UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DOUGLAS BALDON CORREA

ABORDAGEM PARA REDUÇÃO DA HIERARQUIA DE CLASSES EM PROBLEMAS DE CLASSIFICAÇÃO HIERÁRQUICA

PONTA GROSSA

2024

DOUGLAS BALDON CORREA

ABORDAGEM PARA REDUÇÃO DA HIERARQUIA DE CLASSES EM PROBLEMAS DE CLASSIFICAÇÃO HIERÁRQUICA

Approach to class hierarchy reduction in hierarchical classification problems

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientadora: Profª. Drª. Helyane Bronoski Borges

PONTA GROSSA

2024

DOUGLAS BALDON CORREA

ABORDAGEM PARA REDUÇÃO DA HIERARQUIA DE CLASSES EM PROBLEMAS DE CLASSIFICAÇÃO HIERÁRQUICA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 04/06/2024

Helyane Bronoski Borges
Doutora
Universidade Tecnológica Federal do Paraná

Luiz Rafael Schmitke
Doutor
Universidade Tecnológica Federal do Paraná

Simone Nasser Matos
Doutora
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2024

AGRADECIMENTOS

Certamente estes parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre essas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

Agradeço primeiramente à minha orientadora Profª. Drª. Helyane Bronoski Borges, pela orientação dedicada, paciência e incentivo contínuo ao longo desta jornada acadêmica. Sua sabedoria e experiência foram fundamentais para a realização deste trabalho.

A todos os professores que, com paciência e dedicação, tiraram minhas dúvidas e contribuíram significativamente para o desenvolvimento deste trabalho.

Aos meus colegas de sala, pelos momentos compartilhados, pelas discussões produtivas e pelo apoio mútuo nos desafios enfrentados ao longo do curso.

À Secretaria do Curso, pelo suporte administrativo e pela prontidão em atender às minhas necessidades acadêmicas e burocráticas.

Gostaria de expressar minha profunda gratidão à minha família. Sem o suporte emocional e financeiro de vocês, este trabalho não teria sido possível. A compreensão e o carinho que recebi foram essenciais para superar os obstáculos.

Finalmente, agradeço a todos que, direta ou indiretamente, contribuíram para a realização desta pesquisa, seja através de apoio moral, intelectual ou técnico.

Enfim, a todos que de alguma forma fizeram parte desta conquista, meu sincero agradecimento.

RESUMO

Em problemas de classificação hierárquica multirrótulo, a complexidade surge quando uma instância pode ser categorizada em múltiplas classes organizadas em uma estrutura hierárquica do tipo DAG (*Directed Acyclic Graph*). Tais problemas são comumente enfrentados em domínios como bioinformática, reconhecimento de imagens e classificação de texto, onde a granularidade e o desbalanceamento de classes podem levar a desafios computacionais e de precisão de predição. Este trabalho desenvolveu um método de pré-processamento que reduz a hierarquia de classes, visando simplificar as redundâncias e a complexidade da hierarquia. A abordagem desenvolvida foi avaliada por meio de testes em dez bases de dados de classificação hierárquica multirrótulo. Após a redução das classes, as bases de dados foram testadas por meio do classificador hierárquico Clus-HMC e comparadas com os resultados da literatura. Por meio do teste estatístico de Wilcoxon, observou-se que a redução da hierarquia de classes resultou em melhorias estatisticamente significativas nos valores de AUPRC (Área Sob a Curva de Precisão-Recall). Isso indica que o método de pré-processamento proposto foi eficaz em simplificar as estruturas hierárquicas e, consequentemente, aumentar a precisão e a eficiência dos modelos de classificação hierárquica multirrótulo.

Palavras-chave: aprendizado de máquina; classificação hierárquica multirrótulo; redução da hierarquia de classes; grafos ací clicos dirigidos.

ABSTRACT

This work presents an approach to simplify the complexity in hierarchical multi-label classification problems, where an instance can be categorized into multiple classes organized in a Directed Acyclic Graph (DAG) structure. These problems are commonly encountered in domains such as bioinformatics, image recognition, and text classification, where class granularity and imbalance can lead to computational and prediction accuracy challenges. The project developed a pre-processing method to reduce class hierarchy, aiming to simplify redundancies and complexity. The proposed approach was evaluated through tests on ten hierarchical multi-label classification datasets. After reducing the classes, the datasets were tested using the Clus-HMC hierarchical classifier and compared with results from the literature. Using the Wilcoxon statistical test, it was observed that the reduction in class hierarchy resulted in statistically significant improvements in AUPRC (Area Under the Precision-Recall Curve) values. This indicates that the proposed preprocessing method effectively simplified hierarchical structures, consequently increasing the accuracy and efficiency of hierarchical multi-label classification models.

Keywords: machine learning; hierarchical multilabel classification; hierarchy reduction; directed acyclic graphs.

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE TABELAS

LISTAGEM DE CÓDIGOS FONTE

SUMÁRIO

1 INTRODUÇÃO

O aprendizado de máquina (AM) é um campo da inteligência artificial que permite que os sistemas aprendam e façam previsões ou decisões baseadas em dados. Técnicas de AM têm sido aplicadas em diversos domínios, como diagnóstico médico, reconhecimento de padrões e análise preditiva (ZHOU, 2019).

A classificação é uma tarefa central em aprendizado de máquina, onde o objetivo é categorizar dados em diferentes classes ou rótulos. Métodos de classificação incluem algoritmos como Naive Bayes, árvores de decisão, máquinas de vetores de suporte (SVM) e redes neurais, cada um com suas próprias vantagens e desvantagens dependendo da natureza do problema e dos dados disponíveis (ZHOU, 2019).

A classificação hierárquica é uma extensão da classificação tradicional que organiza as classes em uma estrutura hierárquica, permitindo a categorização de dados em múltiplos níveis. Na classificação hierárquica, as classes são organizadas em uma estrutura de árvore ou de grafo acíclico direcionado (DAG), onde cada nó representa uma classe e as arestas representam relações de ancestralidade e descendência entre as classes (SILLA; FREITAS, 2011).

A classificação hierárquica pode ainda ser multirrótulo. Segundo Cerri, Barros e Carvalho (2013), a classificação hierárquica multirrótulo é uma técnica de aprendizado de máquina que permite classificar uma instância em uma ou mais categorias, onde essas categorias são organizadas em uma estrutura hierárquica. Cada nó da hierarquia pode ter um ou mais rótulos associados a ele, ou seja, uma instância pode ser classificada em mais de uma categoria em um único processo preditivo. Esse tipo de classificação é útil em muitos cenários onde as categorias têm uma relação de subordinação entre si.

Existem desafios quanto à classificação hierárquica multirrótulo, como a alta granularidade, desbalanceamento de classes, ambiguidade de categorias, classes interdependentes e pré-processamentos complexos. Devido a isso, um problema abordado na literatura é a redução das classes da hierarquia.

De acordo com Cerri *et al.* (2016), o problema de redução da hierarquia em classificação hierárquica multirrótulo é um cenário em que se busca diminuir o número de classes presentes em diferentes níveis da hierarquia, a fim de simplificar a tarefa de classificação e melhorar o desempenho do modelo. Esse problema surge devido ao grande número de classes presentes em algumas hierarquias, o que pode tornar a tarefa de classificação inviável em termos computacionais e de predição quanto aos classificadores. Além disso, pode haver redundância de informações entre as classes, o que pode levar a resultados não conclusivos.

A classificação hierárquica multirrótulo apresenta desafios devido à complexidade de seus dados. Em casos onde uma instância é associada a múltiplos rótulos hierárquicos, surgem problemas de processamento, como alta granularidade, desbalanceamento de classes, interdependência entre categorias e a necessidade de pré-processamentos complexos. A estrutura do

tipo DAG pode ampliar esses problemas, aumentando o custo computacional e a redundância entre categorias.

Explorar abordagens de redução da hierarquia de classes que simplifiquem as estruturas sem perder a precisão e a integridade das relações faz-se necessário. Segundo Cerri *et al.* (2016), a literatura apresenta lacunas significativas na eficácia e eficiência dos métodos existentes, indicando a necessidade de desenvolvimento de novas técnicas.

Este projeto desenvolveu uma abordagem de pré-processamento que permite a redução da hierarquia de classes em problemas de classificação hierárquica multirrótulo para estruturas do tipo DAG. O trabalho é baseado na ideia de que a redução da hierarquia pode levar a melhorias na performance do modelo de classificação. A abordagem será avaliada em dez conjuntos de dados, comparando os resultados com a utilização do pré-processamento e sem a utilização do mesmo. O desempenho preditivo é avaliado por meio do Clus-HCM, que é um classificador amplamente utilizado na literatura.

1.1 Objetivos

Esta Seção apresenta os objetivos que orientam este trabalho.

1.1.1 Objetivo geral

Este trabalho visa desenvolver uma abordagem para a redução da hierarquia de classes em problemas de classificação hierárquica multirrótulo para estruturas do tipo DAG.

1.1.2 Objetivos específicos

- Implementar um algoritmo de redução da hierarquia de classes para o contexto de classificação hierárquica multirrótulo;
- Realizar experimentos utilizando a abordagem desenvolvida nos conjuntos de dados;
- Comparar os resultados obtidos com os da literatura.

1.2 Organização do Trabalho

Este trabalho está estruturado em cinco capítulos. O Capitulo 2 explora as teorias que fundamentam o tema deste trabalho de conclusão de curso, fornecendo uma base de conhecimento para a compreensão dos conceitos-chave e das metodologias utilizadas. O Capítulo 3 descreve a implementação de um algoritmo para simplificar a hierarquia de classes em classificação hierárquica multirrótulo. A metodologia envolve testes comparativos em dez conjuntos de

dados e a aplicação de um classificador padrão para avaliar a eficiência do pré-processamento proposto. O Capítulo 4 apresenta e analisa os dados coletados, discutindo suas implicações e como contribuem para responder às questões de pesquisa. Por fim, o Capítulo 5 sumariza as descobertas principais, revisita os objetivos e discute a extensão em que foram alcançados. Além disso, sugere direções futuras para pesquisa e discute as limitações do estudo.

2 REFERENCIAL TEÓRICO

Este Capítulo apresenta conceitos importantes abordados neste trabalho. A Seção 2.1 retrata os principais conceitos sobre classificação. A Seção 2.2 aborda métodos de redução de estruturas hierárquicas presentes na literatura. Por fim, a Seção 2.3 apresenta as considerações finais do capitulo.

2.1 Classificação

A classificação de dados envolve a criação de um modelo que descreve diferentes categorias em um conjunto de dados, essencialmente categorizando as informações. Por exemplo, em contextos bancários, clientes com cartões de crédito podem ser classificados como “risco baixo”, “risco normal” ou “risco alto”. Esse processo categoriza os dados atribuindo rótulos que representam características comuns dentro da mesma categoria. É uma técnica aplicada em áreas como diagnóstico médico, reconhecimento de fala e filtragem de spam, onde o objetivo é prever a categoria de novas observações com base em dados de treinamento (CERRI, 2010).

A classificação é um tipo de aprendizado supervisionado, no qual algoritmos são desenvolvidos para criar classificadores a partir de exemplos rotulados. O processo envolve a criação de um classificador que pode categorizar corretamente novos exemplos usando dados que contêm as saídas esperadas. O classificador é gerado por um algoritmo de indução, que tem como objetivo garantir que o classificador possa classificar corretamente novos dados. Inicialmente, os dados devem ser organizados e preparados, onde cada exemplo é representado por uma tupla de atributos. Os atributos de entrada descrevem as características dos exemplos e são usados para treinar o classificador, enquanto o atributo de saída representa as categorias associadas (CERRI, 2010).

Durante o treinamento, os dados são utilizados por um algoritmo de indução, onde cada exemplo é representado por um par (T_i, y_i), com T_i sendo uma tupla de atributos de entrada e y_i o atributo de saída que classifica o exemplo. O objetivo é aprender uma função que mapeie cada conjunto de atributos T_i para uma das classes predefinidas y_i, ajustando os parâmetros do algoritmo de indução (CERRI, 2010).

O modelo resultante deve ser capaz de prever corretamente a classe de novos exemplos. Contudo, o modelo pode sofrer de ajustado de forma errada. O subajuste ocorre quando o modelo não se ajusta suficientemente aos dados de treinamento, resultando em uma classificação imprecisa de novos exemplos. O sobreajuste, por outro lado, acontece quando o modelo se ajusta excessivamente aos dados de treinamento, tornando-se ineficaz para prever novas observações (CERRI, 2010).

Após a criação do classificador, ele deve ser avaliado com novos exemplos não utilizados durante o treinamento, em uma etapa conhecida como fase de teste ou validação. É essencial considerar a habilidade do classificador em prever corretamente as classes dos novos exemplos,

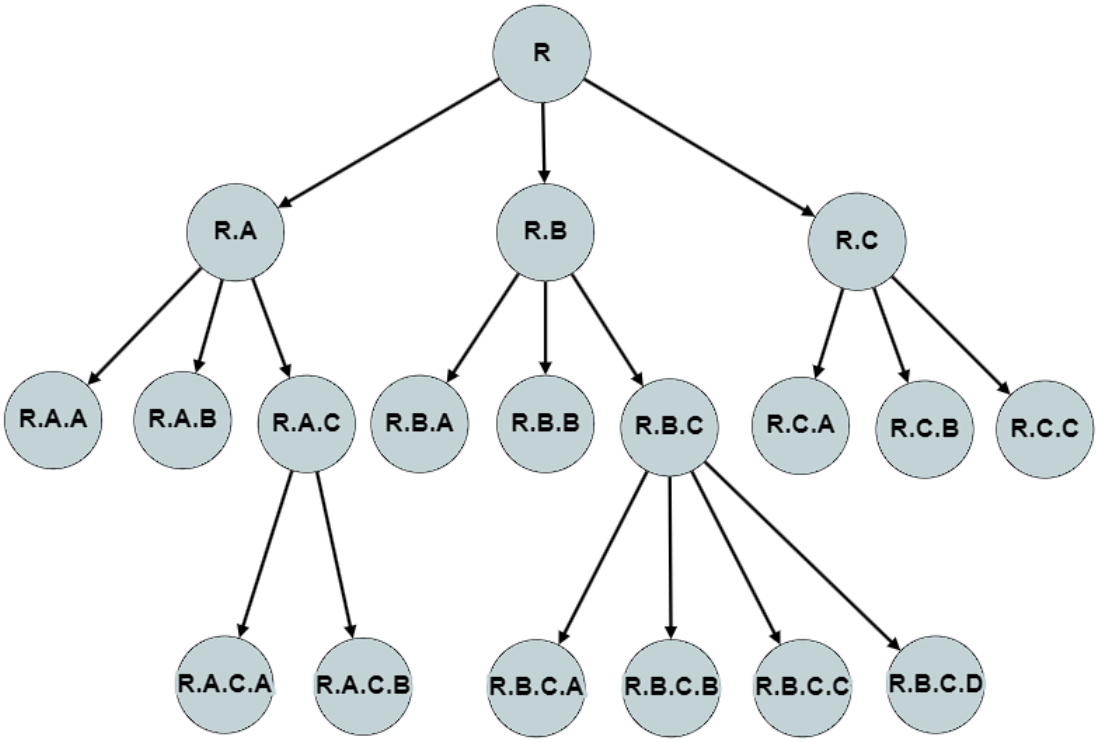
o custo computacional do algoritmo e sua escalabilidade. A qualidade do classificador é medida comparando as classes previstas com as classes verdadeiras dos exemplos (CERRI, 2010).

Segundo Borges e Nievola (2012), há uma variedade de tipos de classificação na literatura, cada uma com suas finalidades específicas, dependendo do problema e dos dados disponíveis. Entre os diferentes tipos de classificação, destacam-se a classificação hierárquica e a classificação hierárquica multirrótulo (CHM), sendo esta última a que será utilizada na experimentação da abordagem. Estas classificações serão abordadas nas subseções seguintes.

2.1.1 Classificação Hierárquica

A classificação hierárquica é uma técnica de aprendizado de máquina que permite a categorização de dados em diferentes níveis. Nesta classificação, as classes são organizadas em uma estrutura hierárquica. A Figura 1 mostra um exemplo de uma árvore organizada nesta estrutura em que os nós da árvore são as classes os ramos são relações de ancestralidade de descendência em um problema de classificação hierárquica.

Figura 1 – Exemplo de uma árvore em estrutura hierárquica



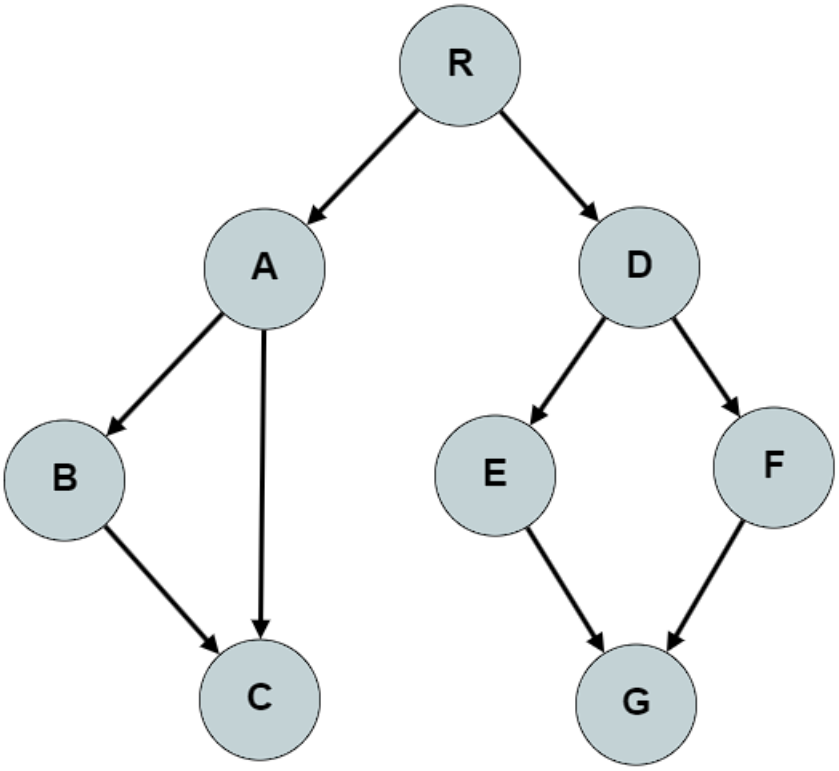
Fonte: Adaptado de Cerri, Barros e Carvalho (2013).

De acordo com Silla e Freitas (2011), as classes em uma hierarquia podem estar organizada em dois tipos de estrutura: árvore e grafos acíclicos direcionados (DAG - *Direct Acyclic Graph*). De acordo com Borges (2012) a diferença entre a estrutura de árvore e a estrutura de

DAG é que na estrutura de árvore cada nó de classe, exceto o nó raiz, tem apenas um pai, enquanto que na estrutura de DAG cada nó de classe pode ter um ou mais nós pai.

- Árvores hierárquicas: são estruturas de dados compostas por nós e arestas que formam uma hierarquia. Cada nó representa uma classe ou categoria, e cada aresta denota a relação de subordinação ou pertinência entre as classes. Este arranjo garante que cada nó, exceto a raiz, tenha exatamente um nó pai.
- DAG: são estruturas de dados similares às árvores hierárquicas, porém, diferenciam-se por permitir que um nó tenha mais de um pai como demonstrado na Figura 2.

Figura 2 – Exemplo de uma árvore em estrutura DAG



Fonte: Adptado de Cerri *et al.* (2016).

Uma das principais vantagens ao lidar com problemas relacionados a DAGs está na capacidade de abordar problemas em estruturas de árvore devido à sua complexidade intrínseca. Além disso, as DAGs podem capturar com mais precisão cenários em que um rótulo é influenciado por vários rótulos de níveis superiores, permitindo uma modelagem exata desses relacionamentos. No entanto, a criação de DAGs tende a ser mais difícil em comparação com abordagens convencionais baseadas em árvores. Isso pode resultar em modelos que são menos intuitivos e mais difíceis de formular e interpretar (BORGES; JR; NIEVOLA, 2013).

A classificação hierárquica é aplicada em uma ampla gama de problemas, incluindo a classificação de texto, reconhecimento de imagens e bioinformática (VIEIRA; BORGES, 2024).

No contexto da classificação de texto, essa abordagem é utilizada para atribuir classes hierarquicamente organizadas a documentos. Por exemplo, na classificação de artigos científicos, as classes “Biologia” e “Genética” podem ser atribuídas, levando em consideração sua relação hierárquica (SADAT; CARAGEA, 2022).

Quando se trata do reconhecimento de imagens, atribuir classes como “Cachorros” e “Mamíferos” a imagens de animais exemplifica essa aplicação, uma vez que reflete as interconexões entre categorias em diferentes níveis da hierarquia. Essa mesma abordagem é estendida à bioinformática, onde a classificação hierárquica é adotada na identificação de sequências genéticas e na atribuição de classes, como “Proteínas” e “Enzimas”, representando as funções moleculares em diferentes níveis de detalhe (CERRI *et al.*, 2016).

O trabalho de Boutell *et al.* (2004) aborda a aplicação de classificação hierárquica em um contexto de classificação de cenas. Nesse cenário, a estrutura hierárquica das classes é explorada para melhorar a eficácia da classificação de imagens complexas categorizando em classes semânticas como praias, pores do sol ou festas. A pesquisa retrata como a organização das categorias em uma hierarquia pode otimizar a precisão e a eficiência da classificação, permitindo que as relações entre as etiquetas sejam consideradas durante o processo.

2.1.2 Classificação hierárquica multirrótulo

Conforme Vens *et al.* (2008), a classificação hierárquica multirrótulo consiste em atribuir várias classes a uma instância e essas classes estão organizadas em uma estrutura hierárquica. Isso significa que as classes estão agrupadas em níveis hierárquicos.

A proposta da classificação hierárquica multirrótulo é prever as múltiplas classes para cada instância de dados, considerando a organização hierárquica dos dados. Isso implica que as decisões sobre a atribuição das classes podem ser influenciadas pelas escolhas feitas em níveis superiores da hierarquia.

Existem diversos algoritmos utilizados na classificação hierárquica multirrótulo. Entre eles, pode-se destacar o Hierarchical k-Nearest Neighbors (HKNN) proposto por Zhang e Zhou (2006), que estende o algoritmo k-Nearest Neighbors (k-NN) para classificação hierárquica multirrótulo. Outro exemplo é o Classifier Chains (CC) apresentado por Read, Pfahringer e Holmes (2008), que transforma o problema multirrótulo em um conjunto de problemas de classificação binária.

O classificador Clus-HMC proposto por Vens *et al.* (2008) é um dos mais utilizados na literatura. Consiste em um classificador hierárquico multirrótulo que utiliza a abordagem de árvores de decisão para prever múltiplas classes organizadas hierarquicamente. No treinamento o Clus-HMC cria uma única árvore de decisão global que é usada para prever todas as classes simultaneamente, onde cada nó representa uma decisão baseada em um atributo dos dados, e as folhas da árvore representam as classes finais. A função de avaliação utilizada para a divisão dos nós é adaptada para considerar a hierarquia das classes, garantindo que as divisões respei-

tem a estrutura hierárquica. Na fase de predição, a árvore de decisão é utilizada para classificar novas instâncias de dados. O processo de predição envolve percorrer a árvore desde a raiz até um nó folha, fazendo escolhas baseadas nos atributos da instância. As classes associadas ao nó folha atingido são então atribuídas à instância, respeitando a hierarquia das classes.

Existem outros classificadores baseados em redes neurais competitivas como Multi-label Hierarchical Classification using a Competitive Neural Network (MHC-CNN) proposto por Borges e Nievola (2012) e Multi-label Hierarchical Classification usinga Strategical Evolutionary (MHC-ES) (BORGES, 2012).

2.1.3 Métricas de Avaliação de classificadores hierárquicos multirrótulo

As métricas de avaliação como *Precision*, *Recall* e *Area Under the Precision-Recall Curve* (AUPRC) são essenciais para medir a eficácia de classificadores hierárquicos multirrótulo. Como apresenta o trabalho de Bishop e Nasrabadi (2006), *Precision* refere-se à proporção de verdadeiros positivos entre as previsões positivas, enquanto *Recall* é a proporção de verdadeiros positivos entre os exemplos que realmente são positivos. A combinação dessas métricas oferece uma visão balanceada do desempenho do modelo, especialmente em problemas com classes desbalanceadas. A métrica AUPRC é particularmente útil para avaliar modelos em cenários de classificação multirrótulo, pois considera a variação de *Precision* e *Recall* em diferentes limiares de decisão, proporcionando uma avaliação mais robusta do desempenho do classificador (VENS *et al.*, 2008). Essas métricas são amplamente utilizadas para avaliar a eficácia de algoritmos de classificação hierárquica multirrótulo, como o Clus-HMC. Tais medidas oferecem insights valiosos sobre a precisão e a capacidade de recuperação do modelo em diferentes níveis hierárquicos, conforme discutido na literatura sobre classificação hierárquica multirrótulo (VENS *et al.*, 2008).

As fórmulas matemáticas dessas métricas são necessárias para a compreensão e aplicação adequada das mesmas na avaliação de classificadores hierárquicos multirrótulo. Conforme discutido por Borges (2012), *Precision* (P) e *Recall* (R) são definidas pelas Equações 1 e 2.

$$P = \frac{\sum_i VP_i}{\sum_i VP_i + \sum_i FP_i} \tag{1}$$

$$R = \frac{\sum_i VP_i}{\sum_i VP_i + \sum_i FN_i} \tag{2}$$

Onde VP representa os verdadeiros positivos, FP os falsos positivos, FN os falsos negativos e i representa as classes.

A média da área sob a curva é representada pela AUPRC. Esta métrica é calculada como a média ponderada das áreas sob as curvas PR individuais. A Equação 3 ilustra como essa abordagem é determinada (BORGES, 2012).

$$\text{AUPRC}_{w_1,\ldots,w_{|C|}} = \sum_i w_i \cdot AUPRC_i \tag{3}$$

Essa abordagem é aplicada de duas maneiras distintas. Na primeira, denominada $AUPRC$, os pesos w_i são definidos como $\frac{1}{|C|}$, onde C representa o conjunto total de classes. Na segunda abordagem, chamada $AUPRC_w$, um peso é atribuído a cada classe com base na sua frequência. Esta frequência é calculada pela fórmula 4 onde v_i é a frequência da classe c_i no conjunto de dados (BORGES, 2012).

$$w_i = \frac{v_i}{\sum_j v_j} \tag{4}$$

As equações do AUPRC representam a área sob a curva *Precision-Recall*, proporcionando uma única métrica que resume a variação de *Precision* e *Recall* em todos os limiares possíveis, oferecendo uma avaliação abrangente do desempenho do classificador, especialmente útil em problemas com classes desbalanceadas (VENS *et al.*, 2008).

Essas fórmulas permitem quantificar de maneira precisa a eficácia dos classificadores hierárquicos multirrótulo, proporcionando uma base sólida para comparações entre diferentes modelos e abordagens.

2.2 Métodos de Redução de DAG presentes na Literatura

A necessidade de processar grandes volumes de dados estruturados em DAGs tem motivado o desenvolvimento de métodos de redução. Esses métodos visam simplificar a complexidade dos grafos sem perder a integridade das relações representadas, facilitando o processamento de consultas de acessibilidade. Um método apresentado por Zhou *et al.* (2017) no qual os autores propuseram uma abordagem composta por duas etapas principais: a Redução Transitiva (TR) e a Redução de Equivalência (ER).

Métodos de redução de DAGs visam examinar e preservar relações essenciais enquanto simplificam a estrutura do grafo, facilitando o processamento e análise. As técnicas incluem a Redução Transitiva, a Redução de Equivalência, a Compactação de Subgrafos e a Contração de Arestas. Cada Seção deste Capítulo descreve um método, incluindo o processo de implementação e referências que fundamentam e validam cada abordagem.

2.2.1 Redução Transitiva

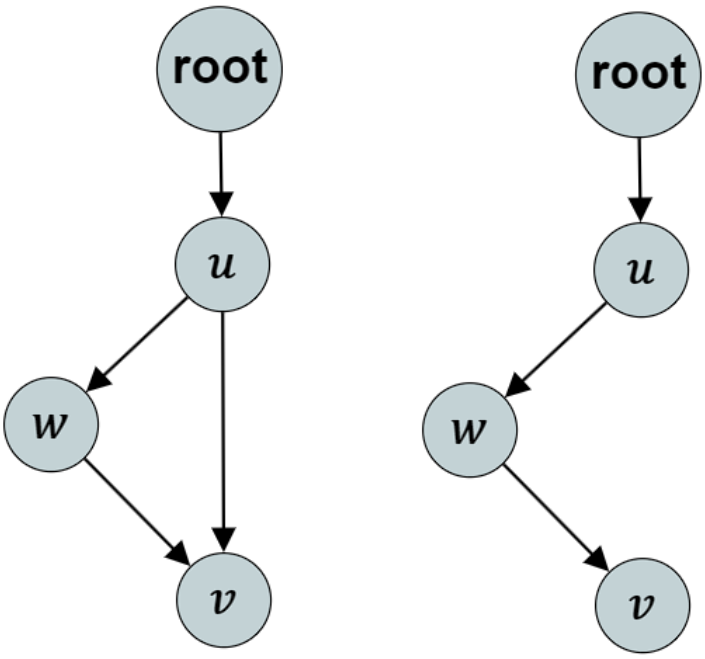
O estudo e aplicação de redução transitiva tem como principal referência o trabalho de Aho, Garey e Ullman (1972), que discute a redução transitiva em termos de algoritmos e suas complexidades. Este trabalho define formalmente a redução transitiva e apresenta algoritmos para sua realização, estabelecendo uma base para muitas das técnicas de processamento de grafos.

A Redução Transitiva (TR) é um conceito utilizado no processamento de Grafos Acíclicos Direcionados que visa simplificar a estrutura do grafo. Esta técnica elimina arestas redundantes, mantendo o fechamento transitivo do grafo e garantindo que a acessibilidade entre os vértices seja preservada.

Considere $G = (V, E)$ um DAG, onde V representa o conjunto de vértices e E o conjunto de arestas direcionadas. A redução transitiva de G, denotada por $G' = (V, E')$, consiste em reter apenas as arestas essenciais em E que são necessárias para manter o fechamento transitivo de G. Uma aresta (u, v) pertence a E' se e somente se não existir um caminho alternativo de u para v que passe por outros vértices intermediários.

O processo envolve verificar para cada aresta (u, v) se existe um vértice intermediário w, tal que existam caminhos direcionados de u para w e de w para v, como demonstra a Figura 3 (a). Se tal caminho existe, a aresta (u, v) é considerada redundante e removida. Este processo é iterativo até que não restem arestas redundantes.

Figura 3 – Exemplo de redução transitiva
(a) Ciclo transitivo **(b) Ciclo transitivo removido**



Fonte: Adaptado de Aho, Garey e Ullman (1972).

O resultado é um novo DAG $G' = (V, E')$, onde E' é um subconjunto das arestas originais E que contém apenas as arestas necessárias para manter o mesmo fechamento transitivo do DAG original. Assim, G' não só mantém todas as relações de acessibilidade presentes em G como também apresenta uma estrutura mais simples e potencialmente mais eficiente para análise e armazenamento, como demonstra a Figura 3 (b).

2.2.2 Redução de Equivalência

Uma referência na literatura para a Redução de Equivalência (ER) em Grafos Acíclicos Direcionados (DAGs) pode ser encontrada no trabalho de Fan *et al.* (2010), que explora a aplicação de ER para otimização de consultas em grafos. O trabalho introduz conceitos e algoritmos detalhados para realizar a redução de equivalência, proporcionando um embasamento sólido sobre como a ER pode ser empregada para simplificar grafos mantendo suas propriedades essenciais.

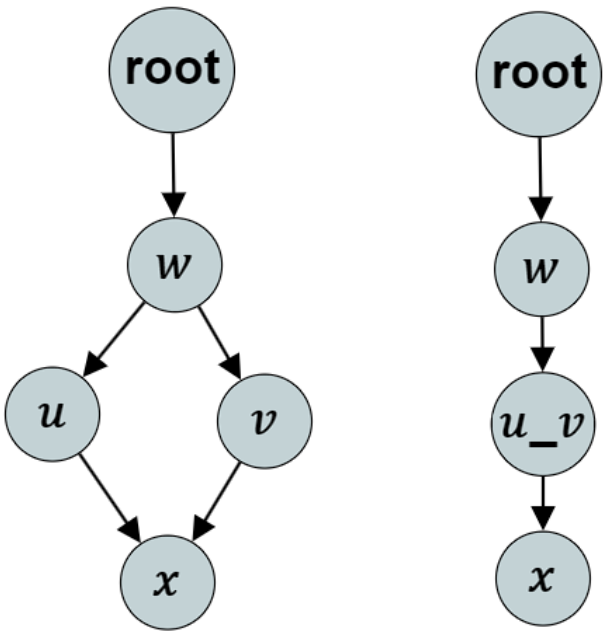
A Redução de Equivalência (ER) é uma operação aplicada em Grafos Acíclicos Direcionados (DAGs) que visa simplificar a estrutura do grafo ao identificar e consolidar vértices equivalentes em um único vértice representante, sem alterar as relações de acessibilidade entre os vértices não-equivalentes do grafo. Esta operação é essencial para reduzir a complexidade e o tamanho dos grafos, facilitando seu processamento, análise e visualização (FAN *et al.*, 2010).

Dado um DAG $G = (V, E)$, onde V representa o conjunto de vértices e E o conjunto de arestas direcionadas, a Redução de Equivalência busca identificar conjuntos de vértices V_i contidos em V nos quais todos os vértices são equivalentes entre si. Dois vértices u e v são considerados equivalentes se, e somente se, para todo vértice x em V, x é acessível a partir de u se e somente se x é acessível a partir de v, e vice-versa. Além disso, u e v devem ter os mesmo vértices pais, ou seja, se um vértice w em V pode acessar u, então w também deve poder acessar v, e vice-versa, como mostra a Figura 4 (a). Após a identificação dos conjuntos de vértices equivalentes, cada conjunto V_i é substituído por um único vértice representante no grafo resultante $G' = (V'$,$E')$ chamado u_v, onde V' é o novo conjunto de vértices após a consolidação e possui u_v, e E' é ajustado para refletir as conexões entre os novos vértices representantes.

O processo de Redução de Equivalência inicia com a identificação de todos os conjuntos de vértices equivalentes em G, com base na análise das relações de acessibilidade entre os vértices. Uma vez identificados, esses conjuntos V_i são substituídos por um único vértice representante em G', preservando as arestas de entrada e saída que refletem as relações de acessibilidade do conjunto original no grafo reduzido, como mostra a Figura 4 (b).

O grafo resultante da Redução de Equivalência, G', representa uma estrutura simplificada com um número potencialmente menor de vértices, preservando, contudo, as relações de acessibilidade essenciais do grafo original G. A ER não apenas facilita a visualização e

Figura 4 – Exemplo de redução equivalente
(a) Vertices equivalentes (b) Vértices fundidos



Fonte: Adaptado de Fan *et al.* (2010).

o processamento do grafo ao reduzir redundâncias mas também simplifica sua estrutura sem comprometer a integridade das relações representadas no DAG.

O trabalho de Fan *et al.* (2010) aborda não apenas a redução de equivalência mas também discute o desafio de correspondência de padrões em grafos, mostrando como a redução pode levar problemas de tempo intratável a soluções de tempo polinomial.

2.2.3 Compactação de Subgrafos

A compactação de subgrafos em Grafos Acíclicos Direcionados é detalhado por Wang e Di (2022). Esta pesquisa investiga a aplicação de técnicas de compactação em subgrafos para a otimização de estruturas de dados complexas. O trabalho detalha os conceitos e algoritmos envolvidos na compactação de subgrafos, proporcionando uma base sobre como essas técnicas podem ser empregadas para simplificar grafos, ao mesmo tempo que preservam suas propriedades essenciais. A eficácia da contração de nós, como meio de reduzir a complexidade dos grafos, é destacada, enfatizando sua utilidade em facilitar análises e otimizações subsequentes.

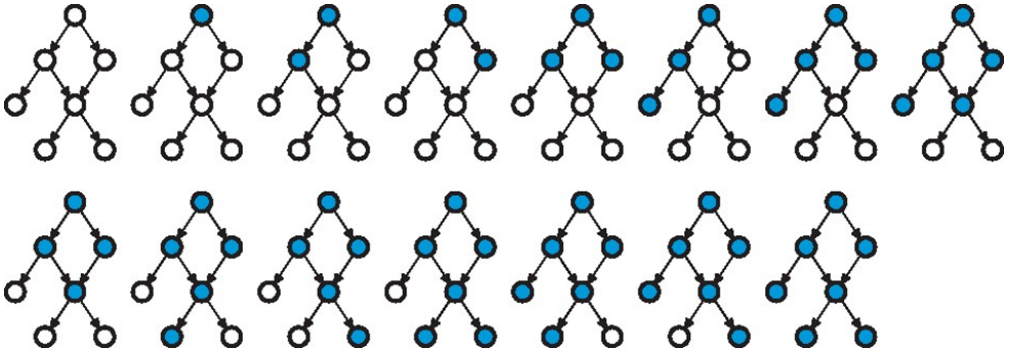
A Compactação de Subgrafos é uma operação aplicada em Grafos Acíclicos Direcionados (DAGs) que visa simplificar a estrutura do grafo ao identificar e consolidar subgrafos consistentes em uma única entidade representativa. Este processo não altera as relações fundamentais de precedência ou acessibilidade entre as entidades consolidadas e as demais partes do grafo. A compactação é utilizada para reduzir a complexidade e o tamanho do grafo, otimizar o

desempenho de operações de processamento e análise, bem como para aprimorar a eficiência na visualização de grandes estruturas de dados.

Seja $G = (V,E)$ um grafo direcionado acíclico, onde V é o conjunto de vértices e E o conjunto de arestas. Define-se um subgrafo consistente $S \subseteq G$ como sendo um subconjunto de G tal que, para quaisquer dois vértices $v, w \in S$, se existe um caminho de v a w em G, então todos os vértices deste caminho também pertencem a S. A compactação de G resulta em um novo grafo $G' = (V',E')$, onde cada vértice $v' \in V'$ representa um subgrafo consistente de G. As arestas em E' são definidas da seguinte forma: existe uma aresta direcionada de v' para w' em G' se e somente se existir pelo menos uma aresta direcionada de um vértice em v' para um vértice em w' em G.

O processo de Compactação de Subgrafos inicia com a identificação de todos os subgrafos consistentes no grafo G. Essa identificação é realizada através da análise das relações de acessibilidade entre os vértices, garantindo que cada subgrafo identificado contém todos os vértices pelos quais é possível transitar dentro do próprio subgrafo sem sair dele, como mostra a Figura 5. Uma vez identificados os subgrafos S_i, cada um deles é substituído por um único vértice representante em G', mantendo as arestas de entrada e saída que refletem as relações de acessibilidade do subgrafo original no grafo reduzido.

Figura 5 – Exemplo de subgrafos possíveis



Fonte: Peng, Jiang e Radivojac (2018).

O grafo resultante da Compactação de Subgrafos, G', apresenta uma estrutura simplificada com um número potencialmente menor de vértices, mas preservando as relações de acessibilidade essenciais do grafo original G. A compactação facilita a visualização e o processamento do grafo ao reduzir redundâncias e simplificar sua estrutura, sem comprometer a integridade das relações representadas no DAG. Este método é particularmente útil para otimizar o desempenho de algoritmos de busca e análise em grandes DAGs, como os utilizados em ontologias biomédicas.

2.2.4 Contração de Arestas

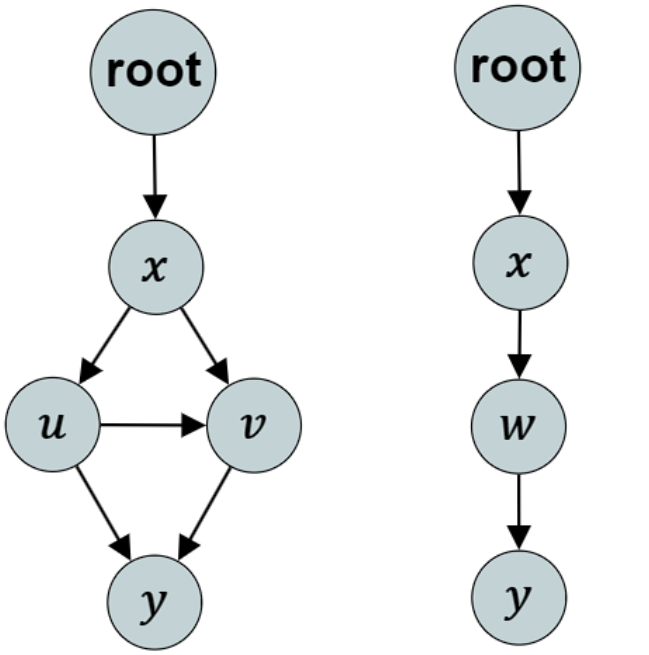
Uma referência na literatura para a Contração de Arestas pode ser encontrada no trabalho de Ibrahim e Tittmann (2022), que explora a reconfiguração de conjuntos independentes em grafos e discute operações como a contração de arestas para transformar grafos. Este trabalho é essencial para compreender como a contração de arestas pode ser usada para simplificar estruturalmente grafos, facilitando tarefas como a otimização de consultas e a manipulação de dados em grafos grandes.

A Contração de Arestas em Grafos Acíclicos Direcionados (DAGs) é uma operação que simplifica a estrutura do grafo ao combinar dois vértices conectados por uma aresta em um único vértice. Essa técnica é importante para reduzir a complexidade do grafo, mantendo a transitividade e as relações de dependência fundamentais entre os vértices restantes.

Seja $G = (V,E)$ um grafo acíclico direcionado. A contração de uma aresta $(u,v) \in E$ em G resulta em um novo grafo $G' = (V',E')$, onde $V' = V \setminus \{u,v\} \cup \{w\}$, como mostra a Figura 6 (a). O vértice w substitui u e v. Para cada aresta (x,u) ou (x,v) em E, G' inclui uma aresta (x,w), e para cada aresta (u,y) ou (v,y) em E, G' inclui uma aresta (w,y).

Figura 6 – Exemplo de contração de aresta

(a) Aresta a contrair **(b) Aresta contraída**



Fonte: Adaptado de Ibrahim e Tittmann (2022).

A operação de Contração de Arestas começa com a escolha de arestas a serem contraídas em G, geralmente aquelas que não resultarão em ciclos quando combinadas. Uma vez selecionada a aresta (u,v), os vértices u e v são fundidos em um novo vértice w, que herda as

arestas de entrada e saída de ambos. Isso reduz o número total de vértices e arestas, simplificando o grafo sem perder a conectividade essencial, como mostra a Figura 6 (b).

O grafo resultante, G', após a contração de arestas, tem menos vértices e pode ter menos arestas, dependendo das arestas duplicadas removidas durante a contração. Esta operação preserva as relações de acessibilidade entre todos os vértices não afetados, simplificando a análise e processamento do grafo enquanto mantém a integridade estrutural.

2.3 Considerações do Capítulo

Este capítulo abordou os principais conceitos relacionados à classificação hierárquica multirrótulo e aos métodos de redução de DAGs. Primeiramente, foram discutidos os conceitos fundamentais de classificação e suas aplicações em diversas áreas. Em seguida, foi explorada a classificação hierárquica, detalhando suas estruturas e características, como o uso de árvores e DAGs. Além disso, foi abordada a classificação hierárquica multirrótulo, destacando a capacidade de atribuir múltiplas classes a uma instância e os algoritmos mais relevantes na literatura.

No contexto da classificação hierárquica multirrótulo, o desenvolvimento de algoritmos que integrem as relações hierárquicas e a complexidade dos DAGs pode aprimorar a precisão e a usabilidade desses sistemas. O algoritmo Clus-HMC, que será utilizado neste trabalho, atende a alguns desses requisitos. A investigação de técnicas de redução de DAGs que facilitem o processamento e a análise de dados em larga escala é importante para avanços na comunidade científica e tecnológica.

Os métodos de redução de DAGs, como a Redução Transitiva, a Redução de Equivalência, a Compactação de Subgrafos e a Contração de Arestas, foram examinados com o objetivo de simplificar as estruturas de grafos, preservando relações essenciais de acessibilidade. Essas técnicas otimizam o processamento de grandes volumes de dados, facilitando a análise e melhorando a eficiência das operações.

3 ABORDAGEM DE REDUÇÃO DE HIERARQUIA DE CLASSES

Este Capítulo detalha o desenvolvimento da abordagem de redução de hierarquia de classes em problemas de classificação multirrótulo. A Seção 3.1 descreve a metodologia de desenvolvimento do trabalho. A Seção 3.2 apresenta o desenvolvimento da abordagem de redução da hierarquia de classes. A Seção 3.3 exemplifica a simulação da abordagem desenvolvida. Por fim, a Seção 3.4 aborda as considerações finais do capítulo.

3.1 Metodologia do trabalho

Para a realização deste trabalho dividiu-se a pesquisa em três etapas principais: 1 - Desenvolvimento da abordagem de redução de hierarquia de classes; 2 - Experimentação; 3 - Avaliação e comparação dos resultados.

1. Etapa 1 - Desenvolvimento da abordagem de redução de hierarquia de classes

   Para o desenvolvimento da abordagem utilizou-se a linguagem de programação Python e a IDE (*Integrated Development Environment*) Visual Studio Code. Na Seção 3.2 serão apresentados detalhes desta etapa da metodologia do trabalho.

2. Etapa 2 - Experimentação

   Nesta etapa foram realizados experimentos com bases de dados de classificação hierárquica multirrótulo adaptadas por Borges (2012), a fim de verificar o funcionamento da abordagem de redução. Primeiramente é realizada a redução da hierarquia e a reestruturação da base de dados. Após a reestruturação da base de dados é feita a avaliação preditiva por meio do classificador Clus-HMC desenvolvido por Vens *et al.* (2008).

3. Etapa 3 - Avaliação e comparação dos resultados

   Após a realização dos experimentos os resultados foram avaliados e comparados com os obtidos por Borges (2012). Nesta avaliação é realizado o testes estatístico de Wilcoxon (WILCOXON, 1992). A análise comparativa permitirá avaliar o desempenho do método e identificar diferenças estatísticas.

3.2 Desenvolvimento da abordagem de redução de hierarquia de classes

A abordagem de redução de hierarquia de classes é composta por um conjunto de algoritmos que interagem para processar e otimizar grafos com base em dados de classificação hierárquica multirrótulo extraídos de arquivos ARFF (*Attribute-Relation File Format*). Os principais processos explicados ao longo desta seção podem ser observados nos pseudocódigos representantes das suas respectivas funções do Apêndice A.

Inicialmente é feita a identificação das característica da bases de dados que estão no formato ARFF. Estes arquivos são comumente usados em ambientes de aprendizado de máquina para armazenar dados de treinamento. O arquivo ARFF inclui um cabeçalho com descrições de atributos e uma seção de dados que lista as instâncias. Outra seção importante neste arquivo é a representação das classes e seus relacionamentos. Por meio dessa informação é possível representar a estrutura do Grafo Acíclico Dirigido (DAG).

O algoritmo trata as relações entre classes como um grafo direcionado, onde cada classe é representada por um vértice e cada relação hierárquica é representada por uma aresta direcionada. Matematicamente, este grafo é denotado como $G = (V,E)$, onde V é o conjunto de vértices (classes) e E é o conjunto de arestas direcionadas que representam relações de dependência hierárquica entre as classes. A função que constrói o grafo, a partir das informações do arquivo ARFF, lê as relações hierárquicas e as transforma em arestas no grafo. Se uma classe A é pré-requisito de uma classe B (indicando $A \rightarrow B$), essa relação é adicionada ao grafo como uma aresta direcionada de A para B. Esta função esta representada no Algoritmo 1. O algoritmo citado é uma tradução para pseudocódigo da Listagem 4 do Apêndice A.

Algoritmo 1 – Construir Grafo a partir de Arquivo ARFF

inserir $arff\_file\_path$
1: $G = (V, E)$ {Inicializar o grafo direcionado com vértices V e arestas E vazios}
2: $V \leftarrow \emptyset$
3: $E \leftarrow \emptyset$
4: $file \leftarrow$ abrir arquivo $arff\_file\_path$
5: **para** cada $line$ em $file$ **faça**
6: **se** $line$.strip().$startswith$(“@$ATTRIBUTEclass$”) **então**
7: $tree\_structure \leftarrow line.split(\text{“}hierarchical\text{”})[1]$
8: $relations \leftarrow tree\_structure.replace(\text{“\{”, “”}).replace(\text{“\}”, “”}).strip().split(',')$
9: **para** cada $relation$ em $relations$ **faça**
10: $source, target \leftarrow relation.strip().split('/')$
11: $source \leftarrow source.strip()$
12: $target \leftarrow target.strip()$
13: $V \leftarrow V \cup \{source, target\}$ {Adicionar vértices ao conjunto V}
14: $E \leftarrow E \cup \{(source, target)\}$ {Adicionar aresta direcionada ao conjunto E}
15: **finaliza para**
16: **break**
17: **finaliza se**
18: **finaliza para**
19: **return** $G = (V, E)$

Fonte: Autoria própria (2024).

As etapas adjacentes à construção do grafo são a captura das informações restantes da base, presente no Algoritmo 2. Ela preserva todas as informações do cabeçalho do arquivo ARFF original. Ao mesmo tempo, a função do Algoritmo 3 captura a seção de dados do arquivo ARFF. Estas informações são importantes pois serão utilizadas posteriormente para a reconstrução do arquivo após a redução da hierarquia de classes. Os algoritmos citados são, respectivamente, traduções para pseudocódigo das Listagem 5 e 6 do Apêndice A .

Algoritmo 2 – Salvar Cabeçalho de Arquivo ARFF

inserir $arff\_file\_path$
1: $header\_content \leftarrow$ "" {Inicializar conteúdo do cabeçalho como string vazia}
2: $file \leftarrow$ abrir arquivo $arff\_file\_path$
3: **para** cada $line$ em $file$ **faça**
4: **se** "@ATTRIBUTE class" in $line$ **então**
5: $header\_content \leftarrow header\_content + line.split(''hierarchical")[0] +'' hierarchical"$
6: **break** {Parar ao encontrar a definição de classe}
7: **finaliza se**
8: $header\_content \leftarrow header\_content + line$ {Adicionar linha ao cabeçalho}
9: **finaliza para**
10: **return** $header\_content$ {Retornar o conteúdo do cabeçalho}

Fonte: Autoria própria (2024).

Algoritmo 3 – Salvar Seção Inferior de Arquivo ARFF

inserir $arff\_file\_path$
1: $global\ bottom\_content$
2: $data\_section \leftarrow False$ {Inicializar a seção de dados como Falso}
3: $file \leftarrow$ abrir arquivo $arff\_file\_path$
4: **para** cada $line$ em $file$ **faça**
5: **se** $line.strip().startswith(\text{"}@DATA\text{"})$ **então**
6: $data\_section \leftarrow True$ {Iniciar a seção de dados ao encontrar "@DATA"}
7: **finaliza se**
8: **se** $data\_section$ **então**
9: $bottom\_content \leftarrow bottom\_content + line$ {Adicionar linha à seção inferior}
10: **finaliza se**
11: **finaliza para**
12: **return** $bottom\_content$ {Retornar o conteúdo da seção inferior}

Fonte: Autoria própria (2024).

Após a criação do grafo, o algoritmo aplica a redução transitiva, utilizando a função 'transitive_reduction' da biblioteca 'Networkx'. Esta etapa simplifica o grafo ao remover arestas redundantes. A execução desta função é mostrada na Linha 13 da função principal descrita na Listagem 3 do Apêndice A.

Após a realização da redução transitiva, a função de redução de equivalência é aplicada para simplificar o grafo por meio da fusão de nós que compartilham os mesmos pais e filhos. Durante o processo de fusão e substituição, um mapeamento é estabelecido para rastrear quais classes são combinadas. Este mapeamento pode ser considerado uma função de mapeamento $f : V \rightarrow V'$, onde V' é um subconjunto de V representando as classes após as fusões. A função f é tal que cada classe em V é mapeada para uma nova classe em V', onde as classes equivalentes são mapeadas para um único representante. O código que realiza esta função é mostrado no Algoritmo 4, este que traduz a Listagem 7 do Apêndice A para pseudocódigo.

Com um grafo reduzido, é necessário ajustar a seção de dados salva anteriormente pela função presente no Algoritmo 3, verificando as fusões ou alterações de classes das amostras. Isso implica na redefinição das linhas de dados, onde cada referência de classe antiga é substi-

Algoritmo 4 – Redução de Equivalência

inserir G {Grafo direcionado}
1: $global\ bottom\_content$
2: $global\ count\_RE$
3: $nodes\_to\_merge \leftarrow \{\}$ {Inicializar dicionário para nós a serem fundidos}
4: $nodes\_by\_degree \leftarrow$ nós de G ordenados por grau de entrada em ordem decrescente
5: **para** cada $node$ em $nodes\_by\_degree$ **faça**
6: $parents \leftarrow$ conjunto de predecessores de $node$
7: $children \leftarrow$ conjunto de sucessores de $node$
8: $key \leftarrow (frozenset(parents), frozenset(children))$
9: **se** key está em $nodes\_to\_merge$ **então**
10: $node\_to\_merge \leftarrow nodes\_to\_merge[key]$
11: **se** G tem $node$ e $node\_to\_merge$ **então**
12: $merged\_node \leftarrow node +$ "_" $+ node\_to\_merge$ {Criar nó fundido}
13: $G \leftarrow nx.contracted\_nodes(G, node, node\_to\_merge, self\_loops =$ False) {Fundir nós}
14: $nx.relabel\_nodes(G, \{node : merged\_node\}, copy =$ False) {Renomear nó fundido}
15: $count\_RE \leftarrow count\_RE + 1$
16: **finaliza se**
17: **senão,**
18: $nodes\_to\_merge[key] \leftarrow node$
19: **finaliza se**
20: **finaliza para**
21: **return** G {Retornar grafo modificado}

Fonte: Autoria própria (2024).

tuída pela sua nova classe correspondente conforme o mapeamento f. Matematicamente, isso é representado por uma transformação dos dados de entrada, onde cada etiqueta de classe c em uma linha de dados é substituída por $f(c)$. A função que realiza estes ajuste está presente no Algoritmo 5, este que traduz a Listagem 8 do Apêndice A para pseudocódigo.

Por fim, é necessário gerar um novo arquivo ARFF, combinando os dados salvos posteriormente, onde está persente o cabeçalho preservado e os dados modificados, com as relações de classes atualizadas no grafo processado. O resultado é um novo arquivo ARFF atualizado que reflete todas as alterações e otimizações realizadas pelo algoritmo. Para a criação desse novo arquivo, é utilizado o código da função presente no Algoritmo 6. O algoritmo citado é uma tradução da Listagem 9 do Apêndice A para pseudocódigo.

Ao longo do processo, é necessário que as transformações preservem a integridade das relações de dependência originalmente representadas no arquivo ARFF. Isso significa que as transformações e simplificações do grafo não devem alterar o significado semântico das relações hierárquicas entre as classes.

Algoritmo 5 – Gerenciar Fusões e Substituições

inserir G {Grafo reduzido}
1: $global\ bottom\_content$
2: $substitution\_map \leftarrow \{\}$ {Inicializar o mapa de substituição}
3: **para** cada $node$ em $G.nodes()$ **faça**
4: **se** $'\_'$ está em $node$ **então**
5: $parts \leftarrow node.split('\_')$
6: **para** cada $part$ em $parts$ **faça**
7: $substitution\_map[part] \leftarrow node$
8: **finaliza para**
9: **finaliza se**
10: **finaliza para**
11: $new\_content \leftarrow$ "$@DATA\backslash n$" {Inicializar novo conteúdo de dados}
12: **para** cada $line$ em $bottom\_content.split('\backslash n')[1:]$ **faça**
13: **se** $line.strip() ==$ "" **então**
14: **continue** {Ignorar linhas vazias}
15: **finaliza se**
16: $parts \leftarrow line.rsplit(',', 1)$
17: $data \leftarrow parts[0]$
18: $classes \leftarrow parts[1]$
19: $new\_classes \leftarrow []$
20: **para** cada cls em $classes.split('@')$ **faça**
21: **se** cls está em $substitution\_map$ **então**
22: $merged\_class \leftarrow substitution\_map[cls]$
23: **se** $all(merged\_class != existing$ para cada $existing$ em $new\_classes)$ **então**
24: $new\_classes.append(merged\_class)$ {Adicionar classe fundida}
25: **finaliza se**
26: **senão,**
27: $new\_classes.append(cls)$
28: **finaliza se**
29: **finaliza para**
30: $new\_content \leftarrow new\_content + data + ',' + '@'.join(new\_classes) + '\backslash n'$ {Atualizar novo conteúdo de dados}
31: **finaliza para**
32: $bottom\_content \leftarrow new\_content$ {Atualizar conteúdo inferior}

Fonte: Autoria própria (2024).

Algoritmo 6 – Reconstruir Arquivo ARFF

inserir $header, Ge, file\_path$
1: $class\_relations \leftarrow$ ",".$join([source +$ "/" $+ target$ para cada $source, target$ em $Ge.edges()])$ {Gerar relações de classes a partir das arestas do grafo}
2: $arff\_content \leftarrow header +$ "" $+ class\_relations +$ "$\backslash n\backslash n$" $+ bottom\_content$ {Combinar cabeçalho, relações de classes e conteúdo inferior}
3: $file \leftarrow$ abrir arquivo $file\_path$ para escrita
4: $file.write(arff\_content)$ {Escrever conteúdo no arquivo}
5: $file.close()$

Fonte: Autoria própria (2024).

3.3 Simulação da abordagem desenvolvida

A simulação descrita nesta seção tem como objetivo principal demonstrar a aplicação prática da abordagem desenvolvida para manipular grafos hierárquicos em problemas de classificação multirrótulo.

Inicialmente, é feita a leitura do arquivo dados.arff, apresentado na Listagem 1. Este arquivo simula uma base de dados de classificação hierárquica multirrótulo. Observa-se na linha 9, desta listagem, o relacionamento hierárquico existente entre a classes (root/AA, root/DD, AA/BB, AA/CC, DD/EE, DD/FF , BB/CC, FF/GG, EE/GG) em que entende-se por essa representação que cada barra representa uma relação de nó pai (esquerda) e nó filho (direita). Para a execução desse procedimento são executados os códigos, comentados na seção anterior, nas Listagem 4, 5 e 6 do Apêndice A.

Listagem 1 – Dados de entrada, arquivo 'dados.arff'

```
1  @RELATION 'example.names'
2
3  @ATTRIBUTE alpha0                                    numeric
4  @ATTRIBUTE alpha1                                    numeric
5  @ATTRIBUTE alpha2                                   numeric
6  @ATTRIBUTE alpha3                                   numeric
7  @ATTRIBUTE alpha4                                   numeric
8  @ATTRIBUTE class                                     hierarchical
9  root/AA,root/DD,AA/BB,AA/CC,DD/EE,DD/FF,BB/CC,FF/GG,EE/GG
10
11 @DATA
12 0.3342,0.4397,0.3651,0.2868,0.308,EE@CC@FF
13 0.3465,0.3401,0.3651,0.2114,0.3243,FF@BB
14 0.1702,0.3159,0.3369,0.2917,0.3772,EE
15 0.2639,0.4005,0.3383,0.3164,0.2931,GG
```

Fonte: Autoria própria (2024).

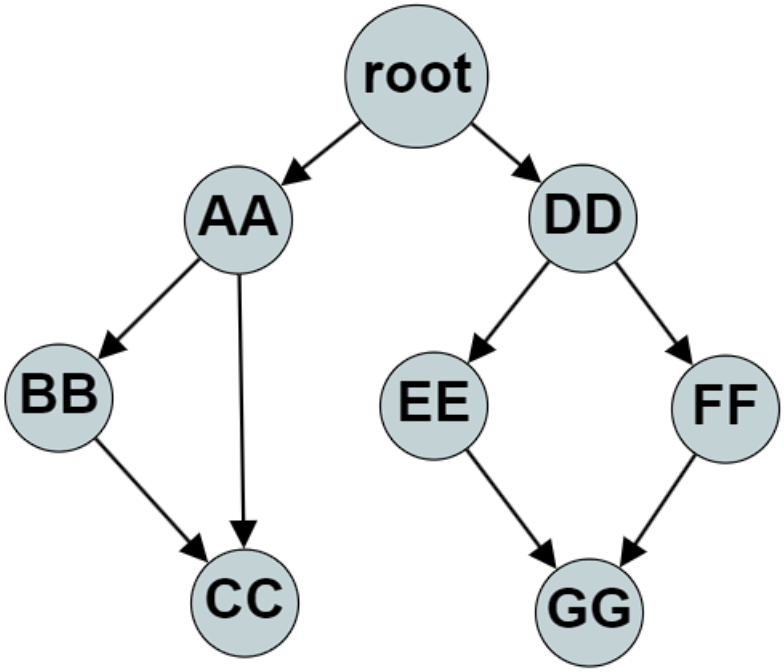
Por meio de uma análise da representação do relacionamento das classes identifica-se a quantidade de classes presentes na base de dados. Neste exemplo tem-se 7 classes (nós) que estão ilustrada na Figura 7. Além disso, outras informações podem ser extraídas como a quantidade de atributos, quantidade máxima de níveis, quantidade mínima e máxima de classes por amostra, quantidade mínima e máxima de amostras por classe. Esta características são mostradas na Tabela 1.

Tabela 1 – Características da base de dados 'dados.arff'

Quant. Amostras	Quant. Atributos	Quant. Classes	Quant. Max. Níveis	Quant. Min/Max Classes por Amostras	Quant. Min/Max Amostras por Classe
4	5	7	3	1/3	0/2

Fonte: Autoria própria (2024).

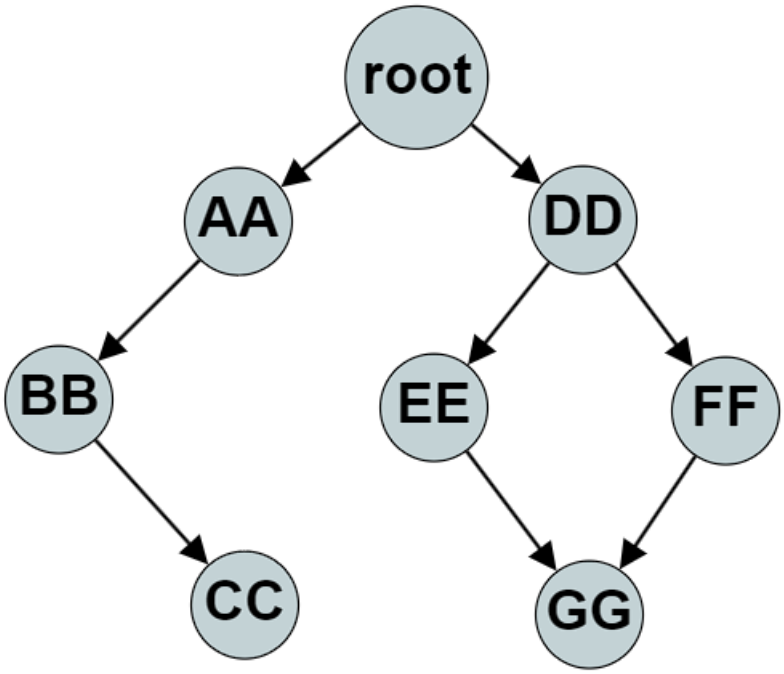
Figura 7 – Arvore hieráquica presente em dados.arff



Fonte: Autoria própria (2024).

Por meio da biblioteca NetworkX (HAGBERG; SCHULT; SWART, 2008), aplica-se a redução transitiva com objetivo de eliminar arestas redundantes. No exemplo o ciclo transitivo, ou relação AA-CC, é excluído. Isso pode ser observado na Figura 8.

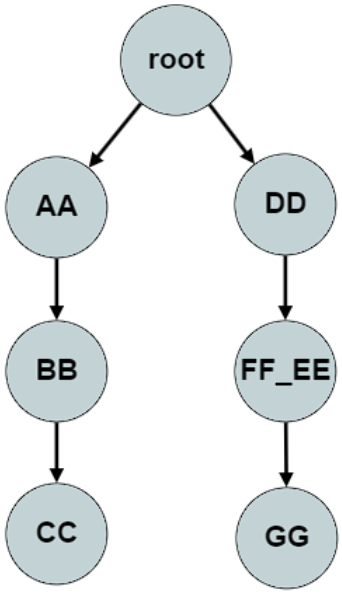
Figura 8 – Arvore hierárquica sem ciclo transitivo



Fonte: Autoria própria (2024).

Na sequência, a redução de equivalência é aplicada representada na Listagem 7 do Apêndice A. Neste exemplo as classes “EE” e “FF” são fundidas na hierarquia dando origem a classe “FF_EE”. A Figura 9 mostra a hierarquia após as reduções.

Figura 9 – Arvore hieráquica presente em new_file.arff



Fonte: Autoria própria (2024).

Com a redução efetuada, o atributo classe da base de dados deve ser ajustado. Isso ocorre devido a redução equivalente que altera as classes fundindo-as, neste exemplo, as classes denominadas “EE” e “FF” se tornam apenas uma classe denominada “FF_EE”, assim todas as amostras que possuíam uma dessas classes devem ser reajustadas para representar a classe fundida, para isso é utilizada o código correspondente a Listagem 8 do Apêndice A. Além disso, é possível notar que o número total de classes diminuiu de 7 (sete) para 6 (seis).

Após esses procedimentos, o arquivo ARFF é reconstruído utilizando o código na Listagem 9 do Apêndice A. Este novo arquivo é representado na Listagem 2 e incorpora todas as modificações realizadas de redução da hierarquia de classes nos dados de classificação hierárquica multirrótulo.

É possível observar que ao realizar a redução de classes da hierarquia, a quantidade de classes por amostras diminuiu. Isso ocorre pois ao fundir classes, a amostra que continha duas classes fundidas passa a ter apenas uma. Por exemplo, a amostra que possuía as classes “EE” e “FF” (Linha 12 da Listagem 1), agora possui apenas a classe “FF_EE” (Linha 12 da Listagem 2), diminuindo assim o número de classes por amostra.

Outra observação é em relação a quantidade de amostras por classe que também teve alteração. Neste caso, as amostras que possuíam uma das classes que foram fundidas passam a compartilhar a classe fundida. Por exemplo, as amostras que possuíam “EE” e/ou “FF” (Linha 12, 13 e 14 da Listagem 1) serão contabilizadas para o número de amostras por classe da

classe “FF_EE”(Linha 12, 13 e 14 da Listagem 2). As características da nova base de dados são apresentadas na Tabela 2.

Tabela 2 – Características da base de dados ’new_file.arff’

Quant. Classes	Quant. Max. Níveis	Quant. Min/Max Classes por Amostras	Quant. Min/Max Amostras por Classe
6	3	1/2	0/3

Fonte: Autoria própria (2024).

Listagem 2 – Dados de saída, arquivo ’new_file.arff’

```
1  @RELATION 'example.names'
2
3  @ATTRIBUTE alpha0                                    numeric
4  @ATTRIBUTE alpha1                                    numeric
5  @ATTRIBUTE alpha2                                   numeric
6  @ATTRIBUTE alpha3                                   numeric
7  @ATTRIBUTE alpha4                                   numeric
8  @ATTRIBUTE class                                     hierarchical
9  root/DD,root/AA,AA/BB,DD/FF_EE,BB/CC,FF_EE/GG
10
11 @DATA
12 0.3342,0.4397,0.3651,0.2868,0.308,FF_EE@CC
13 0.3465,0.3401,0.3651,0.2114,0.3243,FF_EE@BB
14 0.1702,0.3159,0.3369,0.2917,0.3772,FF_EE
15 0.2639,0.4005,0.3383,0.3164,0.2931,GG
```

Fonte: Autoria própria (2024).

3.4 Considerações do Capítulo

Neste Capítulo foi apresentado a descrição da abordagem de redução da hierarquia de classes destacando desde a construção inicial do grafo até a reconstrução final do arquivo ARFF, com a aplicação de técnicas de redução de DAGs para otimizar a representação das relações entre classes.

A simulação, baseada em um exemplo prático, demonstrou a aplicabilidade do algoritmo, ilustrando as transformações realizadas nos dados e a estrutura resultante do grafo.

4 EXPERIMENTOS E RESULTADOS

Este Capítulo detalha os experimentos realizados para avaliar a abordagem de redução da hierarquia de classes em problemas de classificação hierárquica multirrótulo. A Seção 4.1 descreve as bases de dados utilizadas. Na Seção 4.2, os resultados obtidos são detalhados, observando-se as mudanças estruturais e o impacto na eficiência dos classificadores. A Seção 4.3 compara os dados pré e pós-processamento, discutindo a efetividade da abordagem desenvolvida. Por fim, a Seção 4.4 apresenta as considerações finais do capítulo.

4.1 Base de dados

Para a execução dos experimentos, foram empregadas dez bases de dados Gene Ontology (GO), as mesmas utilizadas em estudos por Vens *et al.* (2008) e Borges (2012). A Tabela 3 detalha algumas características dessas bases.

Tabela 3 – Características das Bases de Dados GO

Base de Dados	Quant. Amostras	Quant. Atributos	Quant. Classes	Quant. Max. Níveis	Quant. Min/Max Classes por Amostras	Quant. Min/Max Amostras por Classe
Cellcycle	3751	77	4125	13	3/28	0/785
Church	3749	27	4125	13	3/28	0/786
Derisi	3719	63	4119	13	3/28	0/781
Eisen	2418	79	3573	13	3/28	0/492
Expr	3773	551	4131	13	3/28	0/789
Gasch1	3758	173	4125	13	3/28	0/786
Gasch2	3773	52	4131	13	3/28	0/789
Pheno	1586	69	3127	13	3/21	0/388
Seq	3900	478	4133	13	3/28	0/791
Spo	3697	80	4119	13	3/28	0/775

Fonte: Adaptado de Borges (2012).

4.2 Resultados Experimentais

Os resultados obtidos a partir da aplicação da abordagem de redução nas bases de dados selecionadas são detalhados nesta seção. A análise concentra-se na estrutura das bases após a redução e nas implicações da aplicação do algoritmo em termos de simplificação estrutural dos dados.

A Tabela 4 exibe as características das bases de dados após a aplicação da abordagem de redução. Observa-se que a quantidade classe das bases de dados foi reduzida. Não houve alteração no valor da quantidade máxima de níveis hierárquicos, com todas as bases de dados apresentando 13 níveis. Percebe-se que a quantidade mínima e máxima de classes por amos-

tra permaneceu a mesma em 9 (nove) bases, apenas na base de dados Pheno houve uma redução. No que se refere a quantidade máxima de amostras por classe nota-se um aumento. Esse fenômeno ocorre porque, durante o processo de fusão de classes equivalentes, as amostras que anteriormente estavam distribuídas entre várias classes passaram a ser contabilizadas sob uma única classe resultante da fusão. Em outras palavras, ao combinar classes com características e relações semelhantes, as instâncias associadas a essas classes se aglomeraram, aumentando o total de amostras representadas pela nova classe unificada. Esse aumento é uma consequência da abordagem de redução de hierarquia.

Tabela 4 – Características das Bases de Dados Reduzida

Base de Dados	Quant. Classes	Quant. Max. Níveis	Quant. Min/Max Classes por Amostras	Quant. Min/Max Amostras por Classe
Cellcycle	3789	13	3/28	0/832
Church	3789	13	3/28	0/830
Derisi	3785	13	3/28	0/827
Eisen	3297	13	3/28	0/569
Expr	3795	13	3/28	0/836
Gasch1	3789	13	3/28	0/834
Gasch2	3795	13	3/28	0/836
Pheno	2886	13	3/20	0/401
Seq	3796	13	3/28	0/863
Spo	3785	13	3/28	0/819

Fonte: Autoria própria (2024).

A Tabela 5 apresenta os números de reduções equivalentes e reduções transitivas aplicadas representadas por a quantidade de nós e arestas reduzidas. A falta de presença de arestas transitivas reduzidas também mostra que a estrutura adotada para a criação das bases de dados evita ciclos transitivos.

Tabela 5 – Números de classes (nós) e arestas reduzidas

Base de Dados	Quant. Nós	Quant. Arestas
Cellcycle	336	0
Church	336	0
Derisi	334	0
Eisen	276	0
Expr	336	0
Gasch1	336	0
Gasch2	336	0
Pheno	241	0
Seq	337	0
Spo	334	0

Fonte: Autoria própria (2024).

Após a redução e reestruturação da hierarquia de classes e das bases de dados, estas foram avaliadas utilizando o classificador hierárquico multirrótulo Clus-HMC (VENS *et al.*, 2008). A escolha do Clus-HMC se deve à sua ampla utilização em experimentos descritos na literatura. Este classificador emprega a precisão e o recall como principais métricas de avaliação.

A Tabela 6, demonstra os valores de AUPRC (Área sob a Curva de Precisão/Recall) encontrado após a redução. O AUPRC é uma medida que combina duas métricas importantes: a precisão e o recall. A precisão é a proporção de identificações positivas feitas corretamente pelo modelo, enquanto o recall é a proporção de casos positivos reais que foram corretamente identificados. Portanto, o AUPRC fornece uma representação numérica de quão bem o modelo consegue identificar todas as classes positivas, balanceando tanto a precisão quanto o recall.

Tabela 6 – Resultados do classificador Clus-HMC

Base de Dados	AUPRC
Cellcycle	0.4396
Church	0.4526
Derisi	0.4386
Eisen	0.4565
Expr	0.4736
Gasch1	0.4573
Gasch2	0.4391
Pheno	0.4220
Seq	0.4708
Spo	0.4688

Fonte: Autoria própria (2024).

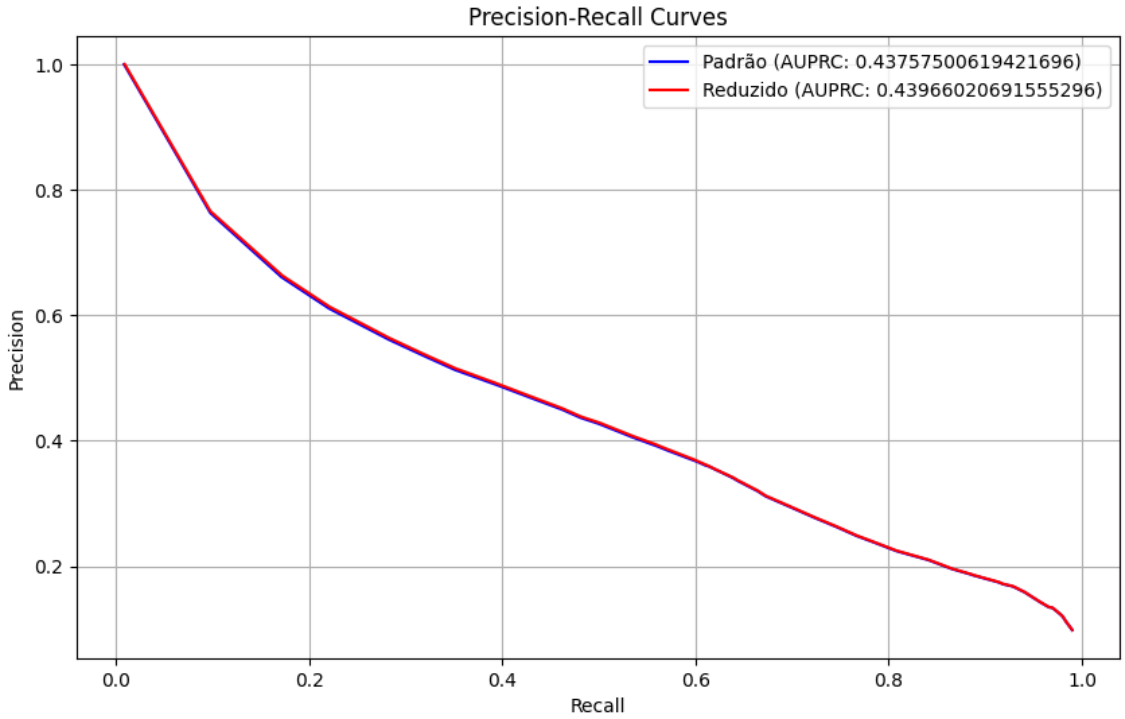
A Figura 10 exibe a curva de Precisão/Recall para a base de dados GOCellcycle. Esta curva é uma ferramenta gráfica para avaliar o desempenho de um modelo de classificação. Ela traça o recall (também conhecido como sensibilidade) no eixo horizontal contra a precisão no eixo vertical. Uma área maior sob esta curva indica um melhor desempenho geral do modelo, refletindo uma alta precisão mantida mesmo com o aumento do recall.

4.3 Comparação dos resultados

A Tabela 5 apresenta que a redução da complexidade do grafo foi efetiva, com uma diminuição significativa no número de classes em todas as bases de dados analisadas. Importante notar que o número de arestas transitivas reduzidas foi zero, indicando que a estrutura original das bases já evitava ciclos transitivos.

Como mencionado anteriormente, observou-se um aumento no número máximo de amostras por classe após a redução da base de dados. Esse aumento resulta do processo de combinação de classes equivalentes, onde amostras que antes estavam espalhadas por diversas classes agora são agrupadas sob uma única classe derivada dessa fusão.

Figura 10 – Curva de Precisão/Recall da base GOCellcycle



Fonte: Autoria própria (2024).

Observando a Figura 10, a curva azul representa o desempenho do modelo utilizando a base de dados original, enquanto a curva vermelha demonstra o desempenho com a base de dados reduzida, ambas avaliadas sob a métrica AUPRC. Pode-se notar que a curva vermelha (base reduzida) está consistentemente acima da curva azul (base original) ao longo de todo o espectro de recall, indicando uma melhoria geral na precisão do classificador, mesmo que singela, quando o recall é aumentado. Este comportamento sugere que a redução da complexidade das bases de dados contribuiu para uma melhor generalização do modelo em identificar as classes corretas, mesmo em situações de maior exigência (maior recall).

A Tabela 7 compara os valores de AUPRC antes e após a aplicação do método de redução. Observa-se um resultado preditivo superior aos obtidos por Borges e Nievola (2012) nos valores de AUPRC em todas as bases de dados. Embora a média da diferença tenha sido na ordem de 0.00236, os resultados indicam uma tendência de aumento na eficiência do classificador após a redução de hierarquia de classes da base de dados. Este resultado é significativo, considerando que mesmo pequenas melhorias na área sob a curva de precisão/recall podem representar um avanço no desempenho de classificadores em tarefas complexas como a classificação hierárquica multirrótulo.

Para avaliar o impacto do algoritmo de redução nas características das bases de dados, foi aplicado o teste de Wilcoxon (WILCOXON, 1992), um teste não paramétrico usado para comparar duas amostras pareadas. O teste de Wilcoxon utilizado foi o bicaudal, o que significa que ele avalia se existem diferenças estatisticamente significativas nas medianas das duas amostras em qualquer direção, sem especificar se uma mediana é maior ou menor que a outra. Esse

Tabela 7 – Resultados do Classificador Clus-HMC AUPRC

Base de Dados	Original (BORGES, 2012)	Base Reduzida
Cellcycle	0.4375	0.4396
Church	0.4508	0.4526
Derisi	0.4362	0.4386
Eisen	0.4541	0.4565
Expr	0.4713	0.4736
Gasch1	0.4546	0.4573
Gasch2	0.4367	0.4391
Pheno	0.4192	0.4220
Seq	0.4684	0.4708
Spo	0.4665	0.4688

Fonte: Autoria própria (2024).

teste é particularmente útil quando não se pode assumir que os dados seguem uma distribuição normal. Os dados do AUPRC (Área sob a Curva de Precisão-Recall) obtidos e os presentes no trabalho de Borges (2012) foram comparados utilizando este teste para determinar se a redução da hierarquia trouxe melhorias significativas nos resultados.

A estatística do teste de Wilcoxon foi 0.0, e o *p-value* obtido foi 0.001953125. Como o *p-value* é menor que o nível de significância de 0.05, rejeita-se a hipótese nula, sugerindo que as diferenças observadas são estatisticamente significativas, ou seja, é improvável que tais diferenças tenham ocorrido por acaso dado o nível de significância estabelecido.

Esses resultados indicam que a melhoria nos valores de AUPRC após a redução de hierarquia de classes é estatisticamente significativa. Em outras palavras, o método de redução aplicado resultou em uma performance aprimorada do classificador Clus-HMC, demonstrando que a simplificação estrutural das bases de dados contribuiu efetivamente para um melhor desempenho na classificação hierárquica multirrótulo.

Portanto, a aplicação do teste de Wilcoxon forneceu uma base para afirmar que as alterações na estrutura das bases de dados, promovidas pela abordagem de redução de hierarquia, têm um impacto positivo e significativo na eficácia do classificador, validando a abordagem proposta neste trabalho.

4.4 Considerações do Capítulo

Este Capítulo discutiu os experimentos realizados e os resultados obtidos por meio da aplicação do método desenvolvido de redução da hierarquia de classes em problemas de classificação hierárquica multirrótulo. As análises foram focadas tanto nas mudanças estruturais dos grafos quanto no desempenho dos classificadores em termos de precisão, recall e AUPRC, antes e após o processo de redução.

A partir dos resultados apresentados, observa-se que a redução estrutural dos grafos, evidenciada pela significativa diminuição no número de classes (nós) e pela manutenção das

arestas, sugere uma simplificação das bases de dados. Importante destacar que a ausência de arestas transitivas reduzidas reforça a qualidade inicial das estruturas de dados, que já evitavam ciclos transitivos. Este aspecto é importante, pois valida as estruturas da base de dados originais.

Quanto ao desempenho do classificador, a melhoria nos valores de AUPRC, é indicativa de um benefício real. O aumento da área sob as curvas de Precisão/Recall para as bases de dados reduzidas sugere que a simplificação contribuiu positivamente para a eficiência dos classificadores. Este resultado é especialmente significativo em contextos de aplicação prática, onde mesmo pequenas melhorias na precisão e no recall podem ter impactos consideráveis no desempenho geral dos sistemas de classificação.

Além disso, as análises estatísticas complementares, utilizando o teste de Wilcoxon, corroboraram que as alterações na estrutura das classes são significativas do ponto de vista estatístico, fortalecendo as conclusões sobre a eficiência do método de redução proposto.

Em suma, os resultados experimentais e as análises apresentadas demonstram que o método de redução não só simplificou a estrutura das bases de dados de forma efetiva, como também melhorou o desempenho dos classificadores. Este capítulo ressalta a importância de estruturas de dados otimizadas para a melhoria contínua dos modelos de classificação hierárquica multirrótulo, apontando para futuras investigações que possam explorar outras dimensões de otimização em contextos semelhantes.

5 CONCLUSÃO

Este trabalho desenvolveu uma abordagem para a redução da hierarquia de classes em problemas de classificação hierárquica multirrótulo, com foco em estruturas do tipo DAG. Através da implementação da abordagem de redução e da realização de experimentos comparativos, foi possível avaliar a eficiência da abordagem proposta em melhorar a performance dos classificadores existentes na literatura.

Entre as vantagens do método desenvolvido, destaca-se a capacidade de reduzir a complexidade computacional dos problemas de classificação, facilitando o processamento e aumentando a precisão dos modelos de classificação. Contudo, as limitações observadas incluem a necessidade de ajustes finos na parametrização do algoritmo para maximizar sua eficácia em diferentes conjuntos de dados, exigindo um conhecimento aprofundado das particularidades de cada aplicação.

Os experimentos comparativos realizados demonstraram que a abordagem proposta é eficaz em melhorar a performance dos classificadores, evidenciado por melhorias nos valores de AUPRC, que indicam um benefício real na precisão e recall dos modelos.

Em relação à contribuição acadêmica e profissional, este trabalho oferece uma nova perspectiva sobre a simplificação de problemas de classificação hierárquica, com implicações práticas que podem beneficiar áreas como bioinformática, reconhecimento de imagens e classificação de texto. Para a sociedade, a melhoria na precisão dos classificadores pode resultar em sistemas de informação mais eficazes e acessíveis.

Concluindo, os objetivos propostos foram alcançados, demonstrando que a redução da hierarquia de classes pode efetivamente melhorar o desempenho dos sistemas de classificação hierárquica multirrótulo. Para trabalhos futuros, sugere-se explorar a aplicação desta abordagem em outras estruturas de dados complexas e realizar um estudo comparativo com outras técnicas de pré-processamento.

5.1 Trabalhos Futuros

Para futuros desenvolvimentos, uma perspectiva é a criação de um novo método matemático para a redução de classes em problemas de classificação hierárquica multirrótulo. Esse novo método buscaria não apenas simplificar a complexidade das estruturas de dados, mas também otimizar o equilíbrio entre precisão e eficiência computacional. Explorando novas teorias matemáticas e algoritmos, tal abordagem poderia oferecer uma solução mais robusta e adaptável, adequada para enfrentar os desafios apresentados por conjuntos de dados extremamente grandes e complexos, abrindo novas fronteiras no campo da aprendizagem de máquina e da análise de dados.

REFERÊNCIAS

AHO, A. V.; GAREY, M. R.; ULLMAN, J. D. The transitive reduction of a directed graph. **SIAM Journal on Computing**, SIAM, v. 1, n. 2, p. 131–137, 1972.

BISHOP, C. M.; NASRABADI, N. M. **Pattern recognition and machine learning**. [*S.l.*]: Springer, 2006. v. 4.

BORGES, H. B. **Classificador hierárquico multirrótulo usando uma rede neural competitiva**. 2012. Tese (Tese) — Pontifícia Universidade Católica do Paraná, Curitiba, 2012.

BORGES, H. B.; JR, C. N. S.; NIEVOLA, J. C. An evaluation of global-model hierarchical classification algorithms for hierarchical classification problems with single path of labels. **Computers & Mathematics with Applications**, Elsevier, v. 66, n. 10, p. 1991–2002, 2013.

BORGES, H. B.; NIEVOLA, J. C. Multi-label hierarchical classification using a competitive neural network for protein function prediction. *In*: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN). **Proceedings [...]**. Brisbane, QLD, Australia, 2012. p. 1–8.

BOUTELL, M. R. *et al.* Learning multi-label scene classification. **Pattern recognition**, Elsevier, v. 37, n. 9, p. 1757–1771, 2004.

CERRI, R. **Técnicas de classificação hierárquica multirrótulo**. 2010. Tese (Doutorado) — Universidade de São Paulo, 2010.

CERRI, R.; BARROS, R. C.; CARVALHO, A. C. de. Hierarchical multi-label classification using local neural networks. **Journal of Computer and System Sciences**, v. 79, n. 5, p. 686–699, 2013.

CERRI, R. *et al.* Reduction strategies for hierarchical multi-label classification in protein function prediction. **BMC bioinformatics**, Springer, v. 17, p. 1–24, 2016.

FAN, W. *et al.* Graph pattern matching: From intractable to polynomial time. *In*: VLDB ENDOWMENT (PVLDB). **Proceedings [...]**. Singapore: Very Large Data Base Endowment Inc., 2010. v. 3, n. 1, p. 264–275.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. **NetworkX**. 2008. Python software for network analysis. Disponível em: https://networkx.github.io/.

IBRAHIM, H.; TITTMANN, P. Edge contraction and line graphs. **arXiv preprint arXiv:2212.10354**, 2022.

PENG, Y.; JIANG, Y.; RADIVOJAC, P. Enumerating consistent sub-graphs of directed acyclic graphs: an insight into biomedical ontologies. **Bioinformatics**, v. 34, n. 13, p. i313–i322, 06 2018.

READ, J.; PFAHRINGER, B.; HOLMES, G. Multi-label classification using ensembles of pruned sets. *In*: IEEE INTERNATIONAL CONFERENCE ON DATA MINING. **Proceedings [...]**. Pisa, Italy, 2008. p. 995–1000.

SADAT, M.; CARAGEA, C. Hierarchical multi-label classification of scientific documents. **arXiv preprint arXiv:2211.02810**, 2022.

SILLA, C.; FREITAS, A. A survey of hierarchical classification across different application domains. **Data Mining and Knowledge Discovery**, v. 22, p. 31–72, 01 2011.

VENS, C. *et al.* Decision trees for hierarchical multi-label classification. **Machine learning**, Springer, v. 73, p. 185–214, 2008.

VIEIRA, R. O.; BORGES, H. B. Dimensionality reduction for hierarchical multi-label classification: A systematic mapping study. **JUCS: Journal of Universal Computer Science**, v. 30, n. 1, 2024.

WANG, Z.; DI, Y. Cluster expansion method for critical node problem based on contraction mechanism in sparse graphs. **IEICE TRANSACTIONS on Information and Systems**, The Institute of Electronics, Information and Communication Engineers, v. 105, n. 6, p. 1135–1149, 2022.

WILCOXON, F. Individual comparisons by ranking methods. *In*: **Breakthroughs in statistics: Methodology and distribution**. [*S.l.*]: Springer, 1992. p. 196–202.

ZHANG, M.-L.; ZHOU, Z.-H. Multilabel neural networks with applications to functional genomics and text categorization. **IEEE transactions on Knowledge and Data Engineering**, IEEE, v. 18, n. 10, p. 1338–1351, 2006.

ZHOU, J. *et al.* Dag reduction: Fast answering reachability queries. *In*: ACM INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. **Proceedings [...]**. [*S.l.*], 2017. p. 375–390.

ZHOU, Z.-H. Abductive learning: towards bridging machine learning and logical reasoning. **Science China. Information Sciences**, Springer Nature BV, v. 62, n. 7, p. 76101, 2019.

APÊNDICE A – Funções em Python

Listagem 3 – Função principal

```
import networkx as nx
bottom_content = ""
count_RE = 0
count_RT = 0

def main(arff_file_path):
    global bottom_content

    G = build_graph_from_arff(arff_file_path)
    header_content = save_header_from_arff(arff_file_path)
    bottom_content = save_bottom_from_arff(arff_file_path)

    Gt = nx.transitive_reduction(G)
    Ge = equivalence_reduction(Gt)

    manage_merges_and_substitutions(Ge)
    new_arff_file_path = "new_file.txt"
    rebuild_arff_file(header_content, Ge, new_arff_file_path)
```

Fonte: Autoria própria (2024).

Listagem 4 – Função que constrói a DAG

```
def build_graph_from_arff(arff_file_path):
    G = nx.DiGraph()
    with open(arff_file_path, 'r') as file:
        for line in file:
            if line.strip().startswith("@ATTRIBUTE class"):
                tree_structure = line.split("hierarchical")[1]
                relations = tree_structure.replace("{", "").replace("}", "")
    .strip().split(',')
                for relation in relations:
                    source, target = relation.strip().split('/')
                    G.add_edge(source.strip(), target.strip())
                break
    return G
```

Fonte: Autoria própria (2024).

Listagem 5 – Função que salva o cabeçalho

```
1  def save_header_from_arff(arff_file_path):
2      header_content = ""
3      with open(arff_file_path, 'r') as file:
4          for line in file:
5              if "@ATTRIBUTE class" in line:
6                  header_content += line.split("hierarchical")[0] + "
       hierarchical"
7                  break
8              header_content += line
9      return header_content
```

Fonte: Autoria própria (2024).

Listagem 6 – Função que salva os dados

```
1  def save_bottom_from_arff(arff_file_path):
2      global bottom_content
3      data_section = False
4      with open(arff_file_path, 'r') as file:
5          for line in file:
6              if line.strip().startswith("@DATA"):
7                  data_section = True
8              if data_section:
9                  bottom_content += line
10     return bottom_content
```

Fonte: Autoria própria (2024).

Listagem 7 – Função da redução equivalente

```
def equivalence_reduction(G):
    global bottom_content
    global count_RE
    nodes_to_merge = {}
    nodes_by_degree = sorted(G.nodes(), key=lambda node: G.in_degree(node),
    reverse=True)

    for node in nodes_by_degree:
        parents = set(G.predecessors(node))
        children = set(G.successors(node))
        key = (frozenset(parents), frozenset(children))

        if key in nodes_to_merge:
            node_to_merge = nodes_to_merge[key]
            if G.has_node(node) and G.has_node(node_to_merge):
                merged_node = f"{node}_{node_to_merge}"
                G = nx.contracted_nodes(G, node, node_to_merge, self_loops=
    False)
                nx.relabel_nodes(G, {node: merged_node}, copy=False)
                count_RE +=1
        else:
            nodes_to_merge[key] = node
    return G
```

Fonte: Autoria própria (2024).

Listagem 8 – Função que atualiza as classes

```
def manage_merges_and_substitutions(G):
    global bottom_content

    substitution_map = {}
    for node in G.nodes():
        if '_' in node:
            parts = node.split('_')
            for part in parts:
                substitution_map[part] = node

    new_content = "@DATA\n"
    for line in bottom_content.split('\n')[1:]:
        if line.strip() == "":
            continue
        parts = line.rsplit(',', 1)
        data = parts[0]
        classes = parts[1]
        new_classes = []
        for cls in classes.split('@'):
            if cls in substitution_map:
                merged_class = substitution_map[cls]
                if all(merged_class != existing for existing in new_classes)
:
                    new_classes.append(merged_class)
            else:
                new_classes.append(cls)
        new_content += data + ',' + '@'.join(new_classes) + '\n'

    bottom_content = new_content
```

Fonte: Autoria própria (2024).

Listagem 9 – Função que cria o nova base

```
def rebuild_arff_file(header, Ge, file_path):
    class_relations = ",".join([f"{source}/{target}" for source, target in
Ge.edges()])
    arff_content = header + " " + class_relations + "\n\n" + bottom_content
    with open(file_path, 'w') as file:
        file.write(arff_content)
```

Fonte: Autoria própria (2024).