qualquer bit do cromossomo pode ser "cruzado" separadamente. Ou seja, para cada uma das posições do vetor binário (regra) dos indivíduos, uma probabilidade é dada para o cruzamento ou não desse bit. Note que

uma probabilidade muito perto de 0 ou de 1 retorna filhos muito semelhantes aos pais. Enquanto probabilidades próximas a 0.5 geram cromossomos bem variados entre as características de cada pai. Na Figura 3.5.1 é exibido um cruzamento uniforme onde a troca de genes só ocorre nos pontos 1 e 4 do cromossomo.



(a) Pais A e B (b) Filhos A' e B'

Figura 3.2. Exemplo de cruzamento uniforme em um AG

Neste trabalho, implementamos o cruzamento uniforme, mantendo uma probabilidade fixa de cruzamento por bit de 50%.

3.5.2 Mutação

A mutação é uma operação que altera o cromossomo por algum estímulo externo, independente do cromossomo gerador. Diferente do cruzamento, que cria indivíduos baseados nas soluções já encontradas, um indivíduo gerado por mutação pode ser completamente distinto dos já encontrados. Assim, essa operação aumenta a divergência de uma população, podendo evitar a concentração da população em ótimos locais. Como o cruzamento, existe a mutação uniforme e a mutação por pontos.

Na mutação uniforme, assim como no cruzamento, uma probabilidade de mutação é dada para cada um dos bits do cromossomo. Assim, o número de bits mutados é variável, alterando o seu número proporcionalmente com o valor dado para a probabilidade de mutação. Uma probabilidade com valor pequeno cria indivíduos muito semelhante ao original, enquanto uma probabilidade a 1 pode criar um indivíduo completamente distinto.

Uma mutação por k pontos possibilita a variação de exatamente k bits aleatórios no cromossomo. Note que utilizar um k pequeno em relação ao tamanho do cromossomo é interessante no problema de classificação, já que uma pequena variação pode representar um grande ganho (ou perda) de informação. A Figura 3.5.2 exemplifica uma mutação realizada no quarto gene do cromossomo A, gerando um novo cromossomo A'.

Neste trabalho implementamos a mutação por um ponto, assim como em [67], pois retorna melhores resultados quando aplicada na representação escolhida do indivíduo

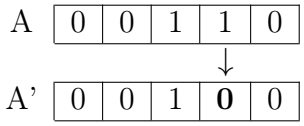


Figura 3.3. Exemplo de mutação de um ponto em um AG

para o problema de classificação. Quando um número maior de pontos é escolhido (como em uma mutação uniforme), regras muito grandes e complexas eram criadas, além de gerarem piores resultados por terem pouca cobertura.

3.5.3 Especialização e Generalização

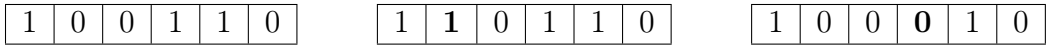
Enquanto a maioria dos trabalhos utilizam apenas os operadores tradicionais de cruzamento e mutação descritos anteriormente, operadores específicos para o problema a ser abordado podem ser utilizados visando enriquecer o processo de evolução ou até mesmo acelerá-lo.

Neste trabalho focamos nas operações de especialização e generalização, um tipo especial de operação sobre classificadores muito genéricos ou específicos, respectivamente [37]. A aplicação dessas operações assume que temos uma forma de detectar a cobertura de uma solução, visando determinar se uma solução candidata está muito específica ou muito genérica. No contexto de classificação, uma regra é considerada muito específica quando poucos exemplos satisfazem a regra. O inverso representa uma solução genérica, quando muitos exemplos satisfazem a regra, mas essas regras pertencem a um conjunto variado de classes.

Considere um indivíduo representado por uma regra ***se*** *contém atributo "bola"* ***ou*** *"placar"* ***então*** *"Esporte"*. Uma especialização deste indivíduo pode ser feita removendo-se um dos atributos da condição (antecedente) ou alterar o operador *ou* para *e*. Isto restringiria o número de exemplos que esta regra seria capaz de cobrir. Enquanto isso, uma generalização pode ser feita adicionando um novo atributo com o operador *ou*.

Um outro exemplo é a aplicação em uma regra representada por um vetor binário de atributos, onde um exemplo deve conter pelo menos um dos atributos ligados nesse vetor para que seja aceito pela regra. Dado um indivíduo 3.4(a), podemos realizar uma especialização 3.4(b) ligando o segundo atributo do vetor desse indivíduo, ou uma generalização 3.4(c), desligando o quarto atributo do mesmo indivíduo.

Ambas operações especialização e generalização foram implementadas no HCGA, como descrito na Seção 4.7.3.



(a) Ind. original (b) Ind. mais especifico (c) Ind. mais geral

Figura 3.4. Exemplos de operações específicas para o problema de classificação sobre um mesmo indivíduo

Tabela 3.1. Resumo dos trabalhos sobre algoritmos evolucionários para classificação tradicional nos últimos anos

Trabalho	Abordagem
Pietramala (2008) [67]	OlexGA: AG para classificação multi-classe com regras binárias
Chow (2008) [21]	AG utilizado em conjunto com SVM e *clustering*
Goldsby (2009) [41]	EA com classificação por vários indivíduos (*ensemble*)
Breaban (2009) [11]	AG para classificação e seleção de atributos
Vallim (2009) [93]	AG com regras para classificação multi-classe
David-Tabibi (2010) [25]	AG para seleção de pesos de atributos e classificação
Mishra (2011) [63]	AG paralelo multi-objetivo para regras de associação

3.6 Trabalhos relacionados

Algoritmos evolucionários já foram bem utilizados na classificação tradicional de várias formas e em diversas aplicações. Porém, não encontramos nenhum AE aplicado ao problema de classificação hierárquica. A Tabela 3.1 lista alguns dos últimos trabalhos publicados sobre AE, focando principalmente nos trabalhos relacionados à AG. A primeira coluna mostra as referências para cada trabalho, e a segunda apresenta informações sobre a abordagem utilizada no trabalho. Observamos que a utilização de regras e funções matemáticas continua sendo tradicional, e que muitos métodos híbridos ou que alinham seleção de atributos e classificação continuam em alta.

Dentre os trabalhos conhecidos de utilização de AE para classificação, destacamos o OlexGA [67], cuja simplicidade e eficiência foram utilizadas como referência para construção do modelo proposto neste trabalho.

3.6.1 OlexGA

Esta seção descreve a modelagem do algoritmo OlexGA, um algoritmo de representação simples que obtêm resultados relevantes na classificação tradicional de documentos, e cuja representação serviu de inspiração para a modelagem do algoritmo proposto neste trabalho.

O OlexGA [67] é uma implementação de um AG para classificação. Ele propõe uma representação de indivíduo bem simples, baseada na presença e ausência de atributos para classificar um exemplo de uma determinada classe. Assim, cada indivíduo representa um classificador binário (responde se um documento pertence ou não a uma classe). Apesar da simplicidade da modelagem, o algoritmo teve boa performance em sua aplicação a bases de texto *multi-label* comparada com vários algoritmos da literatura (Naive Bayes, Ripper, C4.5, SVM), chegando a superar os resultados de vários deles.

Primeiramente, o OlexGA propõe a representação de um classificador H_c (solução) para uma classe c no formato:

$$H_c(Pos, Neg) = (t_1 \in d \vee ... \vee t_n \in d) \wedge \neg(t_{n+1} \in d \vee ... \vee t_x \in d)$$

onde d é um exemplo com x atributos e cada t_i um termo $\{i \in 1, x\}$ de um vocabulário dado. $Pos = \{t_1, ..., t_n\}$ e $Neg = \{t_{n+1}, ..., t_x\}$ representam respectivamente o conjunto de termos positivos utilizados para dar cobertura e o conjunto de termos negativos utilizados para dar precisão sobre conjunto de exemplos de treinamento de c.

Assim, um indivíduo é formato por um vetor binário onde as n primeiras posições representam termos que agregam informação positiva e as últimas posições, agregam informação negativa, assim como Pos e Neg na representação do classificador descrito anteriormente. A seleção dos atributos a serem utilizados nos indivíduos é feita pela escolha dos melhores atributos retornados pela métrica, como a χ^2 ou $infogain$, definida pelo usuário para cada classe.

Sabe-se que o problema de aprender $H_c(Pos, Neg)$ é formulado como uma tarefa de otimização (aqui chamado de $MAX - F$) focado em encontrar os conjuntos Pos e Neg que maximizam a medida $F1$ quando $H_c(Pos, Neg)$ é aplicado ao conjunto de validação. $MAX - F$ pode ser representado como um problema combinatório 0-1, logo, a abordagem AG é uma candidata natural para resolver esse problema. O OlexGA é o algoritmo genético implementado em [67] para resolver $MAX - F$. Em [67] é demonstrado que $MAX - F$ é um problema NP-Completo.

Graças à simplicidade da representação do classificador, podemos representar o indivíduo como um classificador (em vez de apenas uma regra). Assim, a *fitness* desse indivíduo pode ser medida pela métrica $F1$ (Seção 4.5).

Dada a representação acima, o OlexGA retorna a melhor regra gerada pela evolução de um AG. Esse AG utiliza elitismo, tem a seleção feita pelo método de roleta-russa, e utiliza os operadores de cruzamento e mutação uniforme (descritos anteriormente). Ao final, cada nó da hierarquia de classes da base possui uma regra. Cada exemplo

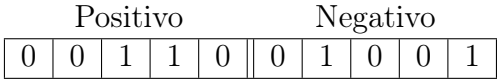


Figura 3.5. Exemplo de um indivíduo no OlexGA

é então avaliado junto a regra de cada classe. Caso a regra retorne positivamente ao exemplo, este é classificado pela classe prevista pela regra. O conjunto das classificações resulta na classificação multi-rótulo das bases utilizadas em [67].

Como visto posteriormente, este trabalho adota uma representação para o problema de classificação hierárquica. Detalhes na Seção 4.3.

Capítulo 4

Algoritmo Genético para Classificação Hierárquica

Este capítulo descreve o HCGA (*Hierarchical Classification Genetic Algorithm*) [14], um método proposto para obter soluções para o problema de classificação hierárquica. Como descrito no Capítulo 2, a maioria dos métodos de classificação hierárquica *constrói modelos de classificação de forma independente* para cada classe, cada nó pai ou cada nível da hierarquia. Mais tarde, durante a fase de teste, esses métodos utilizam a abordagem *top-down* para determinar as classes de novos exemplos.

Embora os métodos baseados nas abordagens acima descritas encontrem uma solução relativamente simples para o problema de classificação hierárquica, apenas a abordagem global ou *big-bang* pode ser considerada originalmente hierárquica. Isso porque o principal foco dos outros métodos não está em encontrar uma solução global levando em conta toda a informação sobre a hierarquia das classes, mas sim em resolver vários problemas menores localmente e, numa segunda fase, integrar os resultados obtidos. Porém, a combinação de soluções ótimas locais não garante a obtenção de um ótimo global [81].

O HCGA resolve esse problema fazendo com que informações dos modelos dos níveis mais altos da hierarquia (e, consequente, informações sobre a própria hierarquia) sejam passadas para os modelos dos níveis mais baixos, utilizando a abordagem *top-down* também na *construção* dos modelos. As próximas seções descrevem como o HCGA consegue trabalhar com informações locais e globais simultaneamente. Porém, inicialmente, apresentamos um histórico que descreve várias tentativas não tão bem sucedidas de modelos evolucionários para classificação hierárquica.

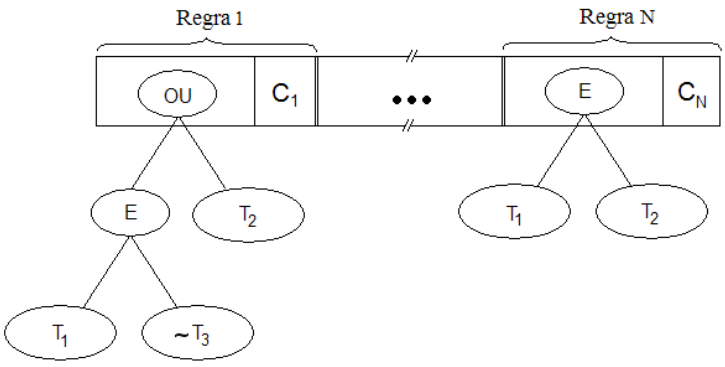


Figura 4.1. Indivíduo formado por regras, sendo cada regra uma árvore de decisão

4.1 Histórico

A partir de estudos sobre abordagens para se lidar com classificação plana e algoritmos evolucionários, e de como elas poderiam ser adaptadas para problemas de classificação hierárquica, foram sugeridas várias versões do método proposto neste trabalho antes de chegar a versão atual. Muitas dessas soluções se mostraram complexas e difíceis de serem analisadas, demandando muito tempo para serem implementadas e avaliadas.

Essa dificuldade inicial está principalmente ligada à representação utilizada para os classificadores e a forma de classificação de novos exemplos. Essas soluções se baseavam em modelos que tinham como principais características:

1. **Representação dos indivíduos**. Indivíduos eram formados por uma ou várias regras, sendo cada regra uma árvore de decisão associada a uma classe, como na Figura 4.1. Cada árvore de decisão era formada por operadores E e OU nos nós internos e, nos nós folhas, condições de presença ou ausência de atributos. Cada regra podia ser associada a uma classe diferente (independente de nível) e ter dependência com sua posição no indivíduo (sequência fixa para classificação) ou ser independente das outras (semelhante a um comitê). Quando diferentes classes podem ser associadas às regras, o indivíduo pode representar um modelo de classificação completo. Quanto mais regras em um indivíduo, mais informação é representada.

   Esse formato inerentemente complexo criou dificuldades de implementação, na aplicação dos operadores de cruzamento e mutação, além de implicar a necessidade do operador *pruning*, que consegue diminuir o tamanho da árvore removendo subárvores que não adicionem valor ao resultado.

Visando construir um modelo menos complexo, criamos uma nova representação baseada no trabalho [67], descrita em detalhes nas Seções 4.3 e 4.7.

2. **Classificação de novos exemplos** A classificação de novos exemplos na fase de teste utilizando o modelo proposto pode ser realizada de duas formas, dependentes da forma de cooperação dos modelos gerados para cada classe da hierarquia:

   a) **Classificadores por nó, nível ou classe pai**. Um modelo é produzido para cada nó da hierarquia de classes. Cada novo documento é classificado utilizando a abordagem *top down*.

   b) **Classificador único**. Um novo exemplo é classificado apenas uma vez pelo classificador resultante. Esse classificador é formado por um único conjunto de regras fixo para todas as classes, formado pelos modelos mais específicos, evoluídos pelas classes folhas. Em teoria, essa escolha de regras evita os modelos muito genéricos das classes intermediárias, que podem adicionar ruído à classificação por cobrir exemplos pertencentes a várias classes. Porém, resultados preliminares mostraram o contrário, especialmente em bases que possuem exemplos em nós internos da hierarquia.

Dada a complexidade do problema, uma abordagem mais simples acabou mostrando-se a mais eficaz na solução do problema: construção de indivíduos representados por apenas uma regra de vetor binário de atributos e classificação de novos exemplos realizados para cada nível da hierarquia. Além disso, indivíduos são criados para uma espécie (classe) específica, não podendo ter sua espécie alterada durante a evolução do algoritmo. Pela complexidade encontrada na avaliação de suas características, nenhuma das outras soluções persistiram no modelo resultante, e, por isso, não aprofundaremos em suas descrições neste trabalho. Porém, o conhecimento adquirido pelas dificuldades encontradas nas tentativas de desenvolvimento destas foi muito importante para um melhor entendimento do problema de classificação hierárquica. O resultado disto foi a proposta de um método mais simples e muito eficaz, o HCGA, descrito neste capítulo.

4.2 Descrição geral

Como qualquer método de classificação, o HCGA trabalha em duas fases. Na primeira fase, é criado um modelo de classificação sobre um conjunto conhecido de exemplos. Na segunda fase, o modelo anteriormente criado é utilizado para prever as classes de novos exemplos, cujas classes são desconhecidas. Em cada fase é utilizado

Algoritmo 4.1 HCGATreino

Parâmetro: c // Uma classe
Parâmetro: $Comite$ // Comitê gerado para a classe c
1: $subClasses$ = classes filhas da classe c
2: **se** $subClasses = \emptyset$ **então**
3: **retorne**
4: **fim se**
5: $pop \leftarrow Comite \cup$ CriaEspecies($subClasses$, $Comite$)
6: Evolui(pop)
7: $comite \leftarrow$ SelecionaComite(i, pop)
8: **para cada** i em $subClasses$ **faça**
9: HCGATreino(i, $comite_i$)
10: **fim para**

um conjunto disjunto de exemplos, retirados da base original mas mantendo a mesma proporção de exemplos entre as classes. Dois conjuntos são utilizados na primeira fase: o conjunto de *treino* para criação do modelo e o conjunto de *validação* para avaliar a qualidade de cada modelo gerado. O conjunto utilizado na segunda fase é chamado de *teste*.

O Algoritmo 4.1 ilustra a primeira parte do processo: a fase de treinamento (HCGATreino). Nesta fase, o principal problema é descobrir as melhores regras para cada classe da hierarquia utilizando um algoritmo genético (AG) semelhante ao AGPadrão ilustrado no Algoritmo 3.1.

O HCGATreino recebe como parâmetros uma classe c, inicialmente representada pela raiz da hierarquia, e um *comite*, que representa um conjunto de regras previamente geradas para a super classe de c, e inicialmente vazio. Para cada classe filha de c (linha 1), criamos uma espécie dentro do algoritmo genético para representá-la, e todas as espécies formam uma *população*. O conceito de espécies foi introduzido para que regras de uma mesma classe sofram apenas operações de cruzamento entre elas. Dentre os indivíduos inseridos em cada espécie, estão especializações das regras do comitê, geradas originalmente para a classe c. Em outras palavras, as regras das classes do nível 1 da hierarquia são especializadas para atender agora as classes filhas do nível 2, fazendo com que informações já adquiridas nos níveis menos profundos sejam passadas para os níveis mais profundos. Esse procedimento é descrito em detalhes na Seção 4.4.

Pop é então evoluída pelo AG, utilizando operações genéticas gerais e específicas para o problema de classificação, como descrito na Seção 4.7. Ao final na evolução (linha 7), as melhores regras (indivíduos) para cada subclasse são selecionadas através do método *SelecionaComite* (descrito na Seção 4.8). Em um próximo passo, apenas as regras do comitê específicas para uma classe i filha de c são utilizadas para criar

regras para o próximo nível da hierarquia (linhas 8-10), repetindo todo o processo até que não existam subárvores sem regras em toda a hierarquia (linhas 2-4).

Dada a forma como é treinado, o HCGATreino segue abordagem ⟨ SPP, NMLNP, T, LCN⟨LI⟩⟩, de acordo com a categorização de [81] apresentada na Seção 2.2. Ou seja, o algoritmo proposto segue uma estrutura em árvore (T) para estruturação das classes, com previsão por único caminho por nível de classe (SPP). Além disso, não obriga que as previsões sejam feitas até as classes folhas, suportando tanto a previsão de caminho parcial quanto o completo (NMLNP). Por fim, um modelo é treinado por nó da hierarquia (LCN), com a política LI de seleção de exemplos positivos e negativos.

A próxima fase do HCGA é o teste, onde são classificados exemplos de classe desconhecida. Nesse momento, todas as regras do comitê avaliam cada exemplo. Para determinar qual classe será associada a um exemplo, cada regra recebe um valor de *confiança*. A *confiança* é calculada com base no conjunto de treinamento e reflete todas as regras dentre todos os exemplos atribuídos a uma classe, quais pertenciam realmente à classe. Quando regras de mesma classe cobrem o exemplo, a confiança dessas regras é somada na previsão desta classe.

Como estamos utilizando bases onde exemplos podem ser atribuídos a apenas uma classe (*unilabel*), atribuímos ao exemplo a classe que tiver maior *confiança*. Observe que, dessa forma, o método pode ser facilmente estendido para trabalhar com problemas *multi classe*, apenas determinando um valor de confiança mínimo, onde todas as previsões que tiverem uma confiança maior que este mínimo terão suas classes associadas ao exemplo.

O classificador resultante, formado pela união dos comitês de cada classe da hierarquia, é utilizado para a classificação de novos exemplos. Para exemplificar o método, considere um domínio onde apenas quatro atributos t_1, t_2, t_3 e t_4 são avaliados, como na Figura 4.2. Como uma regra é composta por uma lista de atributos positivos e uma lista de atributos negativos, uma regra R_1 para a classe c_1 pode ser representada como $R_1 = \{0, 1, 1, 1, 1, 0, 0, 0\}$, ou seja, uma regra com os atributos t_2, t_3 e t_4 positivos e o t_1 negativo. Uma especialização possível da regra R_1 para a classe $c_{1.1}$, sendo esta subclasse de c_1, é $R_{1.1} = \{0, 0, 1, 1, 1, 0, 0, 0\}$, onde o atributo positivo t_2 foi removido. Outra opção seria tornar negativo algum atributo positivo.

4.3 Representação do Indivíduo

A representação de um indivíduo segue a abordagem Pittsburgh [83], onde o indivíduo corresponde a um classificador completo para uma classe (espécie). Assim, o

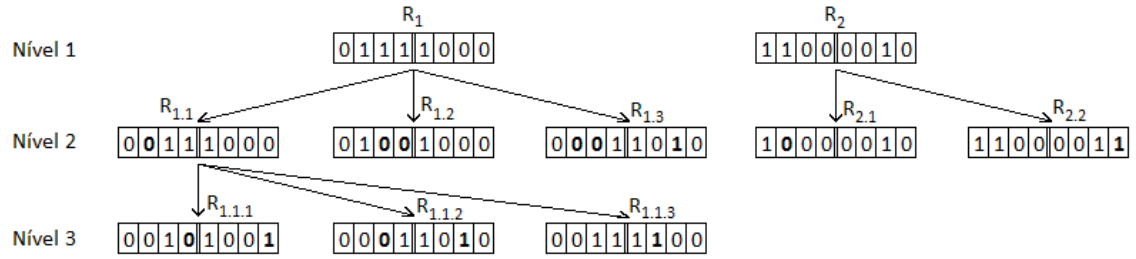


Figura 4.2. Um exemplo de classificações que poderiam ser produzidas pelo modelo proposto neste trabalho.

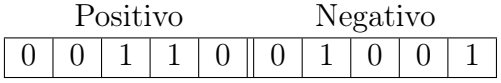


Figura 4.3. Exemplo de um indivíduo de uma classe

indivíduo do HCGA representa uma única regra, associada a uma única classe pré-definida. Cada regra segue uma representação simples, inspirada no modelo proposto em [67]: uma sequência binária, onde os k primeiros bits representam atributos presentes no exemplo, denominados positivos, e os últimos k bits representam atributos ausentes, denominados negativos. Cada bit pode ter o valor 0, quando o atributo não estiver ativo na regra, e 1 se ativo, sendo que um atributo não pode estar ativo como positivo e também como negativo ao mesmo tempo, e vice-versa.

A parte positiva do indivíduo necessita que pelo menos um dos atributos esteja presente no exemplo (implementando uma operação de *ou* lógico), enquanto a parte negativa funciona como um *e* lógico, obrigando que nenhum dos atributos do conjunto estejam presentes no exemplo. Um exemplo que cumprir as restrições do indivíduo pode ser classificado por sua respectiva classe.

Definindo formalmente, considere $T^+ = \{t_1, ..., t_n\}$ e $T^- = \{t_{n+1}, ..., t_{n+m}\}$ os n atributos positivos e os m negativos presentes em uma regra de uma classe C. Para um exemplo ser classificado como pertencente à classe C, a expressão c_i abaixo deve ser verdadeira:

$$c_i = (t_1 \epsilon d_i \vee ... \vee t_n \epsilon d_i) \wedge \neg(t_{n+1} \epsilon d_i \vee ... \vee t_{n+m} \epsilon d_i)$$

A Figura 4.3 exibe um exemplo de um indivíduo representando 5 atributos que podem aparecer ou não na base. Note que o valor de k é fixo, e para bases grandes uma fase de seleção de atributos é realizada para identificar os atributos mais informativos para a sua espécie, que então são considerados durante a pesquisa. O indivíduo da Figura 4.3 classifica exemplos que contenham os atributos 3 *ou* 4, e não contenham os atributos 2 *e* 5.

A seleção dos atributos a serem utilizados em cada indivíduo foi ponderada pelo peso normalizado de cada atributo pela métrica χ^2 utilizando a política LI de seleção de exemplos apresentada na Seção 2.2.

Avaliamos também uma forma de flexibilização dos atributos negativos, permitindo que o exemplo tenha apenas uma parte pequena dos negativos e mesmo assim seja coberto pela regra. Porém, essa flexibilização não foi interessante pois gera regras longas e mais difíceis de serem interpretadas (mais termos negativos). Isso acontece pois não há como identificar quais termos são utilizados e os que só estão atrapalhando a busca. Assim, neste trabalho não utilizamos a flexibilização dos negativos para realização dos experimentos apresentados, deixando para um trabalho futuro a melhor avaliação do impacto que essa alteração pode causar em uma modelagem como a apresentada.

Apesar de cada indivíduo representar uma regra, e portanto estar inserido na abordagem Pittsburgh, note que, ao adotar um comitê de indivíduos para classificação, essa abordagem é imediatamente abandonada, e passamos a seguir a abordagem Michigan (veja Seção 4.8).

4.4 Inicialização da População

Uma população é um conjunto de indivíduos (regras) representando classes de um mesmo nível, e dividida em espécies de acordo com cada classe. Em problemas de classificação em geral, uma inicialização aleatória da população não é recomendada, pois vários dos indivíduos gerados podem não classificar nenhum dos exemplos da base. Assim, neste trabalho inicializamos a população de três formas distintas, sendo uma delas válida apenas a partir do segundo nível da hierarquia.

No primeiro caso, como tradicionalmente feito na classificação plana, um exemplo é escolhido aleatoriamente do conjunto de *treinamento*, e seus atributos considerados na criação da nova regra. Os atributos pertencentes ao exemplo são todos setados como presentes no lado positivo da regra. Porém, como a extensão de atributos diferentes é muito grande, é dada uma probabilidade para cada um dos atributos ausentes no exemplo sejam ligados no lado negativo da regra, em vez de todos, na tentativa de evitar um possível *overfitting* de cobertura única do exemplo.

Em uma segunda forma, uma seleção de atributos é realizada pela métrica χ^2 [43], que resulta em um ranking de atributos mais relevantes para separação das classes. Logo, a probabilidade de ligar ou desligar um bit (atributo) no indivíduo é proporcional ao valor retornado pela métrica. Em outras palavras, quanto melhor o atributo é para

Algoritmo 4.2 CriaEspecies

Parâmetro: *subClasses* // Lista de classes filhas de uma classe c

1: $populacao \leftarrow \emptyset$
2: **para cada** i em *subClasses* **faça**
3: $popEspecie_i \leftarrow Comite_i$ // Indivíduos específicos para classe i
4: **se** $Comite <> \emptyset$ **então**
5: **repita**
6: $I \leftarrow$ indivíduo do $Comite_i$
7: $I' \leftarrow$ variação de I para classe i
8: $popEspecie_i \leftarrow popEspecie_i \bigcup I'$
9: **até que** % da $popEspecie_i$ atingida
10: **repita**
11: $I \leftarrow$ exemplo selecionado aleatoriamente da base de treino com classe i
12: $I' \leftarrow$ variação dos atributos de I
13: $popEspecie_i \leftarrow popEspecie_i \bigcup I$
14: **até que** % da $popEspecie_i$ atingida
15: **repita**
16: $I \leftarrow$ indivíduo vazio
17: $I' \leftarrow$ variação de I com atributos selecionados por χ^2 para classe i
18: $popEspecie_i \leftarrow popEspecie_i \bigcup I'$
19: **até que** $popEspecie_i$ completa
20: $populacao \leftarrow populacao \cup popEspecie_i$
21: **fim se**
22: **fim para**
23: **retorne** *populacao*

distinguir exemplos de diferentes classes, maior a probabilidade de ser ligado, e melhor é sua posição no ranking retornado.

Os dois procedimentos descritos podem ser abordados isoladamente ou em conjunto. Experimentos mostraram que a mistura de ambos melhoram a diversidade dos indivíduos (veja Seção 5.4).

Além das duas formas descritas acima, o HCGA conta com mais um procedimento na inicialização da população, utilizado apenas a partir do segundo nível da hierarquia de classes, e que é uma das contribuições desse trabalho. Enquanto no primeiro nível da hierarquia ainda não temos nenhuma informação a respeito da mesma, após a criação de regras para o primeiro nível, teremos um modelo de classificação que pode ser utilizado para gerar regras para os próximos níveis, já que ele incorpora conhecimento prévio da hierarquia.

Isso permite refinamentos sobre os resultados já obtidos em um nível para os mais específicos da hierarquia em vez de se recomeçar a pesquisa a cada nível. Esses refinamentos são feitos a partir de variações dos indivíduos do comitê, e se dão por variações

similares as geradas a partir de exemplos. Inicialmente, todos os indivíduos do comitê são inseridos na nova espécie. Depois, subconjuntos de atributos são aleatoriamente selecionados a partir desses indivíduos para criar variações.

O Algoritmo 4.2 descreve esse procedimento de inicialização da população. Como ilustrado, para cada subclasse, variações utilizando os diferentes métodos propostos são geradas, até que a população da espécie esteja completa. Após a população ser gerada, indivíduos são avaliados, como descrito na próxima seção.

4.5 Função de *Fitness*

Uma das grandes dificuldades na classificação hierárquica está na avaliação dos resultados. As métricas utilizadas para classificação plana ainda são muito utilizadas na literatura, mas às vezes revelam-se pouco adequadas quando analisadas sob uma ótica de hierarquias, deixando de lado informações que podem ser úteis para avaliação da previsão realizada [81].

O HCGA implementa duas métricas de avaliação de qualidade (*fitness*), uma para avaliação dos indivíduos (regras) durante a fase de construção dos modelos, utilizando a parte conhecida da base com chamada de *validação* e outra durante a classificação de novos exemplos, utilizando a base com rótulos desconhecidos, chamada de *teste*.

A *Fitness* de um indivíduo determina o quão apto ele é para resolver o problema sendo atacado. Neste trabalho, o indivíduo é tão bom quanto sua capacidade de classificar corretamente um conjunto de exemplos. Visando abordar essas características, foi implementada a métrica F_β (Eq. 4.1), sendo β um valor positivo qualquer. É atribuído a β um valor dentro do intervalo $[0, 1)$ para privilegiar a precisão (Equação 4.2), ou o intervalo $(1, \infty)$ para privilegiar a revocação (Equação 4.3). O valor 1, dá o mesmo peso às duas métricas.

Para realizar o cálculo da precisão (Pr) e revocação (Re) para cada classe, geramos uma matriz de confusão aplicando a regra do indivíduo a todos os exemplos pertencentes à classe ou a uma das classes irmãs. Para escolha de quais exemplos serão utilizados durante o treinamento, utilizamos a política LI (*less inclusive*), conforme a categorização de classificadores proposta por [81], apresentado na Seção 2.3.2.1. Neste caso, o conjunto de exemplos positivos de uma classe i consiste nos exemplos da classe i ou de seus descendentes, e o conjunto de exemplos negativos consiste no conjunto de exemplos restantes de toda a hierarquia que foram previstos como i no treinamento do nó pai dessa classe.

Na Tabela 4.1, encontramos uma matriz de confusão para um indivíduo da classe

i, sendo $\neg i$ = as classes diferentes de i. Nela armazenamos o número de exemplos da classe i (positivos) classificados corretamente na classe i (TP_i) ou erroneamente como uma classe diferente de i (FN_i), além dos exemplos de classe diferente de i (negativos) classificados erroneamente como pertencentes à classe i (FP_i) e corretamente como outras classes (TN_i).

Tabela 4.1. Matriz de confusão para uma classe i

		Classe Prevista	
		i	$\neg i$
Classe Correta	i	TP_i	FN_i
	$\neg i$	FP_i	TN_i

Com essas informações, podemos calcular facilmente a métrica desejada para uma regra de classe i pelas Equações 4.1, 4.2 e 4.3, onde F_{β_i}, Pr_i e Re_i variam entre o melhor valor 1 e o pior 0.

$$F_{\beta_i} = \frac{(1+\beta^2) \cdot Pr_i \cdot Re_i}{\beta^2 \cdot Pr_i + Re_i} \tag{4.1}$$

$$Pr_i = \frac{TP_i}{TP_i + FP_i} \tag{4.2}$$

$$Re_i = \frac{TP_i}{TP_i + FN_i} \tag{4.3}$$

Neste trabalho utilizamos vários valores para β, inclusive variando seu valor a cada nível. Porém, resultados mais estáveis foram alcançados quando o valor 1 era utilizado. Como em vários trabalhos na literatura, o objetivo é alcançar regras que tenham boa taxa de acerto em sua previsão, mas que também consigam cobrir o máximo de exemplos da classe possíveis nestas previsões. Por este ser um valor muito comum, a métrica F_β com $\beta = 1$, interpretada como a média harmônica da precisão e da revocação, é denominada na literatura apenas por $F1$ (Eq. 4.4).

$$F1_i = \frac{2 \cdot Pr_i \cdot Re_i}{Pr_i + Re_i} \tag{4.4}$$

Para avaliar o classificador completo utilizamos essa mesma métrica. É importante ressaltar que outras métricas poderiam ter sido avaliadas, como as apresentadas na Seção 2.4. Porém, preferimos utilizar uma métrica que já é bem reconhecida para avaliar algoritmos de abordagem *top-down* [49, 67, 78, 103].

4.6 Estratégia de *Niching*

Em problemas como o de classificação, a função de *fitness* utilizada para avaliar um indivíduo pode apresentar vários ótimos locais e até mesmo globais. Chamamos cada um desses ótimos de pico. Em algoritmos evolucionários, idealmente, a quantidade de indivíduos residindo na vizinhança de cada pico a função de *fitness* é proporcional à altura do pico. Ou seja, quanto maior um pico de *fitness*, maior o número de indivíduos próximos a esse pico. A cada um desses picos damos o nome de *nicho*. Quando todos os indivíduos fazem parte apenas de um nicho, dizemos que a população está convergindo. Observamos nos experimentos preliminares que a diversidade das espécies do nosso método não estava sendo preservada, convergindo rapidamente.

Com o objetivo aumentar a evolução dos indivíduos próximos a ótimos locais do espaço de busca, propondo a diversidade em vez de igualdade (convergência), implementamos uma solução chamada *niching*, baseada no padrão de *fitness sharing* [6]. Esta solução força uma maior dispersão dos indivíduos pelo espaço de busca em ótimos locais durante a evolução, ao invés de priorizar apenas um pico.

Os métodos de *niching* podem ser divididos em dois grandes grupos:

1. Compartilhamento de *fitness* (*fitness sharing*);

2. Agrupamento (*crowding*).

Dado que cada nicho corresponde a um pico da superfície de *fitness*, ou um ótimo local, utilizamos uma métrica chamada *fitness* compartilhado para penalizar os indivíduos semelhantes de cada pico.

O método de compartilhamento de *fitness*, introduzido por [40], é um método de escalonamento que altera apenas o estágio de especificação de *fitness* do algoritmo evolutivo.

O *fitness* compartilhado f_s de um indivíduo x_i é igual ao *fitness* f deste indivíduo dividido pelo *contador de nichos*, que é a soma dos valores das funções de compartilhamento sh entre este indivíduo e o restante da população.

$$f_s(x_i) = \frac{f(x_i)}{\sum_{j=1}^{N} sh(dist(x_i, x_j))}, \tag{4.5}$$

onde N é a quantidade de indivíduos da população, e sh (Equação 4.7) é uma função de compartilhamento de *fitness* proporcional à distância d entre dois elementos da população. A função de compartilhamento considera um *limiar de compartilhamento* (similaridade) σ_{share} tal que, se a distância entre dois indivíduos da população

é maior ou igual a σ_{share}, eles não afetam seus respectivos *fitness*. α é uma constante (geralmente $\alpha = 1$) capaz de regular a forma da função de compartilhamento [29].

$$sh(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} & \text{, se } d < \sigma_{share} \\ 0 & \text{, caso contrário} \end{cases} \tag{4.6}$$

Existem várias formas de se medir a distância entre os indivíduos, algumas delas baseadas em genótipo, outras em fenótipo. Um exemplo de distância por genótipo é pela presença ou ausência de cromossomos no indivíduo. Quanto mais cromossomos iguais, maior é a semelhança, ou seja, menor é a distância. Enquanto em indivíduos de cromossomos distintos, maior é a distância associada. Por outro lado, a distância baseada em fenótipo avalia as características que cada indivíduo apresenta no espaço de soluções, como a *fitness*. Nesse caso, a distância pode ser relacionada à diferença das *fitness* de dois indivíduos.

Como o objetivo deste trabalho é manter indivíduos que cubram diferentes partes do espaço de busca, simplesmente comparar os bits das regras de cada indivíduo não retorna nenhuma informação sobre o quão diferentes são, já que ambos podem cobrir os mesmos exemplos da base. Logo, definimos uma forma de calcular a distância entre dois indivíduos (regras) baseados na diferença entre os exemplos de validação cobertos por cada um (os mesmos utilizados para cálculo da *fitness*). Sejam dois indivíduos ind_1 e ind_2, onde ind_1 cobre o conjunto $Conj_1$ de exemplos e ind_2 cobre $Conj_2$, a distância entre eles é definida como:

$$dist(ind_1, ind_2) = 1 - \frac{||Conj_1 \cap Conj_2||}{||Conj_1 \cup Conj_2||} \tag{4.7}$$

Essa abordagem evita que muitos indivíduos com resultados (fenótipos) semelhantes ocupem o mesmo ótimo local, já que *dist* adquire valor 1 se os exemplos cobrirem os mesmos exemplos, e 0 se não cobrirem nenhum exemplo comum.

4.7 Operadores Genéticos

Depois de serem criados e avaliados, os indivíduos do HCGA são selecionados utilizando o método do torneio para sofrerem operações genéticas. Como já mencionado, as operações genéticas são as responsáveis pela evolução dos indivíduos de uma população. Os indivíduos podem ser alterados através de operações em seu cromossomo visando explorar o espaço de busca a partir de soluções viáveis já encontradas.

São vários os tipos de operadores que podem ser implementados para um AG. No HCGA, propomos o tradicional cruzamento uniforme e mutação de um ponto,

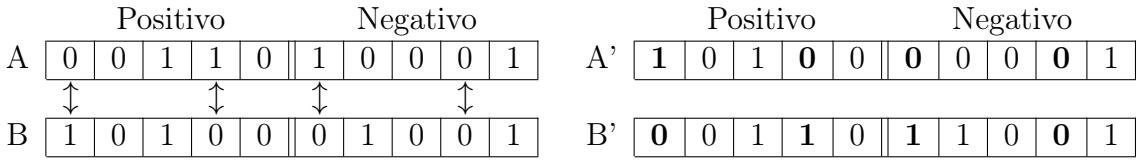
mas implementamos também operadores específicos para classificação, chamados de generalização e especialização.

Foram realizados testes experimentais de desempenho com a presença e ausência de cada um desses operadores para avaliar a necessidade deles no algoritmo. Os operadores de cruzamento são importantes para difundir a qualidade de cada indivíduo, enquanto a mutação é necessária para evitar a convergência de uma população. Resultado semelhante foi observado na utilização dos operadores de generalização e especialização, que permitem que pequenas variações nos indivíduos gerem grandes ganhos para a população. Uma análise mais criteriosa da influência de cada operador no algoritmo foi deixado para trabalhos futuros.

Sabe-se que um indivíduo de uma espécie pode sofrer uma variação tal que passe a ser inútil para a espécie atual, podendo ser melhor utilizado se inserido em uma outra espécie. Além disso, informações de indivíduos de espécies diferentes poderiam ser trocadas em um processo de co-evolução. Porém, neste trabalho, o indivíduo nunca terá sua espécie alterada através de qualquer operação genética. Dada a sua alta complexidade de implementação e avaliação, foi necessário deixar essa forma de evolução para trabalhos futuros.

4.7.1 Cruzamento

Existem várias formas de cruzamento, sendo as mais comuns a uniforme e a de um e dois pontos [37]. Neste trabalho, propomos o cruzamento uniforme, definido na Seção 3.5.1. Porém, dada a característica do indivíduo de ter um lado 'positivo' e um lado 'negativo', onde um atributo ativo como positivo não pode estar ativo como negativo e vice-versa, no HCGA a operação de cruzamento é realizada em apenas um dos lados, espelhando a operação do outro. Na Figura 4.4 podemos ver o resultado de uma operação que cruzou os pais A e B gerando os filhos A' e B', onde apenas os bits 1 e 4 ficaram invertidos em comparação com os pais.



(a) Pais A e B

(b) Filhos A' e B'

Figura 4.4. Exemplo de cruzamento uniforme no HCGA, com troca de informações dos bits 1 e 4

4.7.2 Mutação

O HCGA utiliza a mutação de um ponto. Um atributo qualquer do cromossomo é escolhido aleatoriamente e tem seu valor invertido. Se o bit estiver ativo, é então desativado. Porém, se estiver inativo, ele é ativado e tem o seu correspondente do lado oposto desativado. Ou seja, se o bit escolhido é positivo, mas o seu correspondente negativo já está ativo, ambos são invertidos, como na figura abaixo.

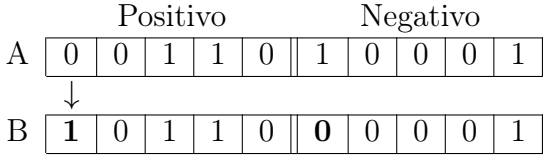


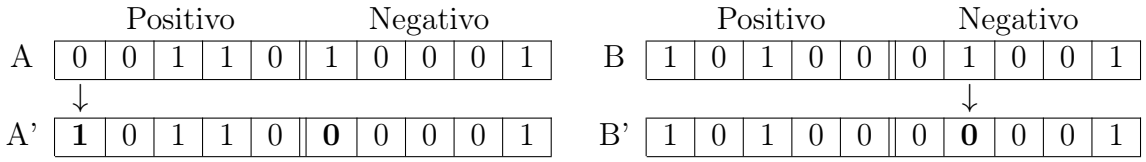
Figura 4.5. Exemplo de mutação do bit 1 positivo de um indivíduo no HCGA, ajustando também o lado negativo

4.7.3 Especialização e Generalização

Como o próprio nome sugere, o operador de generalização é aplicado a regras que são muito específicas, enquanto a especialização para regras muito genéricas. Como descrito anteriormente, no contexto de classificação, considera-se regras muito específicas aquelas em que um pequeno número de exemplos satisfazem a regra. O inverso representa uma solução genérica, quando muitos exemplos satisfazem a regra, mas não necessariamente a classe prevista.

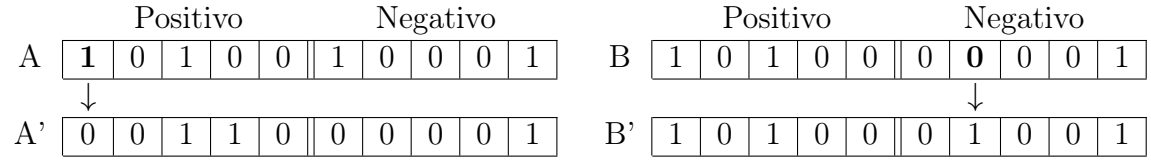
Considerando a representação dos indivíduos deste trabalho (vetor binário de atributos de presença, positivos, e ausência, negativos), foram implementadas operações específicas que se lidam com essas características. Assim, a operação de especialização pode remover atributos do lado positivo do cromossomo ou inserir novos atributos do lado negativo. O inverso dessa operação é a generalização, que liga atributos do lado positivo e desliga os do lado negativo.

A escolha do atributo a ser inserido ou removido de acordo com o objetivo desejado: especializar ou generalizar. Assim, a escolha do operador a ser utilizado depende da qualidade do indivíduo que será alterado. Caso o indivíduo já acerte todos os exemplos cobertos, apenas a operação de generalização é permitida. Caso grande parte dos exemplos cobertos esteja incorretamente classificado, escolhemos apenas a especialização. Porém, quando tanto a sua acurácia quanto sua revocação estiverem ruins, a escolha do operador passa a ser aleatória.



(a) Generalização ligando atributo positivo (b) Generalização desligando atributo negativo

Figura 4.6. Exemplos de generalização no HCGA



(a) Especialização desligando atributo positivo (b) Especialização ligando atributo negativo

Figura 4.7. Exemplos de especialização no HCGA

4.8 Seleção do Comitê

O comitê é o conjunto de regras que serão reunidas para aumentar a cobertura e dar uma confiança maior em relação a um exemplo pertencer a uma categoria. Devido à simplicidade da representação do indivíduo, a utilização de um comitê permite que um classificador cubra melhor o espaço de busca, decompondo o problema em problemas menores (um para cada regra) e, assim, consiga solucionar problemas mais complexos. Essa abordagem é semelhante a utilização de *ensembles* [41], muito utilizada na literatura para tratar esse difícil problema de classificação.

Neste trabalho, uma seleção de comitê de tamanho k pode ser feita de três formas:

1. Utilizando os k melhores indivíduos baseados na *fitness* (elitismo);

2. Utilizando o melhor indivíduo baseado na *fitness* seguidos dos $k-1$ indivíduos mais distantes do comitê já formado;

3. Utilizando os k melhores indivíduos baseados em *fitness sharing*, assegurando a diversidade.

O primeiro método assume que os melhores indivíduos são os melhores previsores, já que se destacaram durante toda a evolução do algoritmo. O ranking dos indivíduos é realizado sobre a *fitness* de cada indivíduo sobre a base de validação e os k primeiros são escolhidos para o comitê , como descrito no Algoritmo 4.3. Porém, é fácil visualizar que

Algoritmo 4.3 SelecionaComite *por fitness*

Parâmetro: *Classe* // Uma classe presente em *populacao*
Parâmetro: *populacao* // População gerada para uma classe interna da hierarquia
1: *especimes* ← indivíduos da espécie *Classe* presentes em *populacao*
2: *comite* ← k indivíduos com melhor *fitness* em *especimes*
3: **retorne** *comite*

Algoritmo 4.4 SelecionaComite *por mais distantes*

Parâmetro: *Classe* // Uma classe presente em *populacao*
Parâmetro: *populacao* // População gerada para uma classe interna da hierarquia
1: *especimes* ← indivíduos da espécie *Classe* presentes em *populacao*
2: *comite* ← indivíduo com melhor *fitness* presente em *especimes*
3: *lista* ← exemplos cobertos por *comite*
4: **while** *comite* incompleto **faça**
5: **para cada** i em *especimes* **faça**
6: calcula $dist(i, lista)$
7: **fim para**
8: *individuo* ← indivíduo de *especimes* que obteve maior *dist*
9: *lista* ← *lista* ∪ {exemplos cobertos por *individuo*}
10: *comite* ← *comite* ∪ *individuo*
11: **fim while**
12: **retorne** *comite*

dois indivíduos podem cobrir os mesmos exemplos, e, assim, não adicionam informação à classificação quando utilizados em conjunto.

O segundo método resolve o problema abordado acima. Inicialmente, escolhe o melhor indivíduo como o método anterior, gerando um comitê parcial. Depois disso, é realizado um simples procedimento guloso para a escolha dos próximos indivíduos, visando a maior cobertura pelo comitê parcial. A partir das regras do comitê parcial, uma lista com todos os exemplos já cobertos é criada. A partir dessa lista, calculamos a distância de cada indivíduo em relação a essa lista, de acordo com a métrica de distância definida na Eq. 4.7. O indivíduo mais distante é adicionado ao comitê. Realiza-se esse procedimento até que o comitê esteja completo (Algoritmo 4.4).

Finalmente, o terceiro método tenta selecionar os melhores indivíduos de diferentes nichos (máximos locais), visando prever com maior qualidade a maior quantidade de exemplos possível. Escolhe indivíduos até que o comitê esteja completo ou não existam mais nichos diferentes (Algoritmo 4.5). A separação entre nichos utiliza a mesma métrica definida em 4.6, mas podendo utilizar um σ_{share} distinto do utilizado na evolução.

Assim, o algoritmo pode ser classificado como ⟨ SPP, NMLNP, T, LCN⟨LI⟩⟩.

Algoritmo 4.5 SelecionaComite *por nichos*

Parâmetro: *Classe* // Uma classe presente em *populacao*
Parâmetro: *populacao* // População gerada para uma classe interna da hierarquia

1: *especimes* ← indivíduos da espécie *Classe* presentes em *populacao*
2: *comite* ← ∅
3: **while** *comite* incompleto **faça**
4: **para** i um indivíduo em *especimes* **faça**
5: $comite_i \leftarrow comite \cup i$
6: calcula $f_s(i)$ sobre $comite_i$
7: **fim para**
8: *individuo* ← indivíduo de *especimes* que obteve maior f_s
9: *comite* ← *comite* ∪ *individuo*
10: **fim while**
11: **retorne** *comite*

Capítulo 5

Experimentos

Este capítulo apresenta os resultados da aplicação do HCGA e de cinco outros algoritmos de classificação sobre um conjunto de bases de duas grandes áreas de classificação: bioinformática e texto. Em bioinformática, utilizamos quatro bases relativas à atividade GPCR (*G protein-coupled receptor*), que são de extrema importância para criação de novos medicamentos. Para classificação de documentos, utilizamos a base "20 Newsgroups", onde cada documento representa uma mensagem em um fórum de várias categorias. Ao final, uma análise é feita sobre a evolução do algoritmo proposto.

5.1 Bases de dados

Na área de bioinformática, utilizamos quatro bases baseadas na atividade GPCR (*G protein-coupled receptor* ou receptor acoplado à proteína G). GPCRs são essencialmente proteínas transmembrânicas (i.e., cruzam a membrana da célula) que transmitem sinais recebidos de fora da célula para dentro desta. Sinais diferentes ativam diferentes processos biológicos na célula, e GPCRs são envolvidos na transmissão de vários tipos de sinais. A identificação de proteínas GPCR e suas famílias é particularmente importante para aplicações médicas, dado que acredita-se que 40–50% dos medicamentos atuais atuam nas atividades GPCR [36].

Por esse motivo, utilizaremos para classificação de proteínas as bases do repositório UniProt [2] e GPCRDB [1], pelos grupos funcionais pfam, prosite, prints e interpro, aqui denominadas simplesmente como GPCRpfam, GPCRprosite, GPCRprints, GPCRinterpro. Nessas bases, cada exemplo representa uma proteína, e cada proteína é descrita como um conjunto de *motifs*. Um *motif* é um padrão de "assinatura" tipicamente encontrado em algumas proteínas. É basicamente uma sequência parcial ou completa de aminoácidos que podem ser utilizados para identificar a função e/ou

Tabela 5.1. Características das bases: número de classes por nível, número de atributos, número de exemplos e média de atributos por exemplo

Nome	#Classes por nvl	#Atrs.	#Ex.	Méd. atrs. por ex.
GPCRPrints	8/46/76/49 (180)	281	5422	2,2
GPCRPfam	12/52/79/49 (193)	73	7077	1,1
GPCRInterpro	12/54/82/50 (199)	448	7461	2,3
GPCRProsite	9/50/79/49 (188)	127	6261	2,1
20News	7/13/7 (28)	3000	18846	14,2

família de uma proteína [45]. Cada *motif* é representado por um atributo binário, que indica a presença ou ausência do *motif* em cada proteína. Esse tipo de representação proteica incorpora implicitamente um grande conhecimento sobre proteínas disponível na literatura, já que cada *motif* foi criado e refinado por um longo tempo por especialistas em biologia e bioinformática.

Nas bases GPCR, a classe a ser prevista é a função de uma proteína (SPL – *single path of labels*), que depende de qual tipo de molécula se conecta à parte do GPCR fora da célula e ativa o sinal de transmissão dentro da célula. As classes são organizadas em quatro diferentes níveis (primeira coluna da Tabela 5.1), e um exemplo pode ser classificado por classes de qualquer nível da hierarquia (PD – *partial depth labeling*). Os atributos a serem previstos variam para cada base, mas todos os quatro tipos de *motifs* utilizados como atributos (*motifs* Prints, Pfam, Interpro e Prosite [45]) representam assinaturas de proteínas.

Essas quatro bases são exatamente as mesmas descritas e utilizadas em [49], disponíveis em [4]. Porém, foram desconsiderados os atributos adicionais peso molecular e tamanho da sequência da proteína, numéricos. Esses atributos podem diferenciar proteínas muito similares, mas não podem ser avaliados dada a restrição da modelagem proposta neste trabalho, que exige atributos binários. Entretanto, uma comparação dos resultados apresentados em [49] e dos cinco métodos apresentados neste trabalho na Seção 5.4, mostra que não há perda significativa de informação sobre a acurácia obtida nas previsões, exceto sobre a base GPCRProsite e os últimos níveis da GPCRpfam. As bases resultantes utilizadas neste trabalho possuem cerca de 180 classes e mais de 5 mil atributos distribuídos muito esparsamente na base, como pode ser notado na Tabela 5.1.

Pelas caracterização de problemas de classificação hierárquica proposta por [81] e descrita na Seção 2.2, classificar as bases de bioinformática é um problema que pode ser descrito pela tupla $\langle$ T, SPL, PD $\rangle$, representando a hierarquia como uma árvore com classificação de caminho único e de profundidade parcial.

Já na área de documentos, utilizaremos a base de dados disponível em [3] pela Universidade da Califórnia e já utilizada em vários outros trabalhos [59, 70, 53, 58], conhecida como "20 newsgroups" (20News). Ela é constituída de 20 listas de discussão (*newsgroups*) organizadas hierarquicamente em classes e subclasses em 5 níveis de profundidade. A quantidade de classes diferentes por nível na base original é: 7 no primeiro nível, 16 no segundo nível, 10 no terceiro nível, 3 no quarto nível e 1 no quinto nível. Cada exemplo é representado por uma classe (SPL – *single path of labels*) do nível mais profundo da sua hierarquia (FD – *full depth labeling*). Note que há classes que apresentam apenas um classe filha, portanto todos os exemplos da superclasse são reclassificados na subclasse. Visando então evitar a criação de modelos desnecessariamente, desconsideramos essas subclasses que não adicionam informação à hierarquia durante o treinamento. A estrutura de classes depois desse corte de classes passa a contar com 28 classes distribuídas em 3 níveis de 7, 13 e 7 classes respectivamente, como pode ser visualizada na Tabela 5.1.

Na base 20News, cada classe contém aproximadamente 1.000 mensagens (documentos), de um total de 18846. Os documentos foram escritos em língua inglesa e contém cabeçalhos, linha de assunto, assinaturas e até mesmo citações de outros documentos, possuindo cerca de 114 mil termos distintos. Visando identificar apenas os termos que possam ser interessantes para a análise, a base foi submetida a um pré-processamento padrão, comum a bases de texto [7]: (i) remoção de caracteres especiais; (ii) remoção de *stop-words*; (iii) *steming*; e (iv) seleção de atributos.

Stop words (ou palavras de parada) são palavras que podem ser consideradas irrelevantes para o conjunto de termos a ser utilizado no espaço de busca por serem comuns a maioria das classes, tais como artigos e preposições, como "a", "about", "but", "if".

Em seguida, aplicamos o algoritmo de *steming* [68], disponível gratuitamente em várias implementações na Internet[1]. O objetivo desse algoritmo é fazer com que palavras que estejam sendo utilizadas com sufixos ou estão em diferentes tempos e formas verbais, sejam identificadas por seus radicais. Um exemplo são as palavras "messed", "messing", "messes", que serão identificadas pelo mesmo termo "mess".

Após o pré-processamento inicial, continuamos com cerca de 84 mil termos distintos. Assim, removemos também termos que ocorrem em apenas um dos documentos de toda a base. Com isso, reduzimos o número de termos pela metade, para 43 mil.

É importante notar que, apesar de atributos n-ários, compostos por conjuntos de n termos sequenciais (conhecidos na literatura como *n-grams*), serem propostos na

[1] http://www.tartarus.org/ martin/PorterStemmer, acessado em 21/07/2010

literatura, neste trabalho consideramos como *atributos* cada um dos termos da base de documentos ($n = 1$). Isto se deve ao fato que a análise da variação do valor de n não ser o foco deste trabalho, e que o número já muito grande de atributos presentes na base 20News ainda seria multiplicado em n vezes, aumentando muito o custo de execução de cada um dos algoritmos e da análise dos seus resultados. Mais informações sobre a variação de n em outras bases de texto podem ser encontradas em [67].

Vemos que uma base de texto continua muito grande (em termos de atributos) mesmo após o pré-processamento. Assim, foi necessário realizar ainda um corte de dimensionalidade na base, utilizando *seleção de atributos*. Existem métricas de seleção de atributos que consideram a hierarquia de classes, como a proposta em [15]. Porém, os resultados adquiridos com a utilização dessa métrica foram ruins se comparados a métodos tradicionais de seleção de atributos. Uma análise inicial mostrou que os atributos selecionados eram os mais frequentes em toda a base, e não os mais discriminativos para cada classe.

Assim, como no método OlexGA, apresentado na Seção 3.6.1, para a seleção de atributos deste trabalho executamos o método χ^2 disponível na ferramenta Weka. Esse método retorna um *ranking* dos atributos com maior importância de separação entre as classes. Em seguida, selecionamos um corte para esse *ranking*, que representa o número máximo de termos no topo do *ranking* que são considerados para a classificação da base. Para cada corte, executamos o Naive Bayes, um dos algoritmos com melhor resultado em classificação de documentos e disponível na mesma ferramenta, para avaliar a perda de informação associada a cada corte.

Ao final, chegamos ao corte de 3000 termos para esta base. Note que pela restrição da modelagem do HCGA os valores dos atributos da base de texto foram binarizados, ou seja, a base contém apenas a informação se um atributo ocorre em um exemplo da base ou não. Com isso, informações sobre a frequência de ocorrência dos atributos nos exemplos são descartados, e consideramos apenas sua presença e ausência. Assim, todos os experimentos apresentados da base 20News foram executados com esse número de atributos. Uma análise mais detalhada sobre a seleção de atributos em um problema hierárquico é deixado para trabalhos futuros.

Dadas essas características da base 20News, pela caracterização de problemas de classificação hierárquica proposta por [81] e descrita na Seção 2.2, classificar essa base de texto é um problema que pode ser descrito pela tupla $\langle$ T, SPL, FD $\rangle$.

A Tabela 5.1 descreve algumas particularidades de cada base utilizada, facilitando observar as principais diferenças entre bases de bioinformática e de texto. Enquanto as GPCRs possuem uma distribuição de atributos muito esparsa (de 1 a 2,3 atributos por exemplo), a 20News possui uma média de atributos bem maior. Com isso, são

necessários menos atributos em uma regra para classificar uma base de bioinformática, gerando soluções mais simples. Além disso, como o número de exemplos por classe na base 20News é muito maior, quanto mais regras utilizarmos para cada classe, menos complexas cada regra necessita ser.

Todas as bases foram divididas em cinco partições independentes de tamanhos semelhantes, sendo que a proporção entre as classes para cada partição foi mantida como na base original. Para a execução dos experimentos com o HCGA, as três primeiras partições foram utilizadas na construção do classificador (treino) e uma na avaliação da qualidade do classificador (*fitness*) gerado durante sua construção (validação), enquanto a última foi utilizada apenas para avaliação do classificador final retornado (teste). Todos os dados seguem um formato padronizado *.arff* para matrizes esparsas em dois arquivos: um com o conjunto de treino e validação (divididos novamente em tempo de execução) e um com o conjunto de teste. Este padrão é o mesmo utilizado pelo Weka [43], ferramenta muito utilizada na área de mineração mantida pela Universidade de Waikato.

O motivo da divisão em partições é para realização da validação cruzada de cinco partições. Ela consiste na divisão estratificada da base em partições mutualmente exclusivas, e em cada uma das execuções, uma partição diferente é utilizada para teste, validação e treino. A mesma divisão é mantida para os experimentos sobre os algoritmos de referência (*baselines*). Porém, por ser um algoritmo não determinístico, o HCGA é executado cinco vezes (aqui chamadas de *repetições*), enquanto os métodos de comparação apenas uma vez.

5.2 Métodos de comparação

Além de comparar o método proposto com outros algoritmos estado da arte da área de pesquisa abordada, alguns outros frequentemente utilizados em estudos de classificação hierárquica [49] também foram utilizados. Os algoritmos considerados foram: Naive Bayes, Bayes Net, J48 e Hyper Pipes. Os dois primeiros são baseados no teorema de Bayes [52], o J48 em árvores de decisão [72] e Hyper Pipes baseado simplesmente na semelhança dos atributos dos exemplos de uma mesma classe. Além destes, comparamos também com o método que inspirou a modelagem para este trabalho, o OlexGA [67]. Todos os cinco classificadores tem implementações disponíveis ou compatíveis para a ferramenta Weka [43].

Note que todos esses métodos utilizados como comparação são métodos de classificação planos. Uma vez que o algoritmo proposto usa estratégias *top-down* para treino

e teste, ele também é comparado com algoritmos que seguem a mesma abordagem. Para serem utilizados em uma classificação hierárquica, é necessário que treinemos um classificador para cada classe da hierarquia. Para isso, durante a fase de treino, treinamos cada classificador em cada classe da hierarquia utilizando as bases de treinamento e validação (a caracterização da abordagem é descrita a seguir). O resultado para cada nível é a probabilidade de previsão para cada uma das classes sobre todos os exemplos de teste de acordo com a certeza dada pelo próprio Weka para cada classificador.

Algoritmo 5.1 TD

Parâmetro: *exemplo* // Exemplo de teste
Parâmetro: *modelos* // Modelos de classificação plana para cada classe da hierarquia
Parâmetro: *classe* // Classe inicial
1: $c' \leftarrow modelos_{classe}(exemplo)$ // classificação
2: **se** c' é folha **e** $confianca(c') > confianca\_minima$ **então**
3: **retorne** $TD(exemplo, modelos, c')$
4: **caso contrário**
5: **retorne** c'
6: **fim se**

Dado esse conjunto de modelos por classe, na fase de teste o método *top-down* (TD), descrito no Algoritmo 5.1, foi utilizado. O TD recebe como entrada cada um dos *modelos* retornados e o *exemplo* a ser classificado. Partindo de uma *classe* raiz (geral) do primeiro nível da hierarquia, o TD prevê a classe c' no próximo nível da hierarquia para *exemplo* (linha 1). Se a classe prevista c' tiver subclasses e a confiança atribuída pelo *modelo* for maior que um limiar dado pelo usuário (linha 2), uma nova classificação é feita para o próximo nível pelo modelo de classificação previamente, dada a classe prevista (linha 3). Quando essa restrição é atingida (linha 4), não são realizadas mais classificações, e a classe prevista c' é retornada como o resultado de classificação para o *exemplo* de entrada (linha 5). Executando esse algoritmo para todos os exemplos de teste, podemos obter a acurácia por nível da hierarquia para o modelo de cada classificador, como observamos na Seção 5.4. A categorização do algoritmo TD, independente do *baseline* utilizado é ⟨ SPP, NMLNP, T, LCN⟨ S ⟩⟩, ou seja, previsão por único caminho por nível de classe (SSP) suportando tanto a previsão de caminho parcial quanto o completo (NMLNP) utilizando por uma estrutura em árvore (T) onde um modelo é treinado por cada nó da hierarquia (LCN) na política de irmãos (S) de seleção de exemplos.

Tabela 5.2. Parâmetros de entrada

Parâmetro	Padrão	Descrição
-g	50	Número de gerações
-i	50	Número de indivíduos por espécie
-c	0,9	Probabilidade de cruzamento
-m	0,4	Probabilidade de mutação
-e	0,4	Probabilidade de especialização e generalização
Elitismo	0,1	Percentagem dos melhores indivíduos de uma população copiada para a próxima geração
Torneio	2	Número de indivíduos para torneio da seleção de indivíduos

5.3 Parâmetros

Após o pré-processamento das bases, foram executados experimentos preliminares para escolher um conjunto de parâmetros para o HCGA. A Tabela 5.2 exibe os parâmetros necessários para execução do método e os valores utilizados na execução de todos os experimentos aqui demonstrados nas diferentes bases. Esses parâmetros foram obtidos em experimentos preliminares, e ainda podem ser otimizados para cada uma das bases.

Além dos parâmetros básicos de entrada, existem parâmetros que podem influenciar o comportamento do HCGA que não foram variados para este trabalho. São parâmetros de inicialização da população, variação entre os operadores de evolução e métricas de avaliação. Uma avaliação da influência desses parâmetros em cada uma das bases foi deixada para trabalhos futuros.

5.4 Resultados Experimentais

Esta seção apresenta os resultados encontrados ao utilizar o HCGA nas quatro bases de bioinformática relacionadas à atividade GPCR, aqui denominadas GPCRPrints, GPCRPfam, GPCRinterpro e GPCRProsite, e na base de texto "20 Newsgroups", aqui chamada apenas de 20News. Todos os resultados foram executados mantendo os valores padrões dos parâmetros e utilizando validação cruzada de cinco partições com cinco repetições (média de 25 execuções).

Para cada repetição realizada no HCGA, foi utilizada uma semente diferente de entrada do algoritmo, com valores 1, 2, 3, 4 e 5. Essa alteração é importante para medir como o algoritmo se comporta utilizando escolhas diferentes durante a sua evolução, já que ele é não determinístico. O desvio padrão relatado após todas as execuções

Tabela 5.3. Resultados obtidos para base GPCRprints

Algoritmo	Nível 1	Nível 2	Nível 3	Nível 4
HCGA	90.91 ± 0.78	76.87 ± 1.71	**55.97** ± 1.25	89.39 ± 0.73
OlexGA	**91.74** ± 0.35 ●	77.52 ± 0.81 ●	49.52 ± 0.64 ▲	77.60 ± 1.56 ▲
NB	90.93 ± 0.53 ●	76.18 ± 0.32 ●	52.52 ± 0.44 ▲	**90.63** ± 0.33 ▼
BayesNet	90.47 ± 0.39 ●	75.21 ± 0.60 ●	48.01 ± 0.22 ▲	69.08 ± 0.61 ▲
J48	91.63 ± 0.42 ●	**80.23** ± 0.19 ▼	54.86 ± 0.58 ●	80.64 ± 0.98 ▲
HyperPipes	89.84 ± 0.87 ●	14.50 ± 1.04 ▲	6.13 ± 1.14 ▲	7.92 ± 2.97 ▲

pode provar que o algoritmo realmente converge em qualquer condição, ou se consegue encontrar boas soluções apenas quando tem sorte na escolha da semente.

As Tabelas 5.3 a 5.6 apresentam os resultados da $F1$ obtido nos experimentos para as bases de bioinformática, e a Tabela 5.7 para a base 20News. A primeira coluna representa os algoritmos executados neste trabalho: o HCGA, e os cinco *baselines*, nominalmente OlexGA, NB, BayesNet, J48, HyperPipes utilizando a abordagem *top-down* como descrito na Seção 5.2. As próximas colunas apresentam os resultados encontrados por cada método para cada um dos níveis da hierarquia, utilizando a média seguido do símbolo ± e o desvio padrão sobre todas as execuções (validação cruzada e repetições). Para melhor visualização, a melhor média encontrada em cada nível é destacada em negrito.

Note que, neste trabalho utilizamos uma métrica plana para retornar a qualidade de classificação para cada nível da hierarquia. Assim, para classificações parciais, propagamos o erro para a classe certa do exemplo, ou seja, caso uma previsão c' seja realizada apenas até o segundo nível de uma classe de quarto nível c, os dois últimos níveis terão sempre uma classificação incorreta. Porém, caso a classe certa c seja do primeiro nível, nenhuma penalização é feita caso o classificador tenha previsto alguma classe c' descendente desta classe c. Também é importante ressaltar que, como detalhado na Seção 2.4, embora métricas de classificação plana podem deixar de lado algumas informações sobre a hierarquia das classes, elas são muito eficazes quando utilizadas para identificar a qualidade das classificações para cada nível da hierarquia.

Para cada *baseline*, os resultados são também seguidos por um símbolo que indica se os resultados são estatisticamente significativos de acordo com o t-test pareado de duas caldas, utilizando um nível de confiança de 95%. O símbolo ▲ implica que o HCGA é significativamente melhor que o método representado na linha, ● representa uma variação não significativa e ▼ uma variação significativamente pior do que o HCGA sobre o método sendo considerado.

Analisando os resultados apresentados na Tabela 5.3, observamos que para o nível

Tabela 5.4. Resultados obtidos para base GPCRpfam

Algoritmo	Nível 1	Nível 2	Nível 3	Nível 4
HCGA	92.24 ± 0.67	47.27 ± 1.28	12.47 ± 0.24	0.00 ± 0.00
OlexGA	**92.86** ± 0.20 ●	24.33 ± 0.26 ▲	7.18 ± 0.17 ▲	**1.19** ± 0.15 ▼
NB	92.36 ± 0.25 ●	46.80 ± 0.86 ●	12.82 ± 0.29 ●	0.00 ± 0.00 ●
BayesNet	91.83 ± 0.49 ●	46.82 ± 0.81 ●	12.70 ± 0.25 ●	0.00 ± 0.00 ●
J48	92.83 ± 0.22 ●	**48.01** ± 0.92 ●	**12.94** ± 0.21 ▼	0.00 ± 0.00 ●
HyperPipes	92.34 ± 0.29 ●	22.90 ± 0.26 ▲	6.78 ± 0.12 ▲	**1.19** ± 0.15 ▼

1 todos os classificadores são estatisticamente iguais, enquanto que no segundo nível o J48 é estatisticamente melhor, obtendo uma acurácia de 80%, contra os 76-77% dos outros classificadores. Quando aprofundamos nos níveis da hierarquia, a classificação se torna mais difícil, pois temos mais classes e menos exemplos. Em GPCRPrints, por exemplo, o primeiro nível tem 8 classes, enquanto o terceiro nível tem 76. No terceiro nível, os melhores resultados são obtidos pelos métodos HCGA e J48. Todos os outros métodos têm valores estatisticamente piores de $F1$. Para o quarto e último nível, HCGA é melhor do que todos os classificadores exceto o Naive Bayes, que obtém um ganho estatisticamente significativo, mas muito pequeno. Assim, em geral, se tivéssemos que escolher um classificador para prever a função em todos os níveis, o J48 iria apresentar vantagens sobre o segundo nível. Já no terceiro nível, o HCGA ou o J48 seriam os classificadores mais indicados, já que eles são estatisticamente melhores que todos os outros algoritmos e estatisticamente equivalente apenas com o J48. No entanto, no quarto nível, o que geralmente é aquele que traz informações mais interessantes para o usuário, o J48 apresenta um $F1$ de 80.64, enquanto HCGA atinge 89.39 e o NB 90.63%. Desta forma, o HCGA seria também uma boa escolha. Assim, não existe um classificador consistentemente bom em todos os níveis.

É importante ressaltar que os resultados apresentados aqui podem ainda ser melhorados. Considerando a forma como funciona HCGA, pequenas modificações e otimizações de parâmetro feitas nível a nível poderiam melhorar significativamente seus resultados. Isso é deixado para trabalhos futuros, já que em um primeiro momento, queríamos mostrar que a abordagem proposta pode ser comparada a outros algoritmos estado da arte. Além disso, o HCGA tem a vantagem de gerar modelos que podem ser facilmente interpretados, que dizem respeito apenas a presença ou ausência de um atributo (neste caso, um *motif* de proteína) em uma classe.

Em relação ao conjunto de dados GPCRPfam, cujos resultados são apresentados na Tabela 5.4, no primeiro nível, mais uma vez todos os algoritmos apresentam desempenhos similares. Isto é esperado, pois estamos lidando com classificadores planos, e

Tabela 5.5. Resultados obtidos para base GPCRinterpro

Algoritmo	Nível 1	Nível 2	Nível 3	Nível 4
HCGA	88.33 ± 0.47	**91.71** ± 0.50	48.69 ± 2.15	89.67 ± 0.72
OlexGA	**90.20** ± 0.24 ▼	74.02 ± 0.46 ▲	49.73 ± 1.21 •	82.15 ± 0.67 ▲
NB	90.11 ± 0.27 ▼	79.05 ± 0.38 ▲	52.88 ± 0.94 ▼	89.32 ± 0.60 •
BayesNet	89.40 ± 0.24 ▼	73.59 ± 0.73 ▲	47.11 ± 0.72 •	88.15 ± 0.66 ▲
J48	90.14 ± 0.29 ▼	78.91 ± 0.44 ▲	**53.66** ± 1.22 ▼	**92.58** ± 0.60 ▼
HyperPipes	78.33 ± 3.33 ▲	13.30 ± 3.66 ▲	4.34 ± 1.05 ▲	1.49 ± 0.20 ▲

prever o primeiro nível não é muito diferente de resolver um modelo de classificação tradicional. Para o segundo nível, no entanto, HCGA e outros três classificadores ainda apresentam um bom desempenho, sendo estatisticamente iguais. No terceiro nível, o HCGA (12.47%) não é estatisticamente melhor que o J48 (12.94%), apesar de um valor absoluto bem próximo, mas seus resultados são equivalentes aos obtidos pelas abordagens Bayesianas. O último nível pode ser previsto com acurácia muito pequena por OlexGA e HyperPipes, enquanto os outros classificadores não podem gerar qualquer modelo de classificação. É interessante ressaltar que o HCGA é melhor do que OlexGA nos níveis 2 e 3, mas que ele perde algo no último nível. Ao investigar por que isso aconteceu, percebemos que este é um resultado da propagação de erros. A maioria dos classificadores não consegue distinguir classes 001.002 e 001.005, pois os valores dos atributos dessas classes são praticamente os mesmos. Esse erro, ainda no segundo nível, impossibilita o acerto nos níveis mais específicos, já que se trata de classificadores com abordagem *top-down*. Apesar da abordagem de construção ser *top-down*, ela não evita a propagação de erros. Tentativas de evitar essa propagação mantendo essa abordagem são deixadas para trabalhos futuros.

Concluindo, na base GPCRPfam o HCGA é competitivo com outros métodos. No entanto, note que este tipo de *motif* é menos indicado para prever proteínas em níveis mais baixos da hierarquia, quando comparados com os resultados obtidos com os conjuntos de dados baseados em Prints.

Ao olhar a Tabela 5.5, o HCGA não se comporta tão bem como nas duas bases de dados anteriores. Neste caso, a classificação no primeiro nível é estatisticamente inferior aos de quatro das cinco baselines, apesar da perda ser pequena. No entanto, no segundo nível o HCGA se recuperou, tendo resultados estatisticamente melhores do que todos os outros classificadores. No entanto, essa situação se inverteu novamente no nível 3 e 4. No nível 3 o HCGA ainda tem resultados equivalentes aos de Redes Bayesianas e OlexGA, mas piores que J48 e Naive Bayes. Já no quarto nível, o J48 é melhor que todos os outros métodos, seguido pelo HCGA.

Tabela 5.6. Resultados obtidos para base GPCRprosite

Algoritmo	Nível 1	Nível 2	Nível 3	Nível 4
HCGA	84.04 ± 0.22	46.71 ± 0.93	**19.27** ± 0.59	**44.42** ± 0.91
OlexGA	84.25 ± 0.20 ●	27.57 ± 0.36 ▲	13.26 ± 0.33 ▲	12.70 ± 0.64 ▲
NB	83.88 ± 0.25 ●	46.55 ± 0.64 ●	18.55 ± 0.43 ●	44.40 ± 1.01 ●
BayesNet	84.13 ± 0.23 ●	46.26 ± 0.67 ●	18.43 ± 0.65 ●	41.74 ± 1.08 ▲
J48	**85.08** ± 0.17 ▼	**47.19** ± 0.76 ●	18.88 ± 0.52 ●	41.85 ± 1.16 ▲
HyperPipes	83.61 ± 0.12 ▲	21.24 ± 0.27 ▲	10.91 ± 0.30 ▲	12.21 ± 0.55 ▲

Tabela 5.7. Resultados obtidos para base 20News

Algoritmo	Nível 1	Nível 2	Nível 3
HCGA	74.41 ± 0.85	75.28 ± 1.77	79.24 ± 4.16
OlexGA	67.21 ± 0.71 ▲	74.76 ± 0.76 ●	90.58 ± 0.49 ▼
NB	**89.28** ± 0.32 ▼	**87.31** ± 0.44 ▼	93.71 ± 0.63 ▼
J48	80.88 ± 0.46 ▼	74.78 ± 0.55 ●	87.08 ± 0.74 ▼
HyperPipes	6.76 ± 0.25 ▲	75.48 ± 4.27 ●	**96.26** ± 1.56 ▼

Os resultados da GPCRProsite, apresentados na Tabela 5.6, mostram que o HCGA parece ser o melhor método para fornecer resultados para o último nível hierárquico. Embora no primeiro nível apresente uma pequena perda para o J48, nos seguintes níveis o HCGA é competitivo com os algoritmos do estado da arte, e depois, no último nível, apresenta resultados estatisticamente melhores do que aqueles produzidos por todos os outros métodos. Esta foi a base em que o algoritmo HCGA obteve resultados estatisticamente melhores ou iguais sobre todos os baselines na maioria dos níveis da hierarquia. Isto mostra que os algoritmos não são consistentes sobre todas as aplicações, mas o HCGA se comporta bem em várias das bases testadas.

Finalmente, na Tabela 5.7, são exibidos os resultados para a base 20News, uma base de texto, onde o número de exemplos e de termos em cada exemplo são bem maiores quando comparada às bases de bioinformática, aumentando muito o custo computacional de uma execução do algoritmo. Por isso, o algoritmo BayesNet, que utiliza muita memória em sua execução, ultrapassou os limites dos recursos disponíveis na máquina utilizada para esses experimentos e não gerou resultados para tal base. O HCGA também não obteve grandes ganhos com os parâmetros utilizados. Enquanto no primeiro nível o HCGA obteve melhores resultados apenas sobre o OlexGA e o HyperPipes, no segundo nível ele foi estatisticamente pior apenas do que o NB. Note que, apesar do HCGA ser baseado na representação do indivíduo do OlexGA, o HCGA foi construído utilizando novos operadores, tais como especialização e generalização, e com seleção de indivíduos utilizando *fitness sharing*, o que gerou melhores resulta-

Tabela 5.8. Resumo dos resultados obtidos pelo HCGA

GPCR	GPCR												20News			Total		
	Prints			PFam			Interpro			Prosite								
	▲	●	▼	▲	●	▼	▲	●	▼	▲	●	▼	▲	●	▼	▲	●	▼
L1	0	5	0	0	5	0	1	0	4	1	3	1	2	0	2	2	13	5
L2	1	3	1	2	3	0	5	0	0	2	3	0	0	3	1	10	9	1
L3	4	1	0	2	2	1	1	2	2	2	3	0	0	0	4	9	8	3
L4	4	0	1	0	3	2	3	1	1	4	1	0	0	0	0	11	5	4
Total	9	9	2	4	13	3	10	3	7	9	10	1	2	3	7	34	38	20

dos nos primeiros níveis da hierarquia. Porém, no último nível, o HCGA obteve um resultado estatisticamente inferior comparado com todos os baselines que tiveram resultados, com perda significativa. Temos que estudar se o HCGA se comporta melhor em bases esparsas, devido a sua representação, ou se seu mal desempenho nesta base está relacionado a outras características dessa base. Uma forma de alteração de representação seria trocar o formato dos atributos positivos e negativos das regras, ou seja, fazer com que os atributos positivos sejam exclusivos ("E") e os negativos sejam inclusivos ("OU") para que exemplos de bases mais densas possam ser melhor identificados.

Um resumo dos resultados apresentados para todos os conjuntos de dados podem ser encontrados na Tabela 5.8. Para cada conjunto de dados em cada nível realizamos cinco comparações. Note-se que, em geral (92 comparações), o HCGA tem resultados estatisticamente melhores em 34 casos, estatisticamente iguais em 38 casos e 20 derrotas (13 de 80 se compararmos apenas as bases de bioinformática). Em geral, o HCGA apresenta bons resultados, especialmente nos níveis mais profundos da hierarquia, o que é desejado. Em todos os conjuntos de dados, além de GPCRPfam e 20News, os resultados obtidos são bastante estáveis.

5.5 Avaliação da Evolução do AG

A Figura 5.1 mostra o resultado da evolução de classes específicas do primeiro ao quarto níveis de uma hierarquia. Os gráficos relatam o resultado da *fitness* do melhor indivíduo a cada geração calculada sobre a base de treinamento (BestTrain), validação (BestValidation) e no teste (BestTest). Eles também relatam separadamente os valores de precisão (PrTest) e revocação (RcTest) no teste, para que possamos ter uma ideia melhor de como as regras estão evoluindo. Por uma questão de completude, mostramos também a *fitness* média (avgValidation) de toda a população calculada sobre o conjunto de exemplos de validação. Note que os gráficos relataram o melhor indivíduo, que é

então combinado com alguns outros indivíduos para formar o comitê. Note também que não há *overfitting*, já que a *fitness* sobre os conjuntos de treinamento, validação e teste são muito semelhantes, enquanto a *fitness* média da população se encontra bem abaixo desses valores. É interessante analisar como se comportam a precisão e a revocação, onde vemos a precisão alta variando muito pouco, enquanto a revocação é facilmente alterada por qualquer variação no indivíduo da espécie analisada.

Além do primeiro nível, onde a revocação está próxima de 100% enquanto os valores de precisão estão por volta de 87%, para todos os outros níveis os valores de precisão são sempre superiores aos de revocação. Em alguns casos, por exemplo, na classe 001, as primeiras regras geradas já são bastante boas, e o HCGA melhora apenas a revocação durante a evolução. Em contrapartida, nos três gráficos no segundo nível, correspondentes ao 3^o e 4^o níveis da hierarquia, a precisão já é 100% durante as primeiras 5 a 10 gerações. No entanto, a revocação melhora significativamente.

5.6 Tempo de Execução

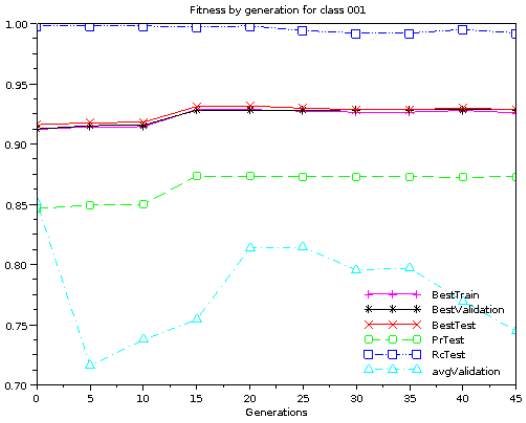
O tempo de execução dos experimentos com HCGA variaram de 4 a 10 minutos em um CPU Intel(R) Core(TM) i7 com 2.67 GHz e 6 GB de RAM. Uma média do tempo das execuções para a execução de todas as classes de uma base para o HCGA e os outros algoritmos apresentados é mostrada na Tabela 5.9. Podemos observar que o custo aumenta proporcionalmente com o número de exemplos. Comparado com o tempo dos outros algoritmos avaliados, o HCGA tem um custo um pouco maior do que os algoritmos mais eficientes, como o HyperPipes e BayesNet. Porém, o algoritmo proposto possui um comportamento estável em relação às diferentes bases, diferente do apresentado pelo J48 e pelo Naive Bayes.

É importante lembrar que o tempo apresentado foi retirado da média do tempo de execução do HCGA limitado a uma única *thread* e com impressão de várias estatísticas apenas para visualização detalhada dos resultados (como geração dos gráficos para evolução). Essas estatísticas obrigam o algoritmo a recalcular três vezes a *fitness* de cada população, uma sobre cada conjunto de exemplos (treino, validação e teste), que é o processo que tem maior custo computacional no HCGA, além do próprio processo evolutivo do algoritmo. Assim, muito pode ser feito para melhorar o tempo de execução do HCGA, como a remoção das estatísticas desnecessárias, evitando recálculos, e utilizando paralelismo (mais de uma *thread*), principalmente em processos sequenciais dentro do HCGA, como o cálculo de *fitness* e evolução de uma população. Como o custo computacional do algoritmo não é um objetivo deste trabalho, deixamos essas

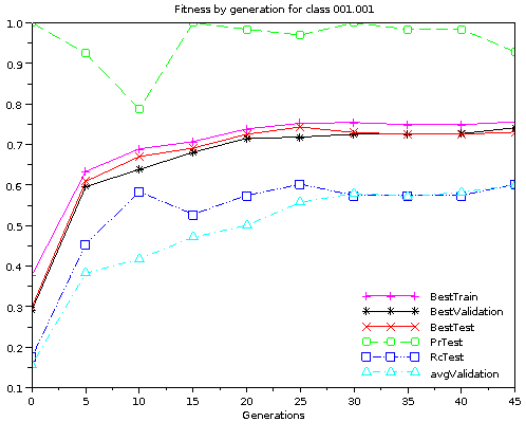
Tabela 5.9. Média de tempo de uma execução dos algoritmos para cada base de dados

Base	tempo médio (minutos)					
	HCGA	OlexGA	BayesNet	HyperPipes	J48	NB
GPCRprints	4.56	4.60	1.46	1.67	2.45	21.55
GPCRpfam	5.60	3.10	1.00	1.40	1.59	4.00
GPCRinterpro	6.94	5.92	2.16	2.20	3.59	36.08
GPCRprosite	4.72	3.97	1.17	1.57	1.67	4.25
20News	9.97	3.97	n/a	3.60	90.82	>1 dia

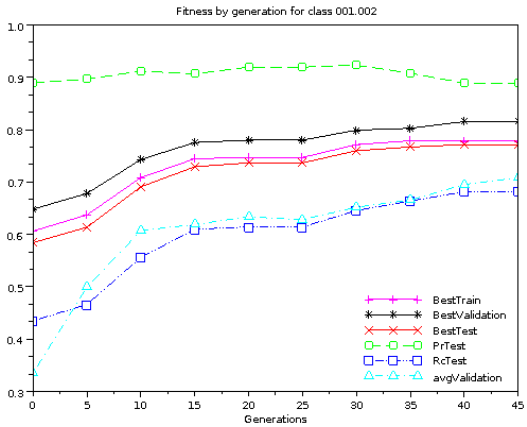
otimizações como trabalhos futuros.



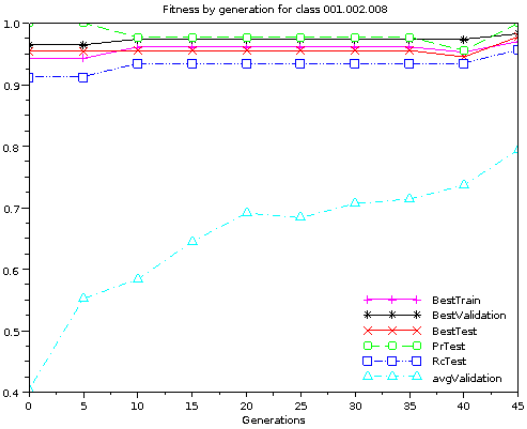
(a) Classe 001 (1^o nível)



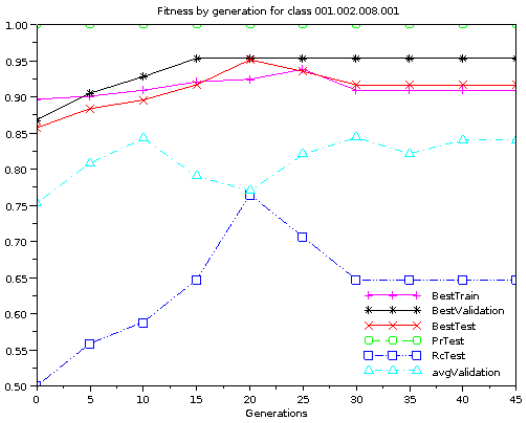
(b) Classe 001.001 (2^o nível)



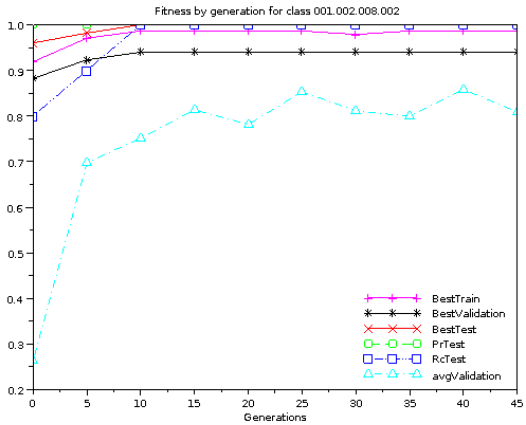
(c) Classe 001.002 (2^o nível)



(d) Classe 001.002.008 (3^o nível)



(e) Classe 001.002.008.001 (4^o nível)



(f) Classe 001.002.008.002 (4^o nível)

(g) Destaque da legenda

Figura 5.1. Gráficos da *fitness* por geração durante a evolução do HCGA em algumas classes da base GPCRInterpro

Capítulo 6

Conclusão e Trabalhos Futuros

A classificação hierárquica apresenta-se hoje como um grande desafio, dado o grande número de classes e a baixa quantidade de exemplos nas classes mais profundas que a maioria dos problemas apresenta. Este trabalho introduziu o HCGA, um algoritmo genético construído para a classificação hierárquica. Em comparação com outros algoritmos na literatura, o HCGA tem a vantagem de lidar com ambas informações locais e globais ao mesmo tempo, enquanto outros métodos podem lidar apenas com uma delas. O método trabalha de forma *top-down* tanto na elaboração dos modelos, quanto ao classificar novos exemplos. Isto difere de outros métodos de abordagem *top-down*, que exploram esta estratégia somente durante a fase de teste.

O HCGA foi aplicado em dados de duas grandes áreas: bioinformática e texto. Quatro conjuntos de dados são relacionados à previsão da atividade GPCR, um assunto relevante para a área de bioinformática, já que cerca de 50% dos medicamentos atuais atuam nas atividades GPCR. Neste caso, os resultados mostraram que o HCGA pode produzir resultados estatisticamente tão bons ou melhores do que cinco baselines executados em conjunto com a tradicional abordagem *top-down*, tendo desempenho muito bom nos últimos níveis da hierarquia.

O HCGA ainda foi avaliado sobre o conjunto de dados "20 Newsgroups", uma base de texto formado por mensagens de um fórum na Internet. A importância desse tipo de classificação cresce tanto quanto a necessidade de se localizar informações específicas na quantidade de informação em constante expansão disponibilizada nessa mídia, que é cada vez e mais necessária no mundo atual. Resultados iniciais sobre esta base mostraram que o HCGA tem resultados promissores em relação aos baselines, mas inferiores se comparados com as bases de bioinformática. Sabemos que ambos domínios possuem características bem diferentes, principalmente em relação a esparsidade dos dados.

Investigar as características responsáveis pela diferença de desempenho nos dois domínios é essencial. Um ponto a ser considerado é entender se uma melhor seleção de atributos nesses domínios pode levar a melhores resultados. Como esse não era o foco deste trabalho, ele aparece como o primeiro item de uma lista de trabalhos futuros.

Além disso, iremos estender a modelagem do HCGA para tratar bases que não sejam binárias. Uma alteração da representação do algoritmo é necessária para que possa tratar um conjunto maior de problemas, evitando a necessidade da discretização da base. Quanto mais valores um atributo pode ter, mais complexa se torna a sua avaliação.

Ainda nesse contexto, novos métodos de seleção de atributos hierárquicos podem ser explorados a fim de reduzir o espaço de busca, incidindo apenas sobre os atributos que são relevantes para todos os níveis hierárquicos.

Uma segunda linha de trabalhos futuros refere-se ao estudo mais extenso de parâmetros. Nossas experiências mostraram que variando a estratégia de seleção do comitê, e também o limiar utilizado para decidir se um exemplo deve ser movido para baixo da árvore durante a fase de teste, mudam significativamente os resultados. A consideração de fitness dinâmicos para diferentes níveis da hierarquia, que dão preferência à precisão ou revocação dependendo das regras vindas do nível superior também parece uma boa opção.

Uma outra linha de pesquisa é como alterar o HCGA para evitar a propagação de erro para os níveis mais profundos da hierarquia, comum em algoritmos de abordagem *top-down*. Esse é um problema de extrema relevância, mas de solução desconhecida.

Por último, o método pode ser facilmente estendido para a tarefa de classificação multi-rótulo. Como mostrado na literatura, grande parte dos problemas hierárquicos encontrados hoje são também multi-rótulo [81]. Com essa extensão, a abordagem ⟨ SPP, NMLNP, T, LCN⟨LI⟩⟩ do HCGA poderá ser alterada para a abordagem ⟨ MPP, NMLNP, D, LCPN⟩. Isto significa, de acordo com a categorização de [81] apresentada na Seção 2.2, que ela passará a seguir uma estrutura em DAG (D) para estruturação das classes, com previsão por múltiplos caminhos por nível de classe (MPP), em vez de uma estruturação em árvore (T), que possui apenas um único caminho para cada previsão (SPP). Ainda, um modelo pode ser treinado por nó pai da hierarquia (LCPN), em vez de por nó (LCN), visando o treinamento simultâneo das espécies filhas do nó pai, que pode utilizar também da co-evolução.

Referências Bibliográficas

[1] GPCRDB. http://www.gpcr.org/, 2007.

[2] UniProt. http://www.expasy.UniProt.org/, 2007.

[3] 20News (20 Newsgroups) dataset. http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html, 2009.

[4] GPCR (G protein-coupled receptor) datasets. http://www.cs.kent.ac.uk/archive/people/rpg/nh56/, 2011.

[5] Thomas Baeck and D.B Fogel. *Evolutionary Computation 1: Basic Algorithms and Operators*. IOP Publishing Ltd, 2000.

[6] Thomas Baeck, D.B Fogel, and Z. Michalewicz. *Evolutionary Computation 2: Advanced Algorithms and Operators*. Taylor and Francis, 2000.

[7] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1st edition, May 1999.

[8] Zafer Barutcuoglu, Robert E. Schapire, and Olga G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22:830–836, 2006.

[9] P.N. Bennett and N. Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 11–18. ACM, 2009.

[10] Hendrik Blockeel. Hierarchical multi-classification. In *Proceedings of the First SIGKDD Workshop on Multi-Relational Data Mining (MRDM-2002)*, pages 21–35, 2002.

[11] Mihaela Elena Breaban and Henri Luchian. Unsupervised feature weighting with multi niche crowding genetic algorithms. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1163–1170. ACM, 2009.

[12] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, pages 78–87. ACM, 2004.

[13] Lijuan Cai and Thomas Hofmann. Exploiting known taxonomies in learning overlapping concepts. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 714–719. Morgan Kaufmann Publishers Inc., 2007.

[14] Rafael V. Carvalho, Gustavo Brunoro, and Gisele L. Pappa. HCGA: A Genetic Algorithm for Hierarchical Classification. In *IEEE Congress on Evolutionary Computation*, 2011.

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

[16] N. Cesa-Bianchi and G. Valentini. Hierarchical cost-sensitive algorithms for genome-wide gene function prediction. *Machine Learning in Systems Biology*, page 25, 2009.

[17] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Hierarchical classification: combining Bayes with SVM. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 177–184. ACM, 2006.

[18] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Incremental algorithms for hierarchical classification. *J. Mach. Learn. Res.*, 7:31–54, 2006.

[19] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal – The International Journal on Very Large Data Bases*, 7(3):163–178, 1998.

[20] Yiming Chen, Zhoujun Li, Xiaohua Hu, and Junwan Liu. Hierarchical classification with dynamic-threshold SVM ensemble for gene function prediction. In *Proceedings of the 6th International Conference on Advanced Data Mining and Applications - Volume Part II*, ADMA'10, pages 336–347. Springer-Verlag, 2010.

[21] Rick Chow, Wei Zhong, Michael Blackmon, Richard Stolz, and Marsha Dowell. An efficient SVM-GA feature selection model for large healthcare databases. In *Proceedings of the 10th annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 1373–1380. ACM, 2008.

[22] A. Clare and R. D. King. Predicting gene function in saccharomyces cerevisiae. *Bioinformatics*, 19($suppl_2$):ii42–49, 2003.

[23] E. Costa, A. Lorena, A. Carvalho, and A. Freitas. Top-down hierarchical ensembles of classifiers for predicting g-protein-coupled-receptor functions. *Advances in Bioinformatics and Computational Biology*, pages 35–46, 2008.

[24] S. D'Alessio, K. Murray, R. Schiaffino, and A. Kershenbaum. The effect of using hierarchical classifiers in text categorization. In *Proceeding of RIAO-00, 6th International Conference Recherche d'Information Assistee par Ordinateur*, pages 302–313. Citeseer, 2000.

[25] Omid David-Tabibi, Nathan S. Netanyahu, Yoav Rosenberg, and Moshe Shimoni. Genetic algorithms for automatic classification of moving objects. In *Proceedings of the 12th annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 2069–2070. ACM, 2010.

[26] I. De Falco, A. Della Cioppa, and E. Tarantino. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269, 2001.

[27] Kenneth A. de Jong, William M. Spears, and Diana F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993.

[28] K Deb. Introduction to selection. *Evolutionary Computation Basic algorithms and Operators*, pages 1–7, 2000.

[29] Kalyanmoy Deb and David E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50. Morgan Kaufmann Publishers Inc., 1989.

[30] Ofer Dekel, Joseph Keshet, and Yoram Singer. Large margin hierarchical classification. In *Proceedings of the 21st International Conference on Machine learning*, ICML '04, pages 27–. ACM, 2004.

[31] Ofer Dekel, Joseph Keshet, and Yoram Singer. An online algorithm for hierarchical phoneme classification. In Samy Bengio and Hervé Bourlard, editors, *Machine Learning for Multimodal Interaction*, volume 3361 of *Lecture Notes in Computer Science*, pages 146–158. Springer Berlin / Heidelberg, 2005.

[32] Susan Dumais and Hao Chen. Hierarchical classification of web content. In *SIGIR '00: Proceedings of the 23rd annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–263, 2000.

[33] R. Eisner, B. Poulin, D. Szafron, P. Lu, and R. Greiner. Improving protein function prediction using the hierarchical structure of the gene ontology. In *Computational Intelligence in Bioinformatics and Computational Biology. Proceedings of the 2005 IEEE Symposium on*, CIBCB '05, pages 1–10, 2005.

[34] A. Esuli, T. Fagni, and F. Sebastiani. Boosting multi-label hierarchical text categorization. *Information Retrieval*, 11(4):287–313, 2008.

[35] Tiziano Fagni and Fabrizio Sebastiani. On the selection of negative examples for hierarchical text categorization. In *Proceedings of the 3rd Language Technology Conference*, LTC'07, pages 24–28. Poznan, 2007.

[36] D. Fillmore. It′s a GPCR world. *Modern Drug Discovery*, 11(7):24–28, 2004.

[37] A. A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2002.

[38] A. A. Freitas. A review of evolutionary algorithms for data mining. In O. Maimon and L. Rokach, editors, *Soft Computing for Knowledge Discovery and Data Mining*, pages 61–93. Springer, 2007.

[39] S. Gauch, A. Chandramouli, and S. Ranganathan. Training a hierarchical classifier using inter document relationships. *Journal of the American Society for Information Science and Technology*, 60(1):47–58, 2009.

[40] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[41] Heather J. Goldsby, Sherri Goings, Jeff Clune, and Charles Ofria. Problem decomposition using indirect reciprocity in evolved populations. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 105–112. ACM, 2009.

[42] Y. Guan, C.L. Myers, D.C. Hess, Z. Barutcuoglu, A.A. Caudy, and O.G. Troyanskaya. Predicting gene function in a hierarchical context with an ensemble of classifiers. *Genome Biology*, 9(Suppl 1):S3, 2008.

[43] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software, 2009.

[44] Hendrik Hendrik Blockeel, Leander Schietgat, Jan Struyf, Saso Dzeroski, and Amanda Clare. Decision trees for hierarchical multilabel classification: A case study in functional genomics. In Johannes Furnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Knowledge Discovery in Databases: PKDD 2006*, volume 4213 of *Lecture Notes in Computer Science*, pages 18–29. Springer Berlin / Heidelberg, 2006.

[45] P. G. Higgs and T. K. Attwood. *Bioinformatics and Molecular Evolution.* Blackwell, 2005.

[46] N. Holden and A.A. Freitas. A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 100–107. IEEE, 2005.

[47] N. Holden and A.A. Freitas. Hierarchical Classification of G-Protein-Coupled Receptors with a PSO/ACO Algorithm. In *Proc. IEEE Swarm Intelligence Symposium (SIS-06)*, pages 77–84, 2006.

[48] N. Holden and A.A. Freitas. Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 13(3):259–272, 2009.

[49] Nicholas Holden and Alex A. Freitas. Improving the performance of hierarchical classification with swarm intelligence. In *Proceedings of the 6th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, EvoBIO '08, pages 48–60, 2008.

[50] B. Jin, B. Muller, C. Zhai, and X. Lu. Multi-label literature classification based on the Gene Ontology graph. *BMC bioinformatics*, 9(1):525, 2008.

[51] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Machine Learning: ECML-98*, pages 137–142. Springer Berlin / Heidelberg, 1998.

[52] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995.

[53] N. Kapalavayi, S. N. J. Murthy, and Gongzhu Hu. Hierarchical approach to select feature vectors for classification of text documents. In *AICCSA '06: Proceedings of the IEEE International Conference on Computer Systems and Applications*, pages 1180–1183, 2006.

[54] Svetlana Kiritchenko. *Hierarchical text categorization and its application to bioinformatics*. PhD thesis, University of Ottawa, Canada, 2006.

[55] Svetlana Kiritchenko, Stan Matwin, Richard Nock, and A. Fazel Famili. Learning and evaluation in the presence of class hierarchies: Application to text categorization. In *Canadian Conference on AI*, pages 395–406, 2006.

[56] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In Douglas H. Fisher, editor, *Proc. of ICML-97, 14th Int. Conf. on Machine Learning*, pages 170–178, 1997.

[57] H.P. Kriegel, P. Kröger, A. Pryakhin, and M. Schubert. Using support vector machines for classifying large sets of multi-represented objects. In *Proc. 4th SIAM Int. Conf. on Data Mining*, pages 102–114. Citeseer, 2004.

[58] Anastasia Krithara, Massih-Reza Amini, Jean-Michel Renders, and Cyril Goutte. Semi-supervised document classification with a mislabeling error model. In *ECIR*, pages 370–381, 2008.

[59] Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. Hierarchical document classification using automatically generated hierarchy. *J. Intell. Inf. Syst.*, 29(2):211–230, 2007.

[60] Marek Lipczak and Evangelos Milios. Agglomerative genetic algorithm for clustering in social networks. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1243–1250. ACM, 2009.

[61] Juliet Juan Liu and James Tin yau Kwok. An extended genetic rule induction algorithm. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 458–463, 2000.

[62] Andrew McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *ICML '98:*

Proceedings of the 15th International Conference on Machine Learning, pages 359–367, 1998.

[63] B. S. P. Mishra, A. K. Addy, R. Roy, and S. Dehuri. Parallel multi-objective genetic algorithms for associative classification rule mining. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*, ICCCS '11, pages 409–414. ACM, 2011.

[64] Dunja Mladenić and Marko Grobelnik. Feature selection on hierarchy of web documents. *Decis. Support Syst.*, 35:45–87, 2003.

[65] Nam Nguyen. Improving hierarchical classification with partial labels. In *Proceeding of the ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 315–320. IOS Press, 2010.

[66] Fernando E. Otero, Alex A. Freitas, and Colin G. Johnson. A hierarchical classification ant colony algorithm for predicting gene ontology terms. In *Proceedings of the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, EvoBIO '09, pages 68–79, 2009.

[67] Adriana Pietramala, Veronica L. Policicchio, Pasquale Rullo, and Inderbir Sidhu. A genetic algorithm for text classification rule induction. In *ECML PKDD '08: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Part II*, pages 188–203, 2008.

[68] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[69] Kunal Punera and Joydeep Ghosh. Enhanced hierarchical classification via isotonic smoothing. In *Proceeding of the 17th International Conference on World Wide Web*, WWW '08, pages 151–160. ACM, 2008.

[70] Kunal Punera, Suju Rajan, and Joydeep Ghosh. Automatically learning document taxonomies for hierarchical classification. In *WWW '05: Special interest tracks and posters of the 14th International Conference on World Wide Web*, pages 1010–1011. International World Wide Web Conference, 2005.

[71] Xipeng Qiu, Wenjun Gao, and Xuanjing Huang. Hierarchical multi-class text categorization with global margin maximization. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, ACLShort '09, pages 165–168. Association for Computational Linguistics, 2009.

[72] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[73] C. Romero, S. Ventura, and P. De-Bra. Knowledge discovery with genetic programming for providing feedback to courseware authors. *User Modeling and User-Adapted Interaction*, 14(5):425–464, 2005.

[74] Juho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. Learning hierarchical multi-category text classification models. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pages 744–751. ACM, 2005.

[75] Juho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. Kernel-based learning of hierarchical multilabel classification models. *J. Mach. Learn. Res.*, 7:1601–1626, December 2006.

[76] M.E. Ruiz and P. Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118, 2002.

[77] M. Sasaki and K. Kita. Rule-based text categorization using hierarchical categories. In *Proc. of IEEE Int. Conf. on Systems, Man, and Cybernetics*, pages 2827–2830, 1998.

[78] A. Secker, M. N. Davies, A. A. Freitas, E. B. Clark, J. Timmis, and D. R. Flower. Hierarchical classification of G-Protein-Coupled Receptors with data-driven selection of attributes and classifiers. *Int. J. Data Min. Bioinformatics*, 4:191–210, March 2010.

[79] Andrew Secker, Matthew N. Davies, Alex A. Freitas, Jon Timmis, Miguel Mendao, and Darren R. Flower. An experimental comparison of classification algorithms for the hierarchical prediction of protein function. *Magazine of the British Computer Society's Specialist Group on AI*, 9:17–22, 2007.

[80] Matthias W. Seeger. Cross-validation optimization for large scale structured classification kernel methods. *J. Mach. Learn. Res.*, 9:1147–1178, June 2008.

[81] Carlos Silla and Alex Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22:31–72, 2011.

[82] Carlos N. Silla and Alex A. Freitas. Novel top-down approaches for hierarchical classification and their application to automatic music genre classification. In

Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics, SMC '09, pages 3499–3504. IEEE Press, 2009.

[83] Stephen F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the Eighth International joint Conference on Artificial Intelligence - Volume 1*, pages 422–425. Morgan Kaufmann Publishers Inc., 1983.

[84] Wei Song, Shi Tong Wang, and Cheng Hua Li. Parametric and nonparametric evolutionary computing with a content-based feature selection approach for parallel categorization. *Expert Syst. Appl.*, 36(9):11934–11943, 2009.

[85] A. Sun, E. Lim, and W. Ng. Performance measurement framework for hierarchical text classification. *Journal of the American Society of Information Science and Technology*, 54(11):1014–1028, 2003.

[86] Aixin Sun. Blocking reduction strategies in hierarchical text classification. *IEEE Trans. on Knowl. and Data Eng.*, 16(10):1305–1308, 2004.

[87] Aixin Sun and Ee-Peng Lim. Hierarchical text classification and evaluation. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 521–528, 2001.

[88] D. Tikk, G. Biro, and A. Törcsvári. A hierarchical online classifier for patent categorization. *Emerging Technologies of Text Mining: Techniques and Applications. Idea Group Inc*, 2007.

[89] Domonkos Tikk and Gyorgy Biró. Experiment with a hierarchical text categorization method on the wipo-alpha patent collection. In *Proceedings of the 4th International Symposium on Uncertainty Modelling and Analysis*, ISUMA '03, pages 104–. IEEE Computer Society, 2003.

[90] Domonkos Tikk, Jae Dong Yang, and Sun Lee Bang. Hierarchical text categorization using fuzzy relational thesaurus. *Kybernetika*, 39(5):583–600, 2003.

[91] A. Tsakonas, G. Dounias, J. Jantzen, H. Axer, B. Bjerregaard, and D. G. von Keyserlingk. Evolving rule-based systems in two medical domains using genetic programming. *Artificial Intelligence in Medicine*, 32(3):195–216, 2004.

[92] Giorgio Valentini. True path rule hierarchical ensembles. In *Proceedings of the 8th International Workshop on Multiple Classifier Systems*, MCS '09, pages 232–241. Springer-Verlag, 2009.

[93] Rosane Maria Maffei Vallim, Thyago S.P.C. Duque, David E. Goldberg, and André C.P.L.F. Carvalho. The multi-label ocs with a genetic algorithm for rule discovery: implementation and first results. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1323–1330. ACM, 2009.

[94] Celine Vens, Jan Struyf, Leander Schietgat, Saso Dzeroski, and Hendrik Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

[95] Junhui Wang, Xiaotong Shen, and Wei Pan. On large margin hierarchical classification with multiple paths. *J. Amer. Statist. Assoc.*, 104(487):1213–1223, 2009.

[96] K. Wang, S. Zhou, and Y. He. Hierarchical classification of real life documents. In *Proc. of the 1st SIAM Int. Conf. on Data Mining*, 2001.

[97] Yi Wang and Zhiguo Gong. Hierarchical classification of web pages using support vector machine. In *Proceedings of the 11th International Conference on Asian Digital Libraries: Universal and Ubiquitous Access to Information*, ICADL 08, pages 12–21. Springer-Verlag, 2008.

[98] A.S. Weigend, E.D. Wiener, and J.O. Pedersen. Exploiting hierarchy in text categorization. *Information Retrieval*, 1(3):193–216, 1999.

[99] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2nd edition, 2005.

[100] M. L. Wong and K. S. Leung. *Data Mining Using Grammar-Based Genetic Programming and Applications*. Springer, 2000.

[101] Feihong Wu, Jun Zhang, and Vasant Honavar. Learning classifiers using hierarchically structured class taxonomies. In Jean-Daniel Zucker and Lorenza Saitta, editors, *Abstraction, Reformulation and Approximation*, volume 3607 of *Lecture Notes in Computer Science*, pages 902–902. Springer Berlin / Heidelberg, 2005.

[102] G.R. Xue, D. Xing, Q. Yang, and Y. Yu. Deep classification in large-scale text hierarchies. In *Proceedings of the 31st annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 619–626. ACM, 2008.

[103] Arthur Zimek, Fabian Buchwald, Eibe Frank, and Stefan Kramer. A study of hierarchical and flat classification of proteins. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 7:563–571, 2010.