UNIVERSIDADE FEDERAL DE OURO PRETO

Relatório Técnico

Aplicação de algoritmos genéticos para o problema de seleção de atributos em classificação hierárquica.

Guilherme Oliveira Santos.
Orientadora: Helen de Cassia Sousa da Costa Lima

Relatório técnico submetido como requisito para o encerramento do projeto pró-ativa.

Outubro de 2024

Sumário

List of Algorithms

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

hF *Hierarchical F-Measure*

NN *Neural Networks*

AG *Algoritmo Genético*

GDA Grafo Direcionado Acíclico

ML *Multi-label*

S.A Seleção de Atributos

SL *Single-label*

GMNB *Global Model Naive Bayes*

GAwNN *Genetic Algorithm with Neural Networks*

GAwNB *Genetic Algorithm with Global Model Naive Bayes*

MLNP *Mandatory Leaf Node Prediction*

NMLNP *Non-Mandatory Leaf Node Prediction*

KDD *Knowledge Discovery in Databases*

CV5 *Cross-validation 5 fold*

Capítulo 1

Introdução

Nas últimas décadas, as técnicas de análise de dados se tornaram o principal foco de atenção devido ao enorme aumento na geração e armazenamento de dados [20].

Transformar esses dados em informações úteis, novas, válidas e compreensíveis apresenta novos desafios para os pesquisadores [10]. Dado que essa tarefa não é trivial. Estratégias automatizadas são necessárias para a análise dos dados. Nesse contexto, o processo de Descoberta de Conhecimento em Bancos de Dados (*Knowledge Discovery in Databases* (KDD)) é frequentemente adotado [7], sendo composto por três etapas principais: pré-processamento de dados, mineração de dados e avaliação de resultados.

No pré-processamento de dados, a Seleção de Atributos (S.A) é uma etapa crucial que visa identificar o máximo possível de atributos relevantes, com o objetivo de melhorar o desempenho das técnicas de mineração de dados [1]. Os principais benefícios da seleção de atributos incluem a melhoria da capacidade preditiva dos classificadores e a redução do tempo de execução dos processos de classificação [10].

Entre as tarefas de mineração de dados, a classificação é uma das mais destacadas pela comunidade científica [8]. A classificação visa prever o(s) rótulo(s) de classe de um objeto com base em seus atributos. Diversos tipos de problemas de classificação são encontrados na literatura, cada um com seu próprio nível de complexidade. Nos problemas de classificação plana, cada exemplo do conjunto de dados é atribuído a uma ou mais classes que não possuem relações hierárquicas entre si. Em contraste, problemas de classificação hierárquica envolvem classes organizadas naturalmente em hierarquias, representadas por uma árvore ou um Grafo Direcionado Acíclico (GDA).

Em problemas de classificação hierárquica, diferentes abordagens são empregadas para lidar com a hierarquia de classes [3]. Nos modelos de abordagem local, a classificação é realizada utilizando um conjunto de classificadores planos. Já nos modelos de abordagem global, um único classificador é projetado para considerar a hierarquia de classes como um todo. Além disso, os métodos de classificação hierárquica podem prever diferentes caminhos de rótulos dentro da hierarquia [3]. Alguns métodos são restritos a prever apenas um único caminho de rótulos *Single-label* (SL), enquanto outros podem prever vários caminhos de rótulos, *Multi-label* (ML).

1.1 Objetivos

Este trabalho busca desenvolver um novo método de seleção de atributos que integra a meta-heurística AG com uma Rede Neural Artificial, utilizando um classificador hierárquico de modelo global para avaliar subconjuntos de atributos. Além disso, serão realizados testes estatísticos para validar o método proposto, visando reduzir o número de atributos necessários enquanto se maximiza a qualidade do conjunto de dados encontrado.

1.1.1 Objetivos Específicos

Os objetivos específicos deste estudo incluem:

- Executar uma revisão literária abrangente para identificar os parâmetros mais recentes e eficientes empregados no AG e como podemos aplicar essa heurística em classificação hierárquica.
- Implementar um AG com os parâmetros selecionados, visando avaliar a capacidade de melhoria da S.A em classificação hierárquica.
- Avaliar a aplicabilidade de *Neural Networks* (NN) como método de avaliação de aptidão no AG proposto.
- Validar e comparar a eficiência dos algoritmos desenvolvidos.

Capítulo 2

Referencial Teórico

Neste capítulo, abordaremos uma análise detalhada dos conceitos teóricos fundamentais e das referências que orientaram o desenvolvimento deste estudo. Posteriormente, forneceremos uma contextualização dos métodos e tecnologias aplicados ao longo da pesquisa. Por fim, conduziremos uma análise descritiva abrangente dos algoritmos escolhidos para este estudo que sustentará todo o desenvolvimento e as conclusões deste trabalho.

2.1 Classificação Hierárquica

A maior parte dos estudos em classificação na área de mineração de dados foca em problemas de classificação plana, onde as classes são tratadas como entidades independentes entre si. No entanto, em muitas aplicações do mundo real, as classes que rotulam as instâncias estão organizadas em estruturas hierárquicas [14].

2.1.1 Métodos de Classificação Hierárquica

Diferentes aspectos podem caracterizar os métodos de classificação hierárquica [14]. O primeiro está relacionado ao tipo de estrutura hierárquica. A Figura 2.1 apresenta uma árvore e a Figura 2.2 um exemplo de GDA, onde os nós representam as classes e as arestas indicam um relacionamento entre elas. Basicamente, em um estrutura em árvore, cada nó (classe) pode possuir apenas um nó pai, enquanto em um GDA, um nó filho (classe) pode ter vários nós pais.

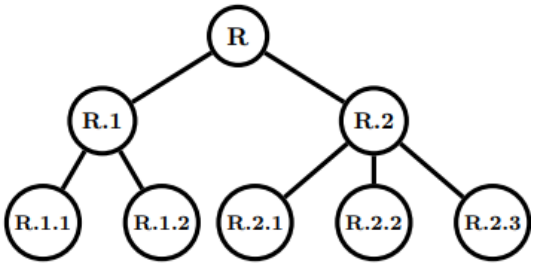

Figura 2.1: Hierarquia estruturada em Árvore

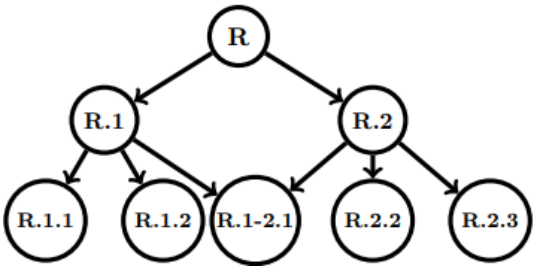

Figura 2.2: Hierarquia estruturada em DAG

O segundo aspecto está relacionado à profundidade da classificação na hierarquia de classes. Um método pode realizar predições obrigatórias de nós folha *Mandatory Leaf Node Prediction* (MLNP) ou predições não obrigatórias de nós folha *Non-Mandatory Leaf Node Prediction* (NMLNP)[23]. No caso do MLNP, a classe mais específica atribuída a uma instância deve ser uma das classes presentes em um nó folha na hierarquia de classes. Em contraste, no NMLNP, a classe atribuída não necessita ser necessariamente uma classe de nó folha, permitindo maior flexibilidade nas predições realizadas ao longo da hierarquia [24].

O terceiro aspecto refere-se ao número de diferentes caminhos de rótulos na hierarquia de classes, aos quais o método pode associar uma instância. Os métodos podem prever, para cada instância, apenas um único caminho de rótulos na hierarquia de classes SL, ou podem ser menos restritivos, prevendo múltiplos caminhos de rótulos, ML[18].

Por fim, o quarto aspecto diz respeito à forma como o método de classificação lida com a hierarquia de classes. Os métodos de classificação podem realizar a classificação de forma plana ou hierárquica (utilizando uma abordagem de modelo local ou global). Na classificação plana, o método ignora a hierarquia de classes e realiza previsões considerando apenas as classes associadas aos nós folha [23]. Na abordagem de modelo local, a hierarquia de classes é explorada a partir de uma perspectiva local, com a combinação

de classificadores que consideram, de maneira isolada, diferentes partes da hierarquia. De acordo com Silla [14], podemos categorizar as abordagens de modelo local conforme a forma como utilizam a informação local da estrutura hierárquica e como constroem seus classificadores em torno dela.

De modo geral, existem três principais abordagens locais:

- *Local Classifier per Node*: Nessas abordagens, um classificador é treinado para cada nó da hierarquia, onde cada classificador toma uma decisão binária sobre a pertinência ou não da instância naquele nó.
- *Local Classifier per Parent Node*: Um classificador é treinado para cada nó pai na hierarquia, com o objetivo de decidir qual de seus filhos deve ser selecionado.
- *Local Classifier per Level*: Um classificador é treinado para cada nível da hierarquia. Nesse caso, o classificador no nível correspondente decide qual classe no nível seguinte a instância pertence.

2.1.2 Global Model Naive Bayes

O algoritmo Naive Bayes (NB) é um método eficaz para aprendizado indutivo e mineração de dados [6]. No entanto, o NB não é adequado para problemas de classificação hierárquica. Para resolver essa limitação, Silla e Freitas [23] propuseram o Global Model Naive Bayes (GMNB), uma adaptação do algoritmo original [6] para lidar com hierarquias de classes.

Seja $D = \{d_1, \ldots, d_t\}$ um conjunto de instâncias de treinamento, $A = \{A_1, \ldots, A_n\}$ o conjunto de características preditivas de uma instância $d_j \in D$, e seja $C = \{c_1, \ldots, c_r\}$ um conjunto de classes que se relacionam através de uma estrutura hierárquica. Cada instância d_j é representada por seu vetor de atributos $Y_j = \{y_{1j}, y_{2j}, \ldots, y_{nj}\}$ com $y_{ij} \in A_i$ e associada a uma classe $c_i \in C$.

Dada uma nova instância $Y = \{y_1, y_2, \ldots, y_n\}$, onde $y_1, y_2, \ldots, y_n$ são os valores das características preditivas associadas a $A_1, A_2, \ldots, A_n$ respectivamente, o classificador NB plano simplesmente atribui a essa nova instância a classe c_i associada à máxima probabilidade posterior, calculada como $P(c_i \mid Y) \propto P(Y \mid c_i)P(c_i)$, onde $P(Y \mid c_i) = \prod_{j=1}^{n} P(y_j \mid c_i)$. Assim, a principal diferença do classificador GMNB está no cálculo das

probabilidades $P(c_i)$ e $P(y_j \mid c_i)$, pois ele as estima levando em consideração a hierarquia de classes.

Mais especificamente, o GMNB considera que qualquer instância da classe c_i também pertence a todas as suas classes parentais. Considerando o exemplo na Figura 2.1, se uma instância de treinamento pertence à classe $R.1.2$, então ela será contada nas frequências de todas as probabilidades envolvendo a classe $R.1.2$ ($P(R.1.2)$ e $P(y_j \mid R.1.2)$) e também em todas as probabilidades relacionadas à sua classe parental $R.1$ ($P(R.1)$ e $P(y_j \mid R.1)$). Essas adaptações permitem que o GMNB preveja classes em qualquer nível da hierarquia de classes.

Dado o contexto deste trabalho, o GMNB foi o classificador escolhido para o projeto, uma vez que sua capacidade de lidar com múltiplos níveis de hierarquia é condizente com os objetivos deste estudo.

2.2 Seleção de Atributos

A seleção de atributos é uma técnica cuja principal finalidade é identificar e escolher um subconjunto relevante de atributos que contribuem significativamente para a performance do modelo de aprendizado. O objetivo é melhorar a precisão do modelo, reduzir o tempo de treinamento e evitar o sobreajuste.

2.2.1 Métodos de Seleção de Atributos

Existem diversos métodos para a seleção de atributos, que podem ser agrupados em três categorias principais: métodos de filtro, métodos wrapper e métodos embutidos [18]. Neste artigo, focaremos particularmente nos métodos de filtro e wrapper.

Métodos de Filtro

Os métodos de filtro realizam a seleção de atributos de forma independente do modelo de aprendizado. Eles avaliam a relevância dos atributos com base em características estatísticas dos dados. Comumente utilizados métodos de filtro incluem a análise de correlação, a análise de variância e a seleção baseada em informações mutuas. A principal vantagem desses métodos é a sua simplicidade e eficiência computacional, uma vez que

não requerem a construção de modelos preditivos para a seleção de atributos.

Métodos Wrapper

Os métodos wrapper, por outro lado, utilizam um modelo de aprendizado como parte do processo de seleção de atributos. Esses métodos envolvem a construção e avaliação de modelos para diferentes subconjuntos de atributos e selecionam o subconjunto que resulta na melhor performance do modelo. O processo wrapper pode ser computacionalmente mais caro, mas tende a encontrar subconjuntos de atributos que são mais relevantes para o modelo específico em questão. Os métodos wrapper frequentemente empregam técnicas como busca sequencial, algoritmos genéticos e outros algoritmos de otimização para explorar o espaço de subconjuntos de atributos.

Foi adotado um método wrapper para a seleção de atributos, utilizando algoritmos genéticos. Os algoritmos genéticos são empregados para explorar o espaço de possíveis subconjuntos de atributos e identificar o subconjunto que proporciona o melhor desempenho.

2.3 Pré-Processamento de Dados

O pré-processamento de dados desempenha um papel fundamental. Trata-se de uma etapa crucial que visa preparar os dados brutos para análise, garantindo que eles estejam em formato e qualidade adequados.

- Limpeza de Dados: Isso envolve a identificação e tratamento de dados ausentes, duplicados, inconsistentes e a eliminação de registros que aparecem poucas vezes.
- Codificação de Atributos Categóricos: Se os atributos incluírem variáveis categóricas, elas precisarão ser codificadas em formato numérico.
- Transformação de Dados: É necessário aplicar transformações aos dados como a discretização caso os dados sejam decimais.
- Divisão de Dados: Envolve a separação do dataset em várias partes diferentes para posterior uso no classificador. Essa etapa é fundamental para treinar, ajustar parâmetros e avaliar o desempenho do modelo.

Um pré-processamento adequado dos dados contribui significativamente para a qualidade e confiabilidade dos resultados. A representação e a qualidade dos dados são, aspectos essenciais a serem considerados em qualquer análise [22].

2.4 Métricas de Avaliação

Nesta seção, abordamos as métricas utilizadas neste estudo para comparar os algoritmos propostos, bem como a métrica escolhida para avaliar a qualidade de um conjunto ou subconjunto de atributos.

2.4.1 Hierarchical F-measure

A métrica *Hierarchical F-Measure* (hF) é uma adaptação da tradicional medida F, amplamente utilizada em problemas de classificação plana, ajustada para considerar a hierarquia de classes. Conforme descrito por Silla e Freitas [13], a principal vantagem do uso da medida hF é sua aplicabilidade a qualquer cenário de classificação hierárquica, seja em árvore, DAG, ML, SL, MLNP ou NMLNP. A fórmula da hF é dada por:

$$hF(X) = \frac{2 \times hP(X) \times hR(X)}{hP(X) + hR(X)} \tag{2.1}$$

Neste estudo, a métrica hF foi utilizada para medir a qualidade dos dados, tanto com todos os atributos selecionados, quanto para os subconjuntos gerados pelo AG.

2.4.2 Teste t de Student

Os algoritmos propostos foram comparados com base nas seguintes hipóteses:

- **Hipóteses Consideradas**

  1. **Hipótese Nula (H_0)**: Não há diferença significativa entre as médias dos métodos.
  2. **Hipótese Alternativa (H_a)**: Existe uma diferença significativa entre as médias dos métodos.

- **Teste t de Student Pareado**

  O teste t de Student pareado foi aplicado para comparar as médias de dois grupos e verificar se existe uma diferença estatisticamente significativa entre elas [9]. O teste é calculado pela seguinte equação:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}} \tag{2.2}$$

  Onde $\bar{d}$ representa a média das diferenças entre os pares de observações, s_d é o desvio padrão dessas diferenças e n é o número de pares de observações. O valor de t é então comparado com os valores críticos da distribuição t de Student para determinar a significância estatística das diferenças observadas. O nível de significância utilizado foi $\boldsymbol{\alpha = 0.05}$.

Foi utilizado a biblioteca *scipy.stats*[1] para utilizar o Teste t de Student Pareado, conforme o pseudo-código abaixo. Para mais informações, conferir a Seção 5.3.

Algorithm 1 Análise estatística dos dados

```
import numpy as np
from scipy.stats import ttest_rel, levene
GAwNBDataset_i = []                    ▷ Valores de cada geração
GAwNNDataset_i = []                    ▷ Valores de cada geração

Teste t de Student pareado:
t_stat, t_p ← ttest_rel(GAwNB_avg, GAwNN_avg)
```

2.5 Ferramentas Aplicadas

É fundamental destacar que a escolha adequada das ferramentas e tecnologias desempenha um papel crucial. Essas escolhas otimizam a execução do projeto, promovendo maior versatilidade e aceleração ao longo das diferentes etapas. Essa abordagem facilita a aplicação de métodos específicos para a análise de dados e contribui para a obtenção de resultados de qualidade.

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.levene.html`

2.5.1 Linguagem de programação C/C++

A linguagem de programação *C*[2] foi desenvolvida por Dennis Ritchie entre 1969 e 1973 nos Laboratórios Bell, e publicada pela primeira vez em 1978. Segundo Kernighan [16], é uma linguagem de propósito geral, estruturada e de baixo nível, o que significa que oferece capacidades de controle próximas ao hardware, permitindo uma maior eficiência e desempenho.

A linguagem de programação *C++*[3] foi desenvolvida por Bjarne Stroustrup em 1983. E segundo o próprio [25], foi desenvolvida como uma extensão da linguagem C, com o objetivo de adicionar características de programação orientada a objetos. O C++ é conhecido por combinar as capacidades de baixo nível do C com as funcionalidades de alto nível da orientação a objetos, tornando-o uma linguagem poderosa e versátil.

2.5.2 Linguagem de programação Python

O *Python*[4] se destaca por sua sintaxe e extensa coleção de módulos internos prontos para uso e a flexibilidade de incorporar *frameworks* de terceiros, ampliando ainda mais suas capacidades [2].

2.5.2.1 cTypes

A biblioteca *ctypes*[5] é uma ferramenta em que permite criar e manipular tipos de dados compatíveis com C. Ela também possibilita chamar funções de bibliotecas dinâmicas (DLLs) ou bibliotecas compartilhadas, essa foi uma biblioteca fundamental na utilização do [13].

2.5.2.2 multiprocessing

O *multiprocessing*[6] é um módulo em Python que permite criar e gerenciar processos. Ele oferece suporte à criação de processos usando uma API semelhante ao módulo de *th-*

[2] `https://www.gnu.org/software/gnu-c-manual/`
[3] `https://gcc.gnu.org/onlinedocs/libstdc++/manual/index.html`
[4] `https://www.python.org/`
[5] `https://docs.python.org/3/library/ctypes.html`
[6] `https://docs.python.org/3/library/multiprocessing.html`

reading[7], porém *multiprocessing* permite a execução paralela de diferentes processos. Ele contorna o ***Global Interpreter Lock (GIL)*** usando subprocessos em vez de threads, o que permite aproveitar totalmente vários processadores em uma máquina.

2.5.2.3 TensorFlow

O *TensorFlow*[8] é uma biblioteca de código aberto para aprendizado de máquina e redes neurais. Ele oferece uma interface flexível para expressar algoritmos de aprendizado de máquina e implementá-los para execução. O TensorFlow é amplamente utilizado em pesquisa, produção e prototipagem rápida.

2.5.2.4 Keras

O *Keras*[9] é uma API de alto nível do TensorFlow para criar e treinar modelos de aprendizado profundo. É otimizado para casos de uso comuns e oferece uma interface simples e consistente. Os modelos Keras são modulares e compostos, permitindo a conexão de elementos configuráveis. Além disso, é fácil estender o Keras para desenvolver novas camadas, métricas e funções de perda.

2.5.2.5 Pandas

O *Pandas*[10], é uma biblioteca essencial para a manipulação e análise de dados em Python [19]. Seu recurso central é o DataFrame, sendo possível manipular dados em tabelas e oferecendo métodos para a seleção, filtragem e agregação de dados, facilitando a análise exploratória dos dados e a extração de informações relevantes.

2.5.2.6 Numpy

A biblioteca *Numpy*[11] do Python é uma ferramenta fundamental, que fornece uma ampla variedade de estruturas de dados e algoritmos para tarefas de análise de dados.

[7] `https://docs.python.org/3/library/threading.html`
[8] `https://www.tensorflow.org/about/bib?hl=pt-br`
[9] `https://www.tensorflow.org/guide/keras?hl=pt-br`
[10] `https://pandas.pydata.org/`
[11] `https://numpy.org/`

O *Numpy* oferece funções matemáticas padrão que permitem a realização eficiente de operações em conjuntos de dados completos. E facilidade para ultilização é uma de suas principais vantagens [19].

2.5.2.7 Scikit-learn

O *Scikit-learn*[12], também conhecido como sklearn, é uma biblioteca de aprendizado de máquina amplamente adotada na linguagem de programação Python, especialmente na área de ciência de dados. Essa biblioteca é altamente valorizada por suas implementações de diversos métodos de aprendizado de máquina e ferramentas para o tratamento de dados [11]

2.5.2.8 Visualização de Dados

A visualização de dados neste trabalho é realizada por meio das bibliotecas*Matplotlib*[13] e *Seaborn*[14].

O *Matplotlib* é amplamente reconhecido por sua capacidade de criar gráficos estáticos com um alto grau de personalização. Além disso, *Seaborn*, uma extensão do *Matplotlib*, simplifica a criação de gráficos mais complexos, especialmente para fins estatísticos, ao disponibilizar estilos estatísticos exclusivos.

2.6 Algoritmos Genéticos

Os AG são meta-heurísticas inspiradas na teoria da evolução de Darwin, inicialmente propostas por J. H. Holland em 1992. A premissa central é que indivíduos com características genéticas superiores têm maior probabilidade de sobreviver e gerar descendentes mais aptos, enquanto os menos aptos tendem a desaparecer ao longo das gerações.

No contexto dos AG, cada indivíduo é avaliado por uma função de aptidão específica ao problema em questão. Para a seleção de atributos, cada indivíduo é representado

[12] `https://scikit-learn.org/stable/`
[13] `https://matplotlib.org/`
[14] `https://seaborn.pydata.org/`

por uma sequência binária, onde cada bit indica a inclusão (1) ou exclusão (0) de um atributo no subconjunto candidato.

O algoritmo começa com uma população inicial aleatória de indivíduos e, iterativamente, seleciona os mais aptos até que um critério de parada seja atingido [12]. Em cada iteração, o AG gera uma nova população $s_1^{t+1}, s_2^{t+1}, \ldots, s_n^{t+1}$ a partir da população existente no tempo t, por meio de operações de reprodução, que envolvem seleção, recombinação e/ou mutação.

De acordo com Katoch [15], os principais operadores genéticos são o cruzamento e a mutação. O cruzamento combina informações genéticas de dois indivíduos para criar descendentes, essa etapa prioriza os indivíduos que apresentam maior aptidão, ou seja, aqueles que estão mais bem adaptados ao ambiente. Podendo ocorrer de diversas formas, como de ponto único, de múltiplos pontos ou uniforme. A Figura 2.3 ilustra a estrutura básica de um algoritmo genético.

A seleção é o processo que determina quais indivíduos de uma população irão contribuir para a geração seguinte. Essa etapa prioriza os indivíduos que apresentam maior aptidão, ou seja, aqueles que estão mais bem adaptados ao ambiente [12].

A recombinação, também conhecida como cruzamento, é o processo pelo qual dois indivíduos (ou pais) combinam suas características para gerar um ou mais descendentes. O objetivo é permitir que informações genéticas valiosas sejam compartilhadas entre indivíduos e passem para a próxima geração. O cruzamento ocorre com uma probabilidade maior, geralmente entre 80% e 95% [15].

A mutação é uma operação que introduz variação aleatória no código genético de um indivíduo. Ela é essencial para manter a diversidade dentro da população e evitar a estagnação da busca em ótimos locais. Geralmente varia entre 0,5% e 5% [15].

2.6.1 Redes Neurais Artificiais

As Redes Neurais Artificiais (Artificial Neural Networks - ANNs) são sistemas computacionais projetados para imitar o comportamento do cérebro humano. Inspiradas na estrutura e funcionamento das redes neurais biológicas, as ANNs são especialmente eficazes na tarefa de reconhecimento de padrões e na resolução de problemas complexos de aprendizado e classificação. Segundo Anders Krogh [17], as ANNs replicam, em uma escala reduzida, a interconexão dos neurônios, que são as células nervosas responsáveis

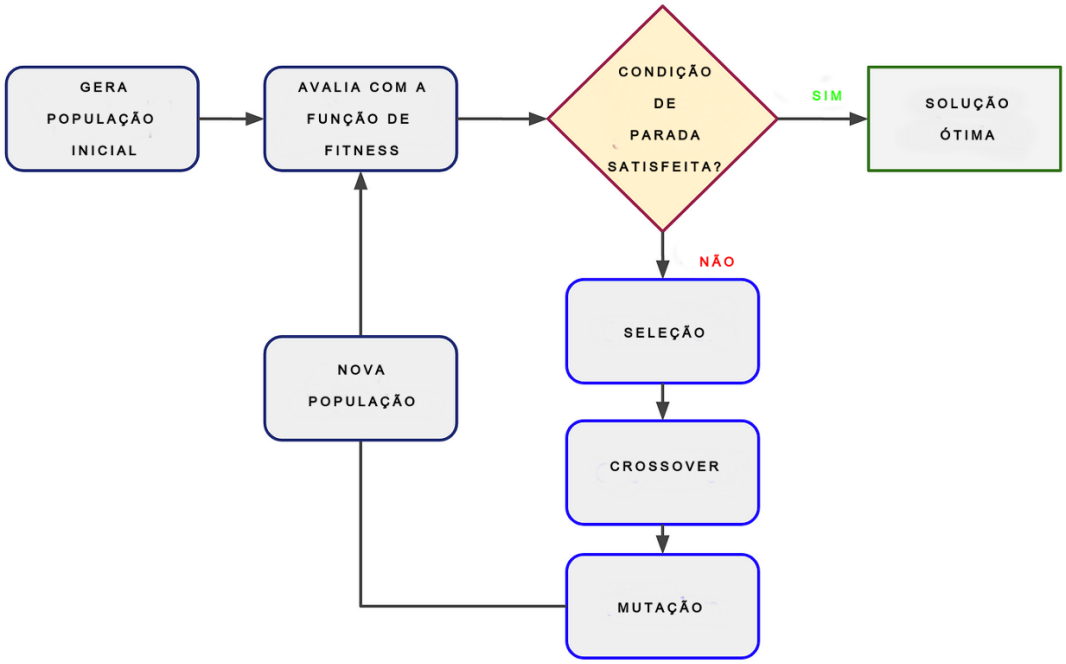

Figura 2.3: Estrutura Básica do AG.

pelo processamento de informações no cérebro humano.

Em uma ANN, cada neurônio artificial, que serve como uma unidade de processamento, recebe múltiplas entradas. Essas entradas são multiplicadas por pesos específicos que representam a força ou influência da entrada sobre o neurônio. Em seguida, uma função de ativação é aplicada ao resultado da soma ponderada dessas entradas, determinando assim a saída do neurônio. Se a soma ponderada exceder um determinado limiar, o neurônio "dispara", gerando uma saída que pode ser transmitida para os neurônios da camada seguinte.

A estrutura das redes neurais é tipicamente organizada em camadas distintas:

- **Camada de entrada:** Recebe os dados brutos do ambiente externo.
- **Camadas ocultas:** Uma ou mais camadas intermediárias, responsáveis por processar os dados através de múltiplas unidades interconectadas, permitindo que a rede capture complexidades intrínsecas dos dados.
- **Camada de saída:** Fornece a decisão ou predição final com base no processamento realizado pelas camadas anteriores.

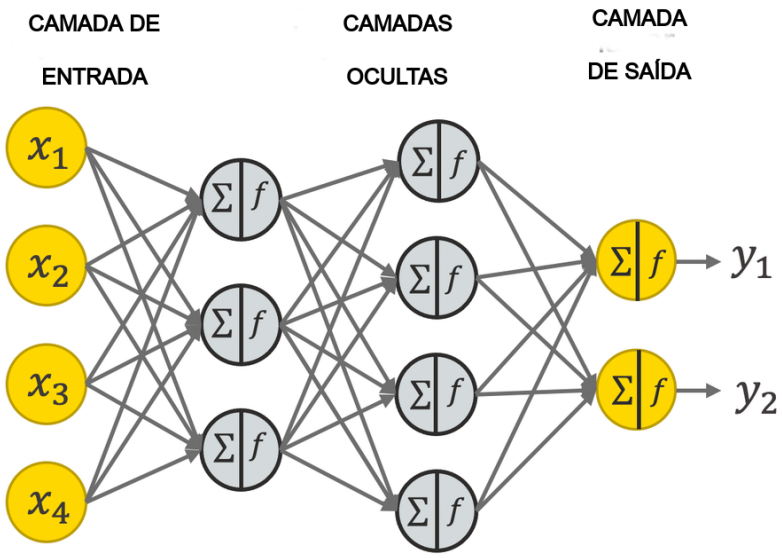

Figura 2.4: Estrutura básica de uma rede neural.

As ANNs que possuem múltiplas camadas ocultas são conhecidas como *perceptrons multicamadas* (Multilayer Perceptrons - MLPs). Essas redes são mais poderosas e capazes de resolver problemas que seriam intratáveis para redes de camada única, graças à sua capacidade de aprender representações hierárquicas dos dados.

Na Figura 2.4, é ilustrada a estrutura básica de uma rede neural artificial, mostrando a interconexão entre as diferentes camadas. As redes neurais podem ser vistas como equações paramétricas gigantescas, onde cada neurônio é governado pela seguinte equação:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.3}$$

Aqui, y representa a saída do neurônio, f é a função de ativação, x_i são as entradas, w_i os pesos associados a cada entrada, e b é o viés (bias) do neurônio. Também pode ser representada a relação entre dois neurônios como:

$$\boldsymbol{z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j} \tag{2.4}$$

onde,

- z_j é a entrada para o neurônio j.
- w_{ij} é o peso associado à conexão entre o neurônio i e o neurônio j.
- x_i é a entrada do neurônio i.
- b_j é o viés do neurônio j.

Com base nisso, projetou-se uma rede neural artificial composta por n neurônios na camada de entrada, $n-1$ na camada oculta e um único neurônio na camada de saída. A configuração segue a seguinte lógica:

- n corresponde ao número de atributos presentes no conjunto de dados. Assim, para um conjunto de dados $X = \{n_1, n_{1.1}, n_{1.2}, \ldots, n_m\}$, a camada de entrada terá m neurônios, onde cada neurônio representa um atributo.
- A utilização de apenas um neurônio na camada de saída deve-se ao fato de que a rede foi projetada como substituta do método *Global Model Naive Bayes* (GMNB). Portanto, o objetivo é prever um valor contínuo.

2.7 Trabalhos Relacionados

A presença de um grande número de atributos em um conjunto de dados hierárquicos pode representar um desafio [23]. Nesse cenário, a exploração de métodos eficientes para a redução de atributos e o aprimoramento de algoritmos preditivos tem ganhado destaque como uma estratégia significativa, especialmente quando aplicada a bases de dados com relações hierárquicas.

Silla [23], propôs um único modelo de classificação global considerando todas as classes na hierarquia, em vez de construir uma série de modelos de classificação locais como é

mais comum na classificação hierárquica. Os resultados alcançados são positivos e mostram que o modelo global proposto é melhor do que utilizar uma abordagem de modelo local. Este trabalho nos mostra que, é mais viável utilizar modelos de classificaodres globais em hierarquias.

Lima [18] conduziu uma investigação sobre a S.A aplicando o algoritmo *General Variable Neighborhood Search for Feature Selection in Hierarchical Classification* (GVNS-FSHC) em doze conjuntos de dados provenientes dos domínios de proteínas e imagens. O objetivo foi validar o impacto do GVNS-FSHC, em combinação com métodos de filtro, no desempenho da classificação hierárquica. Para tal, foram utilizados dois classificadores hierárquicos reconhecidos na literatura. Os resultados dos testes estatísticos indicaram que o GVNS-FSHC proporcionou desempenhos preditivos que foram consistentemente melhores ou, no mínimo, equivalentes aos obtidos ao utilizar todos os atributos. Além disso, o método demonstrou uma redução significativa no número de atributos necessários, destacando sua eficiência no contexto da classificação hierárquica. Este estudo não apenas validou a eficácia do GVNS-FSHC, mas também contribuiu para o avanço da metodologia de seleção de atributos, evidenciando a viabilidade e os benefícios da aplicação de meta-heurísticas nesse campo.

Bettin [1] realizou uma série de estudos que destacam a viabilidade da aplicação de métodos de Seleção de Atributos, como Seleção Univariable, Eliminação Recursiva de Características e Seleção baseada em Importância. Esses métodos mostraram-se eficazes ao serem aplicados em diversos conjuntos de dados, onde o desempenho dos algoritmos preditivos foi significativamente aprimorado após a seleção dos atributos. Ao considerar classificadores globais, uma vez que a eficiência dos algoritmos depende fortemente da relevância dos atributos selecionados, ao focar apenas nos mais importantes, não só otimiza o desempenho dos classificadores globais hierárquicos, como também simplifica a complexidade do modelo, tornando-o mais interpretável e eficiente.

Clare e King [4] apresenta uma investigação relevante sobre a classificação hierárquica de genes utilizando dados fenotípicos. No estudo, os autores desenvolvem técnicas de KDD para analisar dados experimentais de fenótipos, com o objetivo de prever a classe funcional dos genes mutantes abordando a construção de classificadores específicos para cada nível da hierarquia funcional dos genes e utilizando métodos heurísticos avançados para lidar com a escassez de dados e a fragmentação das classes. Destacando a importância de desenvolver métodos de aprendizado de máquina específicos para dados biológicos complexos.

Cerri [3] investigou técnicas de classificação hierárquica multirrótulo, comparando abordagens locais (Top-Down) e globais (One-Shot) no mesmo conjunto de dados biológicos utilizado neste trabalho. O estudo teve como principal objetivo avaliar o desempenho dessas técnicas em problemas de classificação hierárquica multirrótulo, focando na precisão e na capacidade de generalização em diferentes níveis da hierarquia. Os resultados indicaram que as abordagens hierárquicas obtiveram desempenho superior em níveis específicos da hierarquia, destacando a importância de métodos especializados para lidar com a complexidade inerente aos dados biológicos. Esse trabalho é particularmente relevante por demonstrar a necessidade de implementar e ajustar as técnicas para lidar com cada nível da hierarquia.

Capítulo 3

Desenvolvimento

Neste capítulo, detalharemos os principais passos adotados no desenvolvimento do projeto, com ênfase nos datasets utilizados, no pré-processamento realizado, e nos algoritmos aplicados. Serão abordados aspectos da seleção e transformação de dados, técnicas de validação, além da implementação e otimização dos algoritmos empregados.

3.1 Datasets

Os experimentos utilizam onze conjuntos de dados com classes organizadas hierarquicamente em uma estrutura de árvore, cobrindo dois domínios: proteína e imagem. O domínio de proteína é representado por nove conjuntos de dados bioinformáticos referentes ao genoma de levedura [4]. Os dois conjuntos de dados de imagem foram selecionados da competição ImageCLEF 2007 para anotação de imagens médicas de raios-X. A ImageCLEF visa fornecer um fórum de avaliação para a anotação multilíngue de imagens do domínio radiológico médico [5].

Esses conjuntos de dados estão inicialmente disponíveis como dados ML [18]. Como nosso método foca em lidar com o cenário de SL, realizamos uma etapa de pré-processamento para convertê-los em dados SL. A Tabela 3.1 mostra as características gerais dos conjuntos de dados. Para cada conjunto de dados, a segunda coluna corresponde ao domínio do conjunto de dados, enquanto a terceira coluna representa o número total de atributos. A quarta coluna representa o número de instâncias/objetos, e a quinta coluna representa o número de classes em cada nível da hierarquia da árvore.

Tabela 3.1: Características gerais dos conjuntos de dados

Conjunto de Dados	Domínio	N. Atributos	Instâncias	Classes/Nível
CellCycle	proteína	77	3723	15/14/14/8
Church	proteína	27	3720	15/14/14/7
Gasch2	proteína	52	3742	15/14/14/8
SPO	proteína	80	3653	15/13/14/7
Eisen	proteína	79	2359	12/14/13/7
Derisi	proteína	63	3677	15/13/14/7
Gasch1	proteína	173	3727	15/14/14/8
Sequence	proteína	478	3874	15/14/14/8
Expression	proteína	551	3742	15/14/14/8
ImageCLEF07A	imagem	80	11006	4/8/8
ImageCLEF07D	imagem	80	11006	8/7/11

3.2 Pré-processamento

O pré-processamento de dados é uma etapa fundamental em qualquer projeto de aprendizado de máquina ou heurística. Envolve a preparação e limpeza dos dados brutos para torná-los utilizáveis pelos algoritmos de modelagem. Visando corrigir inconsistências, lidar com valores ausentes e transformar os dados em um formato adequado para melhorar a performance e a precisão dos modelos.

Antes da seleção de atributos, o pré-processamento foi realizado em três etapas, que serão detalhadas nas próximas seções. A Figura 3.1 ilustra as etapas comumente utilizadas nesse processo.

3.2.1 Primeira Etapa: Mesclagem de Classes

Na primeira etapa, para garantir que cada classe tivesse um número suficiente de instâncias para treinamento e teste, todas as classes com menos de dez instâncias foram mescladas com suas respectivas classes pai. Este processo continuou até que todas as classes possuíssem pelo menos dez instâncias.

Figura 3.1: Etapas comuns de pré-processamento

3.2.2 Segunda Etapa: Discretização dos Atributos

Na segunda etapa, utilizamos o método de discretização não supervisionada *Equal Frequency Binning* [26]. Esse método divide os dados contínuos em intervalos de modo que cada intervalo contenha o mesmo número de instâncias, convertendo assim todos os atributos contínuos em valores discretos com até 20 partições. Quando não é possível realizar a discretização em 20 partições devido à natureza dos dados, o atributo é discretizado no menor número possível de intervalos. Essa transformação é essencial para a aplicação do algoritmo GMNB [13], descrito na Seção 2.1.2, pois o algoritmo foi projetado para operar exclusivamente com valores inteiros e uma única classe da hierarquia.

3.2.3 Terceira Etapa: Divisão dos Dados

Nesta etapa, utilizamos a técnica de validação cruzada com 5 partições *CV5*[1]. A validação cruzada é um método de avaliação de desempenho que divide o conjunto de dados em partes (ou *folds*). A aleatorização é empregada para mitigar quaisquer viéses estatísticos. Após isso, separamos cada arquivo em 5 partições estratificadas.

A Figura 3.2 mostra como a avaliação é feita utilizando esse método, sendo o dataset completo passando por 5 iterações com as respectivas partições de treino e teste.

O procedimento de CV5 ajuda a garantir que o modelo tenha um bom desempenho geral e não apenas em uma divisão específica dos dados. Através da validação cruzada,

[1] `https://scikit-learn.org/stable/modules/cross_validation.html`

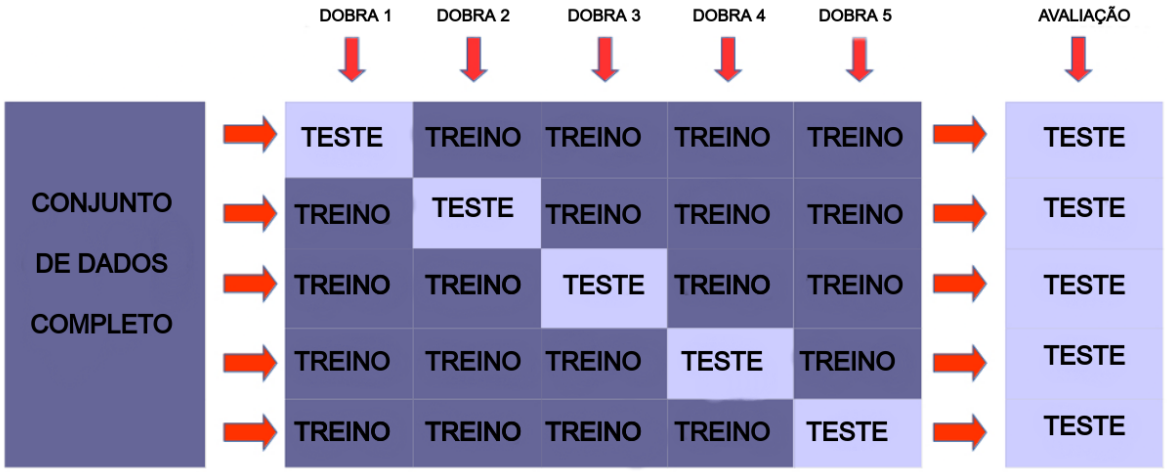

Figura 3.2: Validação cruzada com 5 partições (CV5)

podemos obter uma estimativa mais confiável do desempenho do modelo, pois ele é testado em diferentes subconjuntos dos dados, promovendo uma avaliação mais robusta e imparcial [18].

3.3 Método Proposto

Nesta seção, faremos uma breve exploração dos algoritmos utilizados neste trabalho e suas funcionalidades. Para detalhes específicos sobre a implementação ou código-fonte dos algoritmos consulte o Capítulo 5.3.

3.3.1 Operadores Genéticos

Nesta subseção, detalharemos os operadores genéticos utilizados no algoritmo genético implementado, explicando seu funcionamento e a importância de cada um no contexto da busca por subconjuntos de atributos.

Na literatura existem diversos métodos de seleção [15]. Neste trabalho, foi utilizado o método de seleção por torneio, seguindo pela lógica apresentada no pseudo-código abaixo. A seleção por torneio tema característica de não requerer conhecimento sobre a população, utilizando apenas uma relação de ordem para classificar K indivíduos. Esse método mostrou eficiência no trabalho de Paiva [21].

Algorithm 2 Seleção por Torneio

```
1: procedure TORNEIO(população, tamanho_torneio)
2:     selecionados ← lista vazia
3:     while |selecionados| < |população| do
4:         competidores ← subconjunto aleatório de população
5:         melhor ← competidor com melhor aptidão em competidores
6:         adicionar melhor a selecionados
7:     end while
8:     return selecionados
9: end procedure
```

A mutação swap é outra operação importante em GAs. Ela envolve a troca aleatória de posições entre dois genes em um cromossomo [15]. Embora simples, a mutação swap pode introduzir diversidade na população e ajudar a escapar de mínimos locais.

Algorithm 3 Mutação Swap

```
1: procedure MUTAÇÃOSWAP(indivíduo, taxa_mutação)
2:     for cadageneemindivíduo do
3:         if gerar_número_aleatório() < taxa_mutação then
4:             outro_índice ← escolher aleatoriamente outro índice diferente de gene
5:             trocar indivíduo[gene] com indivíduo[outro_índice]
6:         end if
7:     end for
8: end procedure
```

Existem vários métodos de crossover e mutação, Katoch [15] revisaram vários métodos de seleção, incluindo o PMX, em seu trabalho, demonstrando a eficiência desse método em preservar a disposição dos genes em um dos pais, permitindo variação nos genes.

3.3.2 Parâmetros utilizados no AG

Neste trabalho, a representação das soluções adota um cromossomo p na forma de um vetor com m posições: $p = (x_1, x_2, \ldots, x_m)$, onde cada componente x_i representa um gene binário (0 ou 1). Quando uma posição aleatória n no intervalo m contém $x_n = 1$, isso indica que o atributo correspondente foi selecionado; caso contrário, $x_n = 0$.

Dado as etapas observadas no Capítulo 2.6, temos que adotar alguns parâmetros para nosso algoritmo genético funcionar de forma adequada, neste trabalho foram adotados os seguintes parâmetros:

Algorithm 4 Crossover PMX Ingênuo

```
1: procedure PMX(pai1, pai2)
2:     ponto_corte ← escolher um ponto de corte aleatório ao tamanho dos pais
3:     filho1 ← cópia de pai1
4:     filho2 ← cópia de pai2
5:     for i ← 0 to ponto_corte do
6:         trocar filho1[i] com filho2[i]
7:     end for
8:     for i ← ponto_corte to tamanho do
9:         if filho1 não contém pai2[i] then
10:            troca1 ← pai2[i]
11:            troca2 ← filho1[i]
12:            pos1 ← índice de troca1 em filho1
13:            pos2 ← índice de troca2 em pai2
14:            trocar filho1[pos1] com filho1[i]
15:            trocar filho2[pos2] com filho2[i]
16:        end if
17:    end for
18:    return filho1, filho2
19: end procedure
```

Parâmetro	Valor
Tamanho da População	500
Número de Gerações	300
Taxa de Cruzamento	90%
Taxa de Mutação	10%

Tabela 3.2: Parâmetros do AG

Tamanho da População: O tamanho da população foi definido como 500 indivíduos. Um tamanho maior de população permite uma exploração mais abrangente do espaço de busca, o que pode levar a melhores soluções. No entanto, isso também aumenta o custo computacional, conforme discutido por Krogh [17]. Estudos prévios, como o de Helen [18], demonstraram que uma população de 500 é eficaz para explorar soluções de maneira adequada.

Número de Gerações: O número de gerações foi fixado em 300. Embora um maior número de gerações possa potencialmente levar a uma convergência melhor, ele também aumenta significativamente o tempo de execução. Em experimentos preliminares, testamos variações de 100, 200, 300 gerações e observamos que 300 gerações proporcionam

uma boa convergência dos *datasets*, especialmente com os operadores genéticos discutidos na Seção 3.3.1 [17].

Taxa de Cruzamento: A taxa de cruzamento foi definida como 90%. Taxas altas de cruzamento tendem a favorecer a exploração do espaço de busca, promovendo a diversidade genética entre os indivíduos, como abordado em trabalhos anteriores de Lima [18] e Paiva [12]. Esses estudos reportaram bons resultados com taxas de cruzamento similares.

Taxa de Mutação: A taxa de mutação foi estabelecida em 10%. Uma taxa de mutação baixa é importante para evitar mudanças excessivas nas soluções geradas, o que pode causar uma degradação nas populações. Entretanto, taxas baixas ainda permitem a exploração de novas soluções, conforme relatado em Krogh [17].

3.3.3 GAwNB

O *Genetic Algorithm with Global Model Naive Bayes* (GAwNB), ou Algoritmo Genético com Modelo Global Ingênuo de Bayes em português, é um algoritmo desenvolvido com base no classificador GMNB, conforme descrito na tese de Silla [13]. Este algoritmo utiliza a técnica de AG para otimização da população através do método de fitness.

Inicialmente, um vetor binário, por exemplo, $[1, 1, 0, 0, 1]$, é transformado em cinco arquivos diferentes no formato *.arff*[2] onde 1 representa que o atributo foi selecionado e 0 que o atributo não foi selecionado, permitindo a aplicação da técnica de validação cruzada CV5. O classificador é invocado para cada partição e, ao final, o *fitness* desse vetor seria é da seguinte forma:

$$\frac{\sum_{n=1}^{5} h_F(x_n y_n)}{5}$$

onde,

- n, é a partição atual;
- x representa a partição de teste;
- y representa a partição de treino.

[2] `https://waikato.github.io/weka-wiki/formats_and_processing/arff/`

3.3.3.1 Paralelismo

Ao empregar a técnica CV5, foi observado que o desempenho do algoritmo é notavelmente demorado, o que levantou a necessidade de explorar estratégias de paralelismo para mitigar esse problema. Dentre as abordagens consideradas, destacam-se:

1. **(Sequential):** Neste modo, o processamento ocorre de forma linear, utilizando apenas um núcleo da CPU e executando os arquivos de forma sequencial, seguindo o fluxo padrão de execução.

2. **(Threads):** Esta modalidade de envolve a criação de múltiplas *threads*[3] para o método de *fitness*. Este método, embora ofereça uma divisão de tarefas, ainda implica na criação e destruição repetitiva de threads e no compartilhamento da mesma memória, resultando em um ganho de desempenho limitado.

3. **(Paralelism):** Neste método, um conjunto de n subprocessos é criado, correspondendo ao limite máximo de núcleos da CPU. Aqui, a alocação e controle de recursos são gerenciados pelo próprio processador, permitindo que ele determine a próxima tarefa a ser executada.

Como vemos na Figura 3.3, o método de Paralelismo se destacou como o mais rápido em ambos os gráficos. O gráfico (% Faster) representa a porcentagem que o método é mais rápido em relação ao método *Sequential*.

Dado isso, o GAwNB, segue o Algoritmo 5 mostra como o algoritmo inicia com a criação da população inicial e, em cada geração, avalia a aptidão dos indivíduos usando o GMNB, validação cruzada para evitar viéses e paralelismo. Os melhores indivíduos são selecionados, e operadores genéticos são aplicados para gerar a próxima geração. O processo continua até que o número máximo de gerações seja atingido.

3.3.4 GAwNN

O *Genetic Algorithm with Neural Networks* (GAwNN), ou Algoritmo Genético com Redes Neurais em português, é um algoritmo desenvolvido com base em *Redes Neurais para Problemas de Regressão*[4].

[3] `https://eaulas.usp.br/portal/video.action?idItem=15431`

[4] `https://towardsdatascience.com/deep-neural-networks-for-regression-problems-81321897ca33`

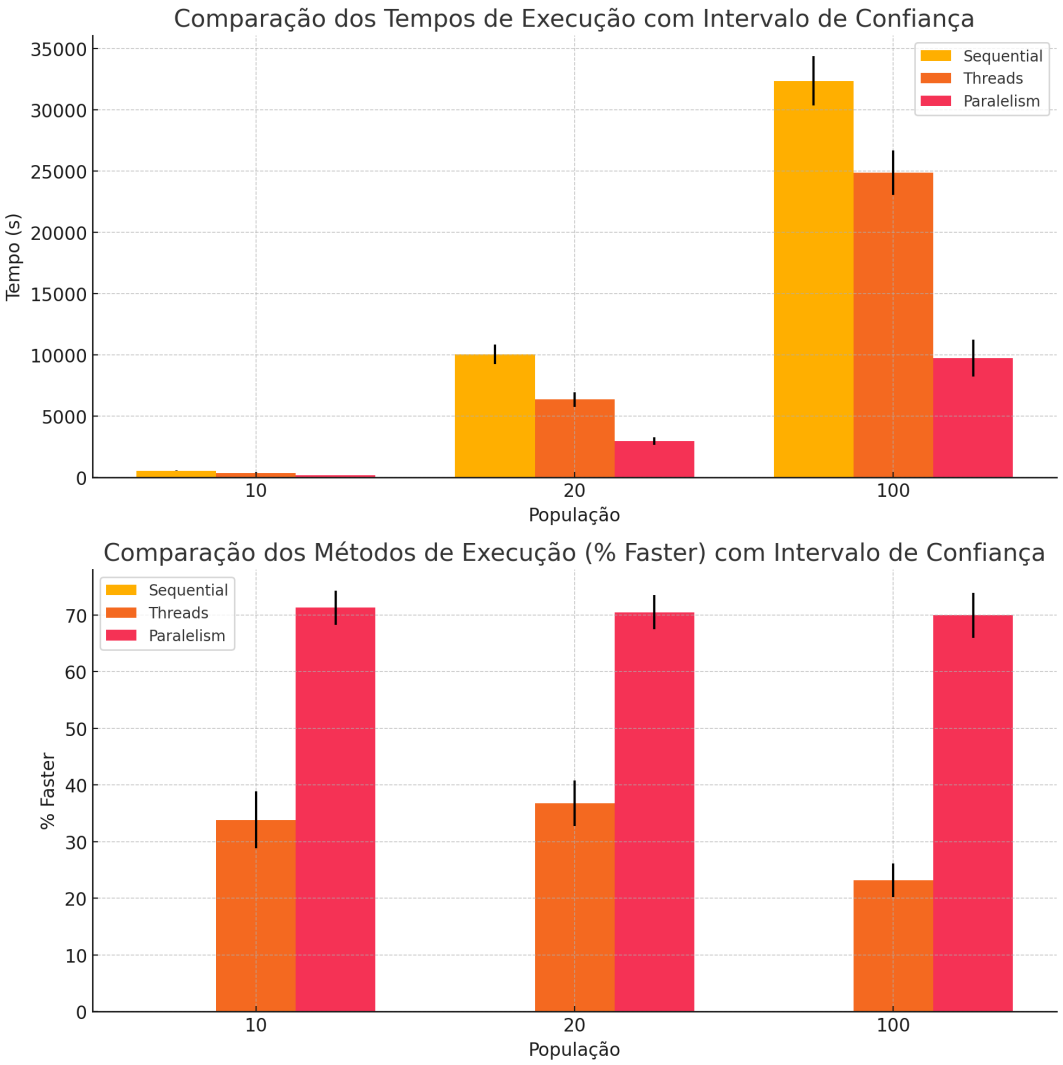

Figura 3.3: Comparação dos Métodos de Execução

Esse método visa empregar a utilização de redes neurais como algoritmo preditivo em substituição ao método GAwNB, devido à sua capacidade de resposta rápida.

3.3.4.1 Configuração de Parâmetros

Designamos um neurônio para cada atributo do nosso *dataset*. Ou seja, se nosso *dataset* tem n atributos, logo, haverá n neurônios na camada de entrada.

Devido à simplicidade da tarefa de regressão linear da rede neural, definimos apenas uma camada oculta. O número de neurônios na camada oculta é dado por:

Algorithm 5 Implementação do GAwNB

```
1: procedure RUN                                          ▷ Inicia a população
2:     population_list, generation ← create_population(size, attributes)
3:     while generation < num_generations do              ▷ Operadores Genéticos
4:         population_fitness ← parallel_CV5.evaluate_population(population_list)
5:         get_best_fitness(population_fitness, population_list, generation)
6:         population_list ← tournament(population_list, population_fitness)
7:         population_list ← pmx_crossover(population_list, crossover_rate)
8:         population_list ← swap_mutation(population_list, mutation_rate)
9:         generation ← generation + 1
10:    end while
11: end procedure
```

$$N_o = \frac{N_e + N_s}{2}$$

Onde:

- N_o é o número de neurônios da camada oculta;
- N_e é o número de neurônios da camada de entrada;
- N_s é o número de neurônios da camada de saída.

Como a nossa rede neural realiza apenas uma tarefa de regressão, ou seja, retorna um valor real dentro de um intervalo, ela deve ter no máximo um neurônio na camada de saída, sendo este o valor estimado pela rede neural, e não retornar probabilidades.

Portanto, nossa equação se simplifica para:

$$N_o = \frac{N_e + 1}{2}$$

Devido à natureza do problema, as funções de ativação precisaram ser diferentes em cada camada da rede. Isso se deve ao fato de que, ao mesmo tempo que buscamos encontrar padrões lineares entre os atributos, também queremos evitar atributos com pesos negativos na rede.

1. **Camada de Entrada/Camada de Saída**

   Utilizamos a ativação linear, dada por:

$$f(x) = x \tag{3.1}$$

2. **Camada Oculta**

   Utilizamos a ativação ReLU *(Rectified Linear Unit)*[5], dada por:

$$f(x) = \max(0, x) \tag{3.2}$$

Dado isso, o GAwNN, segue o seguinte pseudo-código:

Algorithm 6 Implementação do GAwNN

```
1: procedure RUN
2:     create_training_database()          ▷ Treina e inicia a Rede Neural Artificial
3:     train_model()
4:     get_model()
                                                              ▷ Inicia a população
5:     population_list, generation ← create_population(size, attributes)
6:     while generation < num_generations do                  ▷ Operadores Genéticos
7:         population_fitness ← neural_network.evaluate_population(population_list)
8:         get_best_fitness(population_fitness, population_list, generation)
9:         population_list ← tournament(population_list, population_fitness)
10:        population_list ← pmx_crossover(population_list, crossover_rate)
11:        population_list ← swap_mutation(population_list, mutation_rate)
12:        generation ← generation + 1
13:    end while
14: end procedure
```

O algoritmo começa inicializando o banco de dados de treinamento. Este processo envolve a preparação e o treinamento da NN utilizando indivíduos aleatórios, que são avaliados com o GMNB e armazenados em tuplas contendo um vetor binário correspondente aos atributos selecionados e o valor associado a esse vetor. Após a criação do banco de dados, o modelo da NN é treinado com esses dados, e o modelo treinado é utilizado para a avaliação da população.

Na sequência, o algoritmo gera uma população inicial de soluções, definindo o número de indivíduos e começando com a geração inicializada em zero. O processo de evolução da população prossegue iterativamente até que o número de gerações atinja o limite máximo predefinido. Em cada iteração, a aptidão de cada indivíduo é avaliada com a NN treinada. Além disso, operadores genéticos, como seleção, cruzamento e mutação, são aplicados para evoluir a população, visando aprimorar continuamente a qualidade das soluções encontradas.

[5] `https://machinelearningmastery.com/rectified-linear-activation`

Capítulo 4

Resultados

Os algoritmos GAwNB, discutido no Capítulo 3.3.3, e GAwNN, abordado no Capítulo 3.3.4, foram implementados em Python, C e C++, conforme descrito no Capítulo 2.5. Os experimentos foram realizados em um computador com as seguintes especificações:

Intel Core i5-11400 de 11ª Geração, 2.6 GHz (4.4GHz Turbo), 12MB de Cache, 6 Núcleos, 8GB de RAM, rodando Ubuntu 22.04.4 LTS, Linux 6.5.0-35-generic, gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0, e Python 3.10.12.

O GAwNB e GAwNN foram otimizados utilizando paralelismo, conforme mostrado sua eficiência no gráfico 3.3. Os algoritmos propostos foram projetados para atuar como uma etapa de pré-processamento para classificadores hierárquicos. Foi utilizado o classificador hierárquico GMNB, referenciado no Capítulo 2.1.2, para avaliar a qualidade dos atributos, baseando na métrica hF discutida na Seção 3.3.3, conjunto ao CV5, nos *datasets* referenciados no Capítulo 3.1.

As Tabela, são apresentadas da seguinte forma:

- **GAwNB**: Média*(avg)* ± Desvio padrão*(dp)*. Algoritmo proposto na Seção 3.3.3
- **GAwNN**: Média (avg) ± Desvio padrão (dp). Algoritmo proposto na Seção 3.3.4
- **ALL**: Medimos o desempenho do classificador sem qualquer etapa de seleção de recursos no pré-processamento, ou seja, utilizando todos os recursos disponíveis no conjunto de dados.

Os resultados em negrito indicam desempenhos que, comprovadamente, são estatisticamente superiores, de acordo com as métricas discutidas na Seção 2.4.2.

4.1 Análise dos Algoritmos

Dataset	ALL	GAwNB (avg ± dp)	GAwNN (avg ± dp)
Cellcycle	26.11	30.93 ± 0.57	29.87+ ± 0.83
Church	14.28	26.53 ± 0.02	26.53+ ± 0.01
Derisi	11.62	13.39 ± 0.18	13.23+ ± 0.23
Eisen	20.82	**24.29 ± 0.26**	23.14+ ± 0.32
Expression	43.91	49.53 ± 1.34	47.81+ ± 1.79
Gasch 1	22.27	**31.76 ± 0.66**	29.74+ ± 0.76
Gasch 2	19.35	22.91 ± 0.31	23.01+ ± 0.34
Sequence	19.24	20.54 ± 1.46	20.95+ ± 1.93
ImgCLEF07A	80.37	81.49 ± 0.09	81.40+ ± 1.12
ImgCLEF07D	63.04	65.77 ± 0.64	66.35+ ± 1.54
SPO	13.99	**21.65 ± 0.17**	19.74+ ± 0.28

Tabela 4.1: Comparação dos Resultados Computacionais hF

4.1.1 Análises Estatísticas

Nas Tabelas 4.1 e 4.2, os valores destacados em negrito indicam resultados onde o teste estatístico de Student (*Teste t*) demonstrou que $Valor_P < \alpha$. O $Valor_P$ de cada conjunto de dados pode ser observado nas Tabelas 4.3, 4.4.

Este resultado ($p < \alpha$) oferece evidência significativa para rejeitar a hipótese nula (H_0) [9], sugerindo que as médias dos métodos diferem entre os algoritmos considerando o dataset em questão.

Portanto, utilizando a métrica hF o algoritmo GAwNB demonstra desempenho superior ao GAwNN nos datasets Eisen, Gasch 1 e SPO.

Dataset	ALL	GAwNB (avg ± dp)	GAwNN (avg ± dp)
Cellcycle	77	30.4 ± 2.17	31.0 ± 1.34
Church	27	11.2 ± 4.42	8.1 ± 1.21
Derisi	63	33.6 ± 1.38	34.2 ± 0.43
Eisen	79	**41.2 ± 3.16**	51.4 ± 1.45
Expression	551	**257.6 ± 13.59**	359.4 ± 4.38
Gasch 1	173	**61.4 ± 8.21**	94.6 ± 2.56
Gasch 2	52	21.9 ± 1.43	21.2 ± 1.34
Sequence	478	**238.2 ± 6.74**	317.4 ± 3.34
ImgCLEF07A	80	67.1 ± 4.29	71.6 ± 1.34
ImgCLEF07D	80	**36.3 ± 4.54**	48.2 ± 1.27
SPO	79	39.9 ± 1.13	39.1 ± 0.7

Tabela 4.2: Comparação da Quantidade de Atributos Selecionados

Dataset	$Valor_P$
Cellcycle	0.158
Church	0.164
Derisi	0.292
Eisen	**0.001**
Expression	0.052
Gasch 1	**0.008**
Gasch 2	0.499
Sequence	0.363
ImgCLEF07A	0.436
ImgCLEF07D	0.738
SPO	**0.001**

Tabela 4.3: Teste de hipótese da Tabela 4.1

Dataset	$Valor_P$
Cellcycle	0.067
Church	0.465
Derisi	0.320
Eisen	**0.009**
Expression	**0.002**
Gasch 1	**0.001**
Gasch 2	0.673
Sequence	**0.001**
ImgCLEF07A	0.076
ImgCLEF07D	**0.002**
SPO	0.762

Tabela 4.4: Teste de hipótese da Tabela 4.2

$\boldsymbol{\alpha = 0.05}$

Considerando a S.A, o algoritmo GAwNB demonstra desempenho superior ao GAwNN nos datasets Eisen, Expression, Gasch1, Sequence e ImgCLEF07D.

Em contrapartida, em outros conjuntos de dados, ambos os algoritmos exibem desempenho equivalente, conforme indicado pela análise estatística realizada.

4.1.2 Análise de Custo Computacional

A Figura 4.1 ilustra a comparação do tempo computacional entre os dois algoritmos estudados.

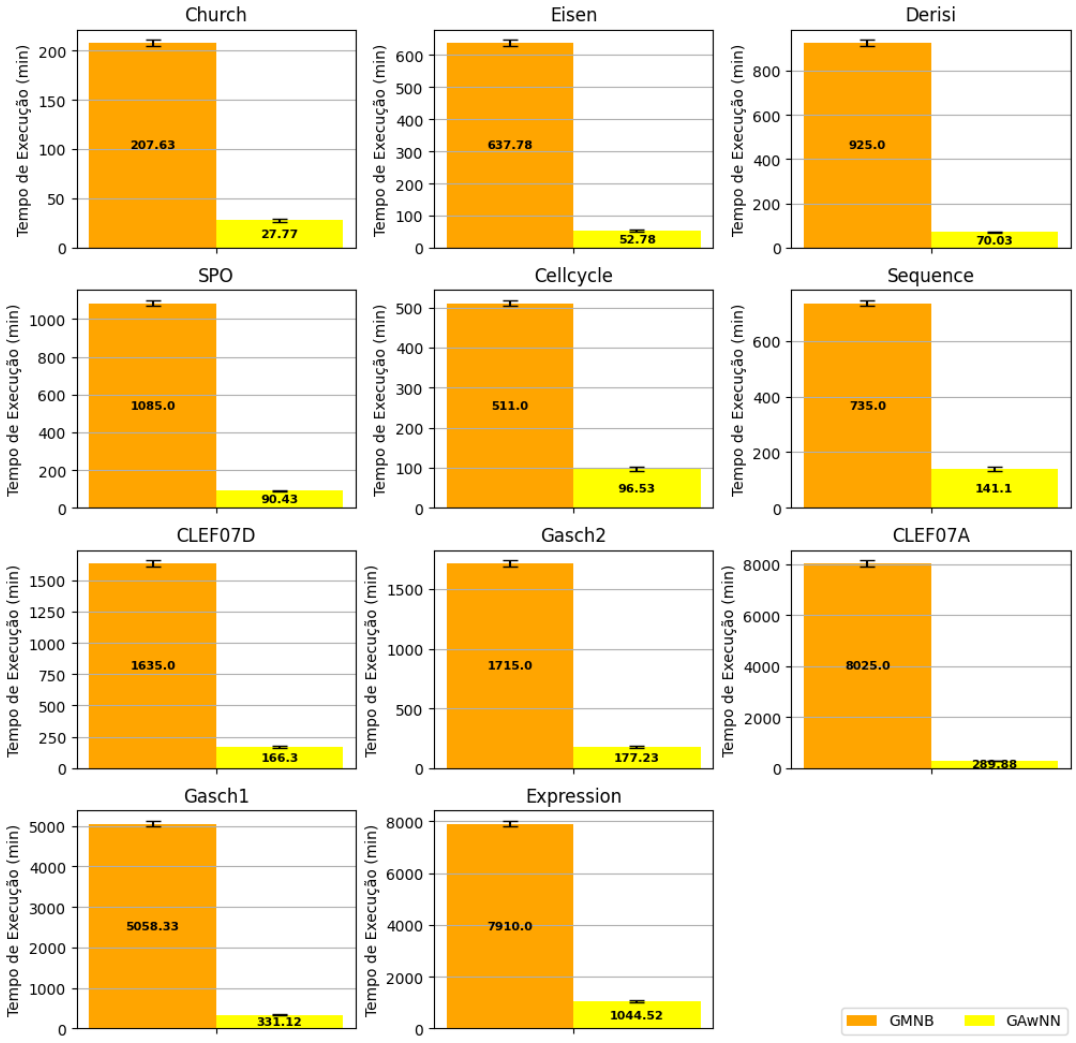

Figura 4.1: Comparação de Custo Computacional

Os resultados obtidos foram submetidos ao Teste t de Student, resultando em um $Valor_p = 0.0171$. Com $p < \alpha$, há evidência estatística significativa para rejeitar a hipótese nula (H_0).

Embora o método GAwNB tenha se mostrado superior ou equivalente ao GAwNN na métrica hF em três conjuntos de dados e na S.A em cinco, em outros cenários ambos os métodos apresentaram desempenho equivalente. No entanto, ao considerar o custo computacional, os resultados indicam, com evidência estatística, que o método GAwNN

supera consistentemente o GAwNB em todos os conjuntos de dados. Em outras palavras, o GAwNN se destaca por ser um algoritmo menos custoso em termos computacionais e com resultados similares.

4.2 Comparação com o Estado da Arte

Utilizou-se o estudo de Lima[18], referenciado como estudo relacionado na seção 2.7. A Tabela 4.5 mostra a comparação entre as médias de cinco execuções entre os dois algoritmos construídos e o algoritmo **(GVNS-FSHC)**.

Dataset	GAwNN (avg ± dp)	**GVNS** (avg ± dp)
Cellcycle	29.87 ± 0.83	25.27 ± 2.4
Church	26.53 ± 0.01	22.28 ± 4.5
Derisi	13.23 ± 0.23	**16.42 ± 1.1**
Eisen	23.14 ± 0.32	21.84 ± 1.9
Expression	47.81 ± 1.79	48.12 ± 2.4
Gasch 1	29.74 ± 0.76	28.04 ± 2.0
Gasch 2	**23.01 ± 0.34**	18.71 ± 1.3
Sequence	**20.95 ± 1.93**	19.46 ± 0.6
ImgCLEF07A	81.40 ± 1.12	80.67 ± 0.8
ImgCLEF07D	66.35 ± 1.54	**67.78 ± 0.7**
SPO	19.74 ± 0.28	20.42 ± 2.4

Tabela 4.5: Compação do GVNS-FSHC e GAwNB

O Teste t de Student, foi empregado para as análises, conforme utilizado na seção anterior. A Tabela 4.6 indicam, estatisticamente, qual método apresentou melhor desempenho para o conjunto de dados em questão.

Para conjuntos de dados que não estão inseridos, apresentam que não há evidência estatística suficiente para inferir que um é melhor que outro.

Quando comparados ao estado da arte, o algoritmo GAwNN foi desenvolvido com foco em eficiência, incorporando técnicas de otimização e paralelismo para maximizar o desempenho computacional. Ao compará-lo com o estado da arte, representado pelo

Dataset	$\boldsymbol{\alpha = 0.05}$
Cellcycle	GAwNB
Derisi	GVNS
Eisen	GAwNB
Gasch 1	GAwNB

Tabela 4.6: Melhores Métodos para Cada Conjunto

GVNS, nossos resultados mostram que o GAwNN apresentou desempenho superior em 2 conjuntos de dados, evidenciando sua capacidade de competir diretamente com o método da literatura.

Além disso, foi projetado para explorar paralelismo, o que permitiu uma execução mais rápida e eficiente, especialmente em grandes volumes de dados. Nos poucos casos em que o desempenho foi inferior (apenas 2 conjuntos), a diferença não foi estatisticamente significativa, e em todos os demais cenários o GAwNN mostrou-se equivalentes ou superior ao GVNS, comprovando sua versatilidade.

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Conclusões

Ambos os algoritmos propostos foram eficazes na redução do número de atributos em todos os conjuntos de dados, alcançando resultados superiores aos obtidos com o conjunto completo de atributos.

De modo geral, o GAwNN demonstrou desempenho superior ao GAwNB, destacando-se por sua maior velocidade e por alcançar resultados comparáveis aos do GAwNB. Além disso, ao ser comparado com estudos da literatura, o GAwNN mostrou-se competitivo em relação ao método GVNS, reforçando sua eficiência e aplicabilidade.

5.2 Trabalhos Futuros

Com base no trabalho desenvolvido e nos resultados alcançados, surgem diversas oportunidades para pesquisas adicionais que possam aprimorar e complementar o nosso entendimento, incluindo a avaliação e exploração de novos métodos que envolvam redes neurais, com um neurônio em cada nível da hierarquia, indo além de sua utilização apenas como um neurônio em cada atributo. Além disso, é pertinente considerar a realização de estudos comparativos utilizando diferentes contextos e com diferentes classificadores, além disso, outras métricas de desempenho, além do hF, podem ser relevantes para avaliar o desempenho do modelo.

5.3 Código Fonte

Todo os algoritmos aqui apresentados, estão disponíveis no repositório GitHub do projeto, acessível através do link aqui abaixo. Seguindo os princípios da licença *GPL3*[1], os algoritmos estão disponíveis para uso, modificação e distribuição livremente, desde que os direitos autorais e o código-fonte sejam preservados.

https://github.com/iyksh/GAFS-HC

5.4 Agradecimentos

Gostaria de expressar minha gratidão ao Programa de Iniciação à Pesquisa (PIP) pelo apoio financeiro para a realização deste trabalho. Agradeço também à Universidade Federal de Ouro Preto (UFOP) e ao ao Discovery Data Lab pelo aprendizad, infraestrutura necessária e pelas oportunidades oferecidas.

Agradeço à Helen, por sua orientação, incentivo e pela confiança. Por fim, não posso deixar de agradecer à minha família, cujo apoio incondicional foi fundamental para o desenvolvimento do interesse acadêmico e conclusão deste projeto.

[1] `https://www.gnu.org/licenses/gpl-3.0.pt-br.html`

Referências Bibliográficas

[1] Bettin, A. S. e da Silva, F. M.: 2023, Estudos da selecao de atributos em modelos de previsao: Uma analise comportamental, Online. Não paginado.
URL: *https://revistaft.com.br/estudos-da-selecao-de-atributos-em-modelos-de-previsao-uma-analise-comportamental/*

[2] Borges, L. E.: 2014, *Python para desenvolvedores*, 1 edn, Novatech.

[3] Cerri, R.: 2010, *Técnicas de classificação hierárquica multirrótulo*, Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (USP), São Carlos.

[4] Clare, A. e King, R. D.: 2001, *Knowledge discovery in multi-label phenotype data*, Phd thesis, University of Wales.

[5] Dimitrovski, I., Kocev, D., Loskovska, S. e Džeroski, S.: 2011, Hierarchical annotation of medical images, *Pattern Recognition* **44**(10–11), 2436–2449.

[6] Duda, R. e Hart, P.: 1973, *Pattern classification and scene analysis*, A Wiley-Interscience, New York.

[7] Figueiredo, C. X. e Paes, O. N.: 2004, Descoberta de conhecimento em banco de dados: Um estudo de caso da pesquisa científica na universidade federal de lavras, *Article* p. 8.

[8] Galvão, N. D. e de Fátima Marin, H.: 2009, Data mining: a literature review, *Article* p. 5.

[9] Gaspar, J. d. S., Reis, Z. S. N., Oliveira, I. J. R., Silva, A. P. C. e Dias, C. d. S.: 2023, *Introdução à análise de dados em saúde com Python*, CI-IA Saúde-UFMG, Belo Horizonte.

[10] Han, J. e Kamber, M.: 2011, Data mining: Concepts and techniques, *Article* **3**, 744.

[11] Hao, J. e Ho, T. K.: 2019, Machine learning made easy: A review of scikit-learn package in python programming language, *Journal of Educational and Behavioral Statistics* **44**(3), 348–361.

[12] Izidoro, S. C., de Melo-Minardi, R. C. e Pappa, G. L.: 2014, Gass: identifying enzyme active sites with genetic algorithms, *Bioinformatics* **31**(6), 864–870.

[13] Jr., C. N. S. e Freitas, A. A.: 2009, A global-model naive bayes approach to the hierarchical prediction of protein functions, *Article* p. 6.

[14] Jr., C. N. S. e Freitas, A. A.: 2011, A survey of hierarchical classification across different application domains, *Springer US* **1–2**, 31–72.

[15] Katoch, S., Chauhan, S. S. e Kumar, V.: 2020, A review on genetic algorithm: past, present, and future, *Multimedia Tools and Applications* **79**(21–22), 397–402.

[16] Kernighan, B. W. e Ritchie, D. M.: 1988, *The C Programming Language*, 2nd edn, Prentice Hall, Englewood Cliffs, NJ.

[17] Krogh, A.: 2008, What are artificial neural networks?, *Nature Biotechnology* **26**(2), 195–197.

[18] LIMA, H. C. S. C. et al.: 2021, *Hybrid feature selection approaches using metaheuristics for hierarchical classification*, Master's thesis.

[19] McKinney, W.: 2012, *Python for Data Analysis*, O'Reilly Media, Inc.

[20] Muir, P. e Li, S.: 2016, The real cost of sequencing: scaling computation to keep pace with data generation, *Article* p. 9.

[21] Paiva, V. d. A.: 2021, *Gass-metal: um servidor web para identificação de sítios metálicos similares em proteínas baseado em algoritmos genéticos paralelos*, Dissertação de mestrado, Universidade Federal de Itajubá, Itajubá, Brasil. `https://repositorio.unifei.edu.br/jspui/handle/123456789/2318`.

[22] Pyle, D.: 1999, *Data Preparation for Data Mining*, Morgan Kaufmann Publishers, Los Altos, California.

[23] Silla, C. N. e Freitas, A. A.: 2009, A global-model naive bayes approach to the hierarchical prediction of protein functions, *Proceedings of the Ninth IEEE International Conference on Data Mining (ICDM-2009)*, IEEE Press, Miami Beach, pp. 182–196.

[24] Silva, L. e., Peres, S. e Boscarioli, C.: 2016, *Introdução à Mineração de Dados com Aplicações em R*, Elsevier, Rio de Janeiro.

[25] Stroustrup, B.: 2013, *The C++ Programming Language*, 4th edn, Addison-Wesley, Boston, MA.

[26] Yang, Y. e Webb, G. I.: 2001, Proportional k-interval discretization for naive-bayes classifiers, *12th European Conference on Machine Learning*, Vol. 2167 of *LNCS*, Springer, Berlin, pp. 564–575.