UNIVERSIDADE FEDERAL DO ABC
CMCC - CENTRO DE MATEMÁTICA, COMPUTAÇÃO E COGNIÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME SEIDYO IMAI ALDEIA

ALGORITMOS EVOLUTIVOS PARA REGRESSÃO SIMBÓLICA UTILIZANDO A ESTRUTURA INTERAÇÃO-TRANSFORMAÇÃO

PROJETO DE GRADUAÇÃO EM COMPUTAÇÃO

SANTO ANDRÉ
2019

GUILHERME SEIDYO IMAI ALDEIA

ALGORITMOS EVOLUTIVOS PARA REGRESSÃO SIMBÓLICA UTILIZANDO A ESTRUTURA INTERAÇÃO-TRANSFORMAÇÃO

Projeto de Graduação em Computação apresentado ao Bacharelado em Ciência da Computação da Universidade Federal do ABC, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Fabrício Olivetti de França
Universidade Federal do ABC

SANTO ANDRÉ
2019

Dedico este trabalho à minha família.

AGRADECIMENTOS

À minha família, pelo incentivo, apoio, compreensão e amor incondicional. Por terem me fortalecido e feito de mim quem sou hoje.

Ao meu orientador Fabrício, pela oportunidade e por toda a atenção, confiança e suporte, despertando e fomentando minha paixão pelos estudos e pela pesquisa.

A esta universidade e cada um dos professores que cruzaram meu caminho ajudando a construir meus conhecimentos.

A todos que fizeram parte desta minha jornada, direta ou indiretamente, o meu muito obrigado.

Quando bem feita, a programação é uma arte sutil, profunda e altamente qualificada, embasada em diversas habilidades técnicas. (STROUSTRUP, Bjarne, 1997).

RESUMO

ALDEIA, G. S. I. Algoritmos evolutivos para Regressão Simbólica utilizando a estrutura Interação-Transformação. 2019. 98 f. Projeto de Graduação em Computação – Bacharelado em Ciência da Computação, Universidade Federal do ABC. Santo André, 2019.

Dada uma base de dados com variáveis explanatórias $\mathbf{x}$ e uma variável alvo y, onde exista uma função desconhecida $f(\mathbf{x}) = y$, a regressão simbólica busca por uma função $\hat{f}(\mathbf{x})$ que se aproxime suficientemente bem de $f(\mathbf{x})$. Ela costuma ser feita por meios da programação genética, um método de busca e otimização que simula conceitos da teoria da evolução para construir e ajustar $\hat{f}(\mathbf{x})$ através da manipulação de populações de soluções, mas a forma em que as soluções são expressas (por meio de árvores sintáticas) apresenta um espaço de busca amplo, soluções com um comportamento qualitativo instável para pequenas modificações em sua estrutura e resultados pouco interpretáveis. Nesse contexto foi introduzida recentemente uma nova representação para a regressão simbólica - a Interação-Transformação - de forma a evitar os problemas da representação por árvores. Esse trabalho visa aplicar e avaliar o desempenho dessa nova estrutura em algoritmos evolutivos como uma alternativa à representação por árvores, com o objetivo de retornar resultados competitivos e com maior interpretabilidade.

Palavras-chave: Computação evolutiva. Algoritmos evolutivos. Interação-transformação. Regressão simbólica.

ABSTRACT

ALDEIA, G. S. I. Evolutionary algorithms for Symbolic Regression using the Interaction-Transformation structure. 2019. 98 f. Projeto de Graduação em Computação – Bacharelado em Ciência da Computação, Universidade Federal do ABC. Santo André, 2019.

Given a dataset with explanatory variables $\mathbf{x}$ and a target variable y, where there is an unknown function $f(\mathbf{x}) = y$, the symbolic regression searchs for a function $\hat{f}(\mathbf{x})$ that fits approximately well $f(\mathbf{x})$. It is usually done by means of genetic programming, a search and optimization method that simulates concepts from the theory of evolution to build and adjust $\hat{f}(\mathbf{x})$ through the manipulation of populations of solutions, but the way the solutions are expressed (through syntactic trees) presents a wide search space, solutions with an unstable qualitative behavior for small changes in its structure, and results with poor interpretability. In this context, it was been introduced a new representation for symbolic regression - called Interaction-Transformation - in order to avoid the problems of representation by trees. This work aims to apply and evaluate the performance of this new structure in evolutionary algorithms as an alternative to representation by trees, with the objective of returning competitive results with greater interpretability.

Keywords: Evolutionary computing. Evolutionary algorithms. Interaction-transformation. Symbolic regression.

LISTA DE FIGURAS

Figura 1 – Exemplo de regressão linear. 9
Figura 2 – Exemplo de regressão polinomial com diferentes graus. 9
Figura 3 – Exemplo de regressão KNN. 11
Figura 4 – Composição de uma função por meio de uma árvore. Ao lado de cada nó está a função que ele representa. 12
Figura 5 – Exemplo de regressão simbólica por meios da programação genética. 13
Figura 6 – Exemplo da instabilidade dos operadores. Adaptado de (FRANÇA, 2018). 13
Figura 7 – Recombinação entre duas cadeias de caracteres utilizando uma máscara, com 2 resultados possíveis. 17
Figura 8 – Métodos de busca. 19
Figura 9 – Derivação da árvore sintática para a expressão $x_1 + \pi * cos(x_2)$. . . 22
Figura 10 – Exemplo de equivalência (em azul) e não-interpretabilidade (em vermelho) que podem ocorrer em expressões representadas por árvores sintáticas. 23
Figura 11 – Ilustração do processo de *cross-validation*. 49
Figura 12 – Distribuição do RMSE das melhores expressões para 100 execuções, obtida a partir do empilhamento da distribuição de 10 em 10 execuções. 57
Figura 13 – Convergência da média do melhor RMSE de cada grupo de mutação individual sobre 100 execuções para cada base de dados. 59
Figura 14 – Convergência da média do melhor RMSE de cada grupo de mutação composto sobre 100 execuções para cada base de dados. 61
Figura 15 – Convergência da média do melhor RMSE de cada taxa de *crossover* sobre 100 execuções para cada base de dados. 63
Figura 16 – Convergência da média do melhor RMSE para o IT-MUT, IT-CX e ITEA sobre 100 execuções, variando a população ou o número de gerações. 66
Figura 17 – Raiz de uma expressão IT, contendo o coeficiente e o termo IT. . . 68
Figura 18 – Nó de interação genérico. 69

Figura 19 – Diagrama de Gantt mostrando o planejamento de execução do projeto. 84
Figura 20 – Variação do tamanho médio das melhores expressões para cada base de dados com o aumento da taxa de crossover. 89

LISTA DE QUADROS

Quadro 1 – Algumas funções que podem e não podem ser expressadas utilizando essa representação. 31
Quadro 2 – Representação computacional de expressões representáveis por meios da estrutura IT. 32
Quadro 3 – Algoritmos implementados e seus hiper-parâmetros. Em destaque estão os parâmetros específicos de cada algoritmo. 47
Quadro 4 – Algoritmos comparados e seus hiper-parâmetros. 51
Quadro 5 – Melhores hiper-parâmetros encontrados para o IT-MUT. 75

LISTA DE TABELAS

Tabela 1 – Bases de dados com suas respectivas dimensões. 48
Tabela 2 – RMSE médio das melhores expressões encontradas pelo algoritmo IT-MUT, para diferentes tamanhos de população ou número de gerações. 53
Tabela 3 – Pontuação e classificação final do algoritmo IT-MUT para diferentes tamanhos de população ou número de gerações. 54
Tabela 4 – Medianas das melhores expressões para os grupos de mutação isolados. 58
Tabela 5 – Medianas das melhores expressões para os grupos de mutação combinados. 60
Tabela 6 – Medianas das melhores expressões para diferentes taxas de *crossover*. 62
Tabela 7 – Medianas das melhores expressões para o algoritmo IT-MUT, para diferentes tamanhos de população ou número de gerações. 65
Tabela 8 – Medianas das melhores expressões para o algoritmo IT-CX, para diferentes tamanhos de população ou número de gerações. 65
Tabela 9 – Medianas das melhores expressões para o algoritmo ITEA, para diferentes tamanhos de população ou número de gerações. 67
Tabela 10 – Tamanho médio das melhores expressões encontradas para cada algoritmo desenvolvido com suas melhores configurações. 70
Tabela 11 – RMSE médio das melhores expressões para cada algoritmo desenvolvido. 73
Tabela 12 – NMSE médio das melhores expressões para cada algoritmo desenvolvido. 73
Tabela 13 – MAE médio das melhores expressões para cada algoritmo desenvolvido. 74
Tabela 14 – RMSE médio dos melhores resultados para o IT-MUT e os modelos comparados. 76
Tabela 15 – *p*-valor do tamanho das expressões para a base Airfoil. 86
Tabela 16 – *p*-valor do tamanho das expressões para a base Concrete. 86
Tabela 17 – *p*-valor do tamanho das expressões para a base Energy Cooling. . . 87
Tabela 18 – *p*-valor do tamanho das expressões para a base Energy Heating. . . 87

Tabela 19 – *p*-valor do tamanho das expressões para a base Tower Data. 87
Tabela 20 – *p*-valor do tamanho das expressões para a base Wine Red. 87
Tabela 21 – *p*-valor do tamanho das expressões para a base Wine White. 87
Tabela 22 – *p*-valor do tamanho das expressões para a base Yacht. 87
Tabela 23 – *p*-valor do tamanho das expressões para a base Airfoil para diferentes taxas de *crossover*. 88
Tabela 24 – *p*-valor do tamanho das expressões para a base Concrete para diferentes taxas de *crossover*. 88
Tabela 25 – *p*-valor do tamanho das expressões para a base Energy Cooling para diferentes taxas de *crossover*. 90
Tabela 26 – *p*-valor do tamanho das expressões para a base Energy Heating para diferentes taxas de *crossover*. 90
Tabela 27 – *p*-valor do tamanho das expressões para a base Tower Data para diferentes taxas de *crossover*. 90
Tabela 28 – *p*-valor do tamanho das expressões para a base Wine Red para diferentes taxas de *crossover*. 91
Tabela 29 – *p*-valor do tamanho das expressões para a base Wine White para diferentes taxas de *crossover*. 91
Tabela 30 – *p*-valor do tamanho das expressões para a base Yacht para diferentes taxas de *crossover*. 91
Tabela 31 – *p*-valor do RMSE das melhores expressões encontradas para a base Airfoil. 93
Tabela 32 – *p*-valor do RMSE das melhores expressões encontradas para a base Concrete. 93
Tabela 33 – *p*-valor do RMSE das melhores expressões encontradas para a base Energy Cooling. 93
Tabela 34 – *p*-valor do RMSE das melhores expressões encontradas para a base Energy Heating. 93
Tabela 35 – *p*-valor do RMSE das melhores expressões encontradas para a base Tower Data. 94
Tabela 36 – *p*-valor do RMSE das melhores expressões encontradas para a base Wine Red. 94

Tabela 37 – *p*-valor do RMSE das melhores expressões encontradas para a base Wine White. 94
Tabela 38 – *p*-valor do RMSE das melhores expressões encontradas para a base Yacht. 94
Tabela 39 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Airfoil. 95
Tabela 40 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Concrete. 95
Tabela 41 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Energy Cooling. 96
Tabela 42 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Energy Heating. 96
Tabela 43 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Tower Data. 96
Tabela 44 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Wine Red. 97
Tabela 45 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Wine White. 97
Tabela 46 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Yacht. 97

LISTA DE ALGORITMOS

Algoritmo 1 – Estrutura geral de um algoritmo genético. 21

Algoritmo 2 – Operador de origem (método *oporigem*). 37
Algoritmo 3 – Operador de mutação (método *opmutacao*). 39
Algoritmo 4 – Operador de seleção (método *opselecao*). 40
Algoritmo 5 – Operador de crossover (método *opcrossover*). 41
Algoritmo 6 – Função de torneio (ftournament). 43
Algoritmo 7 – Algoritmo IT-MUT. 43
Algoritmo 8 – Algoritmo IT-CX. 44
Algoritmo 9 – Algoritmo ITEA. 45

SUMÁRIO

1 – INTRODUÇÃO 1
1.1 PROBLEMA DE PESQUISA 2
1.2 OBJETIVO 3
1.3 JUSTIFICATIVA 4
1.4 ORGANIZAÇÃO DO TEXTO 4

2 – REFERENCIAL TEÓRICO 6
2.1 O PROBLEMA DE REGRESSÃO 7
2.2 ALGORITMOS DE REGRESSÃO PARAMÉTRICOS 8
2.3 ALGORITMOS NÃO-PARAMÉTRICOS 10
2.4 REGRESSÃO SIMBÓLICA 11

3 – REVISÃO DE LITERATURA 15
3.1 O PENSAMENTO EVOLUTIVO 15
3.2 A EVOLUÇÃO NA COMPUTAÇÃO 17
3.3 PROGRAMAÇÃO GENÉTICA 20
3.4 APLICAÇÕES NOTÁVEIS 24
3.5 TÉCNICAS UTILIZADAS ATUALMENTE 24

4 – ESTRUTURA INTERAÇÃO-TRANSFORMAÇÃO 28
4.1 REPRESENTAÇÃO COMPUTACIONAL 31
4.2 PROPRIEDADES DA ESTRUTURA 33

5 – ALGORITMOS EVOLUTIVOS IMPLEMENTADOS UTILIZANDO A ESTRUTURA INTERAÇÃO-TRANSFORMAÇÃO 36
5.1 OPERADORES 36
5.1.1 OPERADOR DE ORIGEM 36
5.1.2 OPERADOR DE MUTAÇÃO 38
5.1.3 OPERADOR DE SELEÇÃO 40
5.1.4 OPERADOR DE *CROSSOVER* 41

5.2 ALGORITMOS IMPLEMENTADOS 42
5.2.1 IT-MUT 43
5.2.2 IT-CX 44
5.2.3 ITEA 44

6 – METODOLOGIA 46
6.1 BENCHMARK 48
6.2 COMPARAÇÃO COM OUTRAS TÉCNICAS 50

7 – RESULTADOS E DISCUSSÃO 53
7.1 DEFINIÇÃO DE HIPER-PARÂMETROS GERAIS 54
7.2 DEFINIÇÃO DE HIPER-PARÂMETROS ESPECÍFICOS 55
7.2.1 GRUPOS DE MUTAÇÃO 58
7.2.2 TAXA DE *CROSSOVER* 62
7.2.3 VARIAÇÕES NO TAMANHO DA POPULAÇÃO E NO NÚMERO DE INDIVÍDUOS 64
7.3 ANÁLISE DOS RESULTADOS 68
7.3.1 TAMANHO DAS EXPRESSÕES 68
7.3.2 ERRO NA PREDIÇÃO 72
7.4 DESEMPENHO COMPARADO COM TÉCNICAS ATUAIS 75

8 – CONCLUSÃO 78
8.1 TRABALHOS FUTUROS 78
8.2 CONSIDERAÇÕES FINAIS 79

Referências 80

Apêndices 83

APÊNDICE A – PLANEJAMENTO DE EXECUÇÃO 84

APÊNDICE B – TESTE ESTATÍSTICO PARA O TAMANHO DAS EXPRESSÕES 86

APÊNDICE C–TESTE ESTATÍSTICO PARA O ERRO DE PREDIÇÃO 93

APÊNDICE D–TESTE ESTATÍSTICO PARA PERFORMANCE COMPARADO COM TÉCNICAS ATUAIS 95

1 INTRODUÇÃO

Técnicas de regressão buscam encontrar relações entre um conjunto de atributos e uma variável alvo. Isso é interessante pois permite obter um modelo matemático que descreve o comportamento da variável alvo de acordo com os atributos, possibilitando predizer novos valores para dados desconhecidos, ou encontrar relações entre os conhecidos e a variável alvo, possivelmente com um erro de predição baixo dependendo da qualidade do modelo.

Existem atualmente diversas técnicas de regressão, sendo classificadas na forma em que se baseiam para mapear os valores. Muitas dessas técnicas são amplamente conhecidas e possuem diversas aplicações, como a regressão linear, regressão polinomial, entre outras. Esses modelos são classificados em *(i)* paramétricos - que partem de funções matemáticas previamente estabelecidas e, pelo ajuste dos parâmetros livres dessa função tentam minimizar o erro entre a função e os resultados conhecidos - e *(ii)* não-paramétricos - que não partem de uma função inicial definida, ajustando tanto os parâmetros quanto a própria função. Os modelos não-paramétricos tem a vantagem de não estarem presos à uma forma fixa, mas tem a desvantagem do espaço de busca mais amplo, tornando a busca mais desafiadora.

Uma das técnicas de regressão não-paramétricas utilizada atualmente é a regressão simbólica, comumente realizada por meios da programação genética, um paradigma de otimização e busca inspirado no processo evolutivo da teoria da evolução proposta por Darwin, onde em uma população os indivíduos mais aptos tem maiores chances de se reproduzirem e gerarem descendentes que irão compor a nova geração. Esse processo é simulado computacionalmente, criando populações de soluções e, de forma probabilística, selecionando candidatos que serão progenitores da nova geração - essas soluções vitoriosas passam pelo processo de recombinação genética (ou *crossover*, uma troca de informações entre duas soluções, como na reprodução sexuada), onde a recombinação é vulnerável à mutação gênica, modificando aleatoriamente suas características e introduzindo variabilidade na população.

A regressão simbólica foi inicialmente pensada como uma aplicação direta da programação genética, porém em sua forma canônica ela apresenta um problema na

fragilidade das soluções, não conseguindo garantir uma transição suave cada vez que modifica uma função por meio da recombinação ou mutação. Além disso, apresenta um espaço de busca infinito, que permite que ocorram redundâncias ou expressões de alta complexidade e baixa interpretação.

Diversos algoritmos de regressão simbólica atuais, quando comparados com algoritmos de estado-da-arte de *gradient boosting*, apresentam resultados competitivos, mas com custo computacional muito mais alto do que técnicas tradicionais de regressão (ORZECHOWSKI; CAVA; MOORE, 2018).

Ainda, há críticas à respeito da precisão de algoritmos de regressão simbólica (KORNS, 2011), onde o autor argumenta que vários dos algoritmos de estado-da-arte falham em encontrar funções simples quando não há ruído nos dados.

Para minimizar os problemas atuais da aplicação de Programação Genética para o problema de Regressão Simbólica, podemos (FRANÇA, 2018):

- Restringir o espaço de busca;
- Priorizar funções simples;
- Mudar a construção e representação de expressões.

Pensando em atender esses três itens, em (FRANÇA, 2018) foi proposta uma estrutura denominada *Interação-Transformação* (IT) que restringe o espaço de busca em expressões matemáticas simples de tal forma a impedir a geração de expressões muito complexas. Nesse artigo foi também apresentado um algoritmo de busca que utilizava essa estrutura, denominado *SymTree*, e os resultados se mostraram competitivos em relação a diversos algoritmos de regressão linear e não-linear.

Em (ALDEIA; FRANCA, 2018), foram testadas diversas equações da física e da engenharia como *benchmarking* do algoritmo *SymTree*, apresentando um outro resultado interessante. Esse algoritmo conseguiu encontrar a expressão matemática mais simples e correta na grande maioria dos casos testados, servindo como uma prova de conceito da estrutura IT.

1.1 PROBLEMA DE PESQUISA

Atualmente, existem diversos algoritmos que podem ser classificados como pertencentes a família de algoritmos evolutivos, como *Programação Genética*, *Estratégias Evolutivas*, *Algoritmos Genéticos*, e outros que podem ser classificados como bio-

inspirados, como *Sistemas Imunológicos Artificiais*, *Otimização por Enxame de Particulas*, dentre outros. Cada uma dessas variações possui suas particularidades, embora todas possam ser definidas em um arcabouço evolutivo. Esses algoritmos compartilham algumas vantagens em relação a algoritmos de busca tradicionais como:

- Poder de exploração, evitando a atração para o ótimo local mais próximo da região inicial.
- Simplicidade de implementação, visto que a adaptação dos algoritmos para diferentes problemas requer pouco esforço.
- Bons resultados - encontramos na literatura diversos exemplos bem sucedidos de aplicação dessas técnicas em diversos problemas.

Com isso, esses algoritmos podem ser interessantes para a busca de expressões utilizando a estrutura IT para o problema de regressão simbólica.

O problema de pesquisa desse trabalho será a adaptação de um algoritmo evolutivo para a busca de expressões matemáticas representadas pela estrutura Interação-Transformação no problema de regressão simbólica.

1.2 OBJETIVO

O objetivo deste projeto é propor um algoritmo evolutivo de regressão simbólica baseado na estrutura Interação-Transformação, aprofundando as aplicações da estrutura e avaliando os resultados de sua aplicação em algoritmos evolutivos, comparando-os com os algoritmos bem sucedidos de regressão simbólica e *machine learning* amplamente utilizados atualmente.

Como objetivos específicos podemos citar:

- Criação de uma estrutura de dados que representa cada indivíduo da população.
- Definição da função de afinidade (*fitness*) utilizada pelo algoritmo para guiar a busca.
- Criação de um algoritmo de mutação capaz de introduzir novas soluções com potencial de melhora.
- Criação de um algoritmo de cruzamento capaz de combinar duas soluções com potencial de melhora.

1.3 JUSTIFICATIVA

O algoritmo *SymTree* apresenta evidências de que a estrutura IT permite a implementação de regressão simbólica de forma eficiente, evitando os problemas mais comuns presentes nos algoritmos de Programação Genética (FRANÇA, 2018).

Em outro trabalho o autor propõe uma estratégia evolutiva baseada na estrutura de dados (ALDEIA; FRANCA, 2018), que se mostrou satisfatória e capaz de realizar regressão simbólica com baixo custo computacional. O foco do trabalho foi uma prova de conceito da estrutura, não tendo a implementação evolutiva como principal objetivo. Ainda, o algoritmo foi implementado em *JavaScript*, uma linguagem executada em navegadores que apresenta diversas limitações, principalmente para algoritmos estocásticos, uma vez que o gerador de números aleatórios não apresenta distribuição uniforme.

Devido às desvantagens da regressão simbólica inerentes à forma que esta é comumente aplicada, através da programação genética, é de interesse explorar o desempenho que a estrutura IT apresenta na tarefa de regressão simbólica, para que possa ser analisado como essa representação interfere no poder de exploração do espaço de busca e na qualidade dos resultados, tentando fornecer algoritmos evolutivos competitivos, de baixo custo computacional e precisos.

O projeto busca propor um algoritmo evolutivo baseado na estrutura de dados e propor algoritmos que foquem na precisão e desempenho para a aplicação da estrutura em regressão simbólica.

A regressão simbólica por meios de algoritmos evolutivos é reconhecida como uma ferramenta útil, mas também é reconhecida como uma linha de pesquisa com problemas abertos que valem a pena serem estudados (KOTANCHEK; SMITS; KORDON, 2003).

1.4 ORGANIZAÇÃO DO TEXTO

Este relatório está organizado da seguinte forma. O capítulo 2 contém o referencial teórico para compreensão do trabalho. O capítulo 3 contém uma revisão bibliográfica, apresentando a história e inspiração da computação evolutiva, comentando suas origens e conceitos fundamentais, e introduzindo os algoritmos de regressão

que foram estudados neste projeto. O capítulo 4 apresenta a estrutura Interação-Transformação, analisa suas propriedades, e discute estratégias de implementação dos operadores evolutivos sobre tal estrutura. O capítulo 5 apresenta de forma aprofundada os algoritmos implementados, e discute como pode-se utilizar a estrutura IT em cada um dos algoritmos. O capítulo 6 apresenta a metodologia científica utilizada para testar e obter resultados dos algoritmos. O capítulo 7 apresenta os resultados e discussão. Finalmente, no capítulo 8 temos a conclusão do projeto.

2 REFERENCIAL TEÓRICO

Antes de continuar, cabe definir algumas convenções para tornar a leitura matemática do trabalho unificada, facilitando sua compreensão.

Escalares (valores reais) serão denotados por letras minúsculas em itálico (n, d, y, k); vetores (arranjos ordenados de n escalares) denotados por letras minúsculas em negrito ($\mathbf{x}$, $\mathbf{k}$); e matrizes (um arranjo ordenado de m vetores) denotadas por letras maiúsculas em negrito ($\mathbf{X}$). Quando feita uma referência à um elemento específico de um vetor ou matriz, seu índice é indicado sub-escrito ao lado da representação do vetor ou matriz, com a contagem iniciando-se em 1. Vetores terão seus elementos indicados por um único valor, enquanto matrizes terão um par de inteiros indicando sua posição.

A função alvo que deseja-se obter uma aproximação será denotada por letras minúsculas em itálico, seguida de n números reais entre parênteses indicando seus argumentos ($f(x_1, x_2, \ldots, x_n)$), e a função aproximada obtida através de um modelo de regressão por $\widehat{f}(x_1, x_2, \ldots, x_n)$. Termos utilizados comumente na literatura serão também utilizados aqui: uma amostra é um par ($\mathbf{x}$, y) com $\mathbf{x} \in \mathbb{R}^n$ um vetor que contém n atributos, e y um escalar real que resulta da aplicação de uma função desconhecida sobre $\mathbf{x}$, $f(\mathbf{x}) = y$:

$$(\mathbf{x}, y) = ((x_1, x_2, \ldots, x_n), y)$$

Um conjunto de amostras é chamado de base de dados, e é representado por uma matriz $\mathbf{X}$, onde cada linha é uma amostra e cada coluna um atributo; e uma matriz coluna $\mathbf{y}$ (interpretada como um vetor), onde cada linha é a variável alvo da amostra de índice respectivo:

$$\mathbf{X} = \begin{pmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \vdots \\ \mathbf{x_d} \end{pmatrix} = \begin{pmatrix} x_{(1,1)} & x_{(1,2)} & \cdots & x_{(1,n)} \\ x_{(2,1)} & x_{(1,2)} & \cdots & x_{(2,n)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{(d,1)} & x_{(d,2)} & \cdots & x_{(d,n)} \end{pmatrix}$$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{pmatrix}$$

A variável n comumente irá se referir ao número de atributos de uma amostra; d ao tamanho de uma base de dados; $\mathbf{x}$ ao vetor de atributos de uma amostra (com seus atributos (escalares) denotados por um índice sub-escrito); y a uma variável alvo; $\mathbf{y}$ ao vetor de variáveis alvo de uma base de dados; k representará um expoente (e $\mathbf{k}$ um vetor de expoentes, com $\mathbf{k} \in \mathbb{Z}^n$); $\mathbf{X}$ a matriz de atributos de uma base de dados. Os inteiros i e j serão utilizados para indicar índices de matrizes. Vale esta convenção a menos que seja explicitamente atribuído um novo significado para a notação.

2.1 O PROBLEMA DE REGRESSÃO

Técnicas que buscam encontrar relações entre uma variável alvo y e uma ou mais variáveis explanatórias $\mathbf{x}$, $\mathbf{x} = (x_1, x_2, \ldots, x_{n-1}, x_n)$ são conhecidas como técnicas de análise de regressão. Tais técnicas funcionam ajustando os parâmetros de uma determinada função $\widehat{f}$, buscando aproximar a função por meio dos pontos observados. Por se tratar de uma aproximação, a predição terá um erro relacionado (i.e. ruídos de medição), por isso muitos pesquisadores dedicam seus esforços à buscar por modelos e métodos que diminuam este erro associado aos modelos de regressão (JAMES et al., 2014). Existem dois principais interesses em encontrar uma função $\widehat{f}$ que se aproxime dos dados: **predição** e **inferência** (JAMES et al., 2014).

Predição: Ajustando uma equação conhecida à um conjunto de dados, é possível utilizar esta equação para predizer o valor de y dado uma configuração das variáveis explanatórias $\mathbf{x}$ que não precisam ter sido vistas anteriormente durante a fase de ajuste da função, pois teremos uma função $\hat{f} : \mathbb{R}^n \to \mathbb{R}$ que estabelece como as variáveis explanatórias se relacionam com a variável alvo y.

Inferência: Quando é possível estudar como a variável alvo se comportará de acordo com as mudanças nas variáveis explanatórias, podendo analisar o modelo buscando por respostas como:

- Qual a influência de cada variável no resultado?

- Quais as variáveis mais relevantes?
- As relações entre as variáveis são lineares? Não-lineares? Exponenciais?
- Qual a menor modificação que pode causar a maior variação no resultado?

A inferência está relacionada com a interpretabilidade do modelo. Quando não é possível encontrar respostas para essas perguntas e muitas outras que possam ser feitas, chamamos o modelo de *black box*, uma analogia à uma caixa preta, onde só temos conhecimento da entrada e saída, mas não podemos (e, se quiséssemos, não conseguiríamos) responder essas perguntas.

2.2 ALGORITMOS DE REGRESSÃO PARAMÉTRICOS

Existem atualmente diversas técnicas de regressão. Regressões que se baseiam em funções previamente conhecidas, realizando apenas os ajustes de parâmetros livres, são chamadas de regressões paramétricas. Em oposição, técnicas que não possuem inicialmente uma forma fixa de função para ser utilizada na aproximação são chamadas de regressões não-paramétricas (JAMES et al., 2014).

A vantagem de utilizar modelos paramétricos que assumem que os dados se comportam de uma maneira pré determinada e conhecida é que o problema de regressão se torna mais simples, sendo necessário apenas estimar os parâmetros livres (JAMES et al., 2014) com o objetivo de minimizar a distância entre uma função conhecida de uma função desconhecida. Por exemplo, a regressão linear, a forma mais simples de regressão, busca ajustar os parâmetros livres $\mathbf{a}$, com $\mathbf{a} = a_1, a_2, \cdots, a_n$ de uma combinação linear entre as variáveis de $\mathbf{x}$:

$$\widehat{f}(\mathbf{x}) = \widehat{f}(x_1, x_2, ..., x_{n-1}, x_n) = a_0 + \sum_{i=1}^{n} a_i \cdot x_i, \tag{1}$$

onde a_0 representa o intercepto, e $\mathbf{a}$ a inclinação da reta (ou, de forma mais generalizada, o hiper-plano caso $\mathbf{a} \in \mathbb{R}^n$ com $n > 1$) que queremos ajustar.

A desvantagem dos métodos de regressão paramétricos é a limitação a formas definidas, que nem sempre podem ser o suficiente para encontrar um modelo que proporcione de forma satisfatória a tarefa de *predizer* e *inferir*.

Nas figuras 1-3[1] temos à esquerda um conjunto de dados originários de

[1]Para realizar as regressões destas figuras, foram utilizadas as implementações da biblioteca

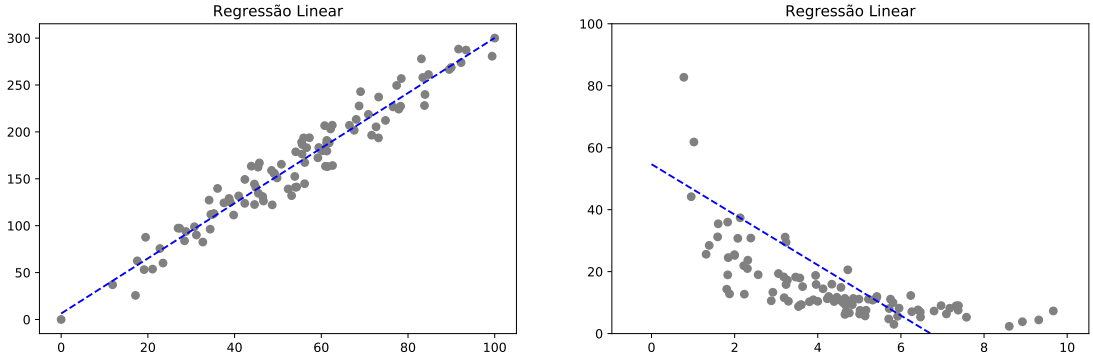


Figura 1 – Exemplo de regressão linear.

uma equação de reta $f(x) = a_1 x$ e à direita temos dados originários da equação $f(x) = 1/a_1 x$, ambas com um ruído adicionado.

É possível ver que para a regressão linear (figura 1) temos um resultado satisfatório para os dados da esquerda (que, no caso, sabemos que vieram de uma equação linear) mas para os dados à direita o resultado deixa a desejar. Além disso, a tendência é fazer previsões negativas para todo x maior que 6, o que não condiz com a equação que originou os pontos. Este método obtém resultados bons quando as variáveis apresentam alto grau de correlações entre si (FRANÇA, 2018). Apesar da interpretabilidade e simplicidade, a maioria dos problemas não apresentam relações puramente lineares, tornando essa técnica de regressão incapaz de modelar adequadamente uma solução.

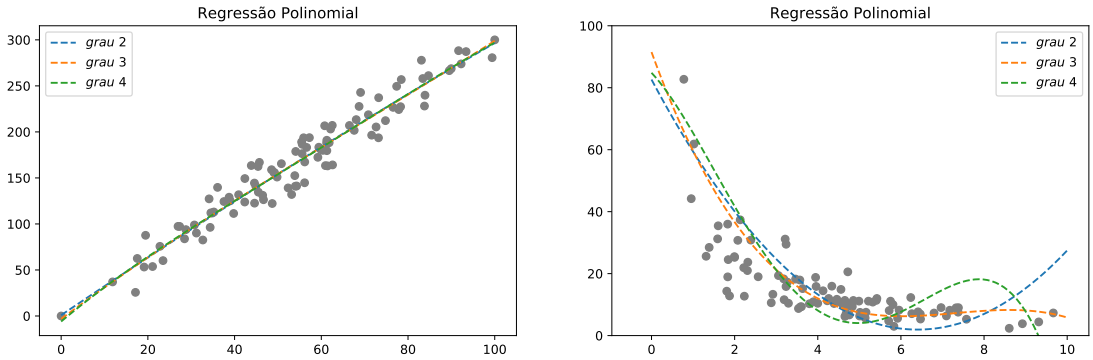


Figura 2 – Exemplo de regressão polinomial com diferentes graus.

Métodos não lineares de regressão são utilizados para esses casos onde a relação

scikit-learn (PEDREGOSA et al., 2011).

entre as variáveis explicatórias não é linear, tais como a regressão polinomial. Esta funciona ajustando os coeficientes de um polinômio de grau k. A regressão polinomial (figura 2) pode ajustar uma curva cuja complexidade é definida pelo grau da equação, tendo mais liberdade que a regressão linear (note que se o grau utilizado for 1 elas se comportam da mesma forma). É possível observar que ela é capaz de justificar bem os dados lineares, e descrever bem os dados da equação $f(x) = 1/a_1 x$, obtendo um resultado melhor que a anterior. Porém, o problema da regressão linear em explicar dados fora do intervalo utilizado para o treino também existe aqui. Na verdade, esse é um problema de técnicas de regressão em geral - a dificuldade de apresentar um comportamento satisfatório fora do intervalo dos dados utilizados para o treino (SMITS; KOTANCHEK, 2005).

2.3 ALGORITMOS NÃO-PARAMÉTRICOS

Por outro lado, temos regressões não-parametrizadas, que não partem do pressuposto de que a função tem forma fixa, podendo encontrar resultados mais expressivos que a regressão parametrizada, providenciando uma abordagem mais flexível para a regressão (JAMES et al., 2014), ao custo da necessidade de uma maior quantidade de dados e uma maior complexidade para obtenção de um resultado de boa qualidade. Por não partirem de uma suposição, tem-se um grande número de parâmetros que esses modelos precisam estimar, tornando-os mais propícios ao *overfitting*, quando a função se ajusta excessivamente a um conjunto de dados específicos, falhando em predizer e justificar resultados para novos pontos desconhecidos.

O modelo mais conhecido de regressão não-paramétrica é o *K-Nearest-Neighbors* (KNN) (SIVANANDAM; DEEPA, 2007), método que apresenta simplicidade e ainda sim obtém bons resultados. Este modelo não assume uma função prévia, e para fazer predições funciona utilizando a regra dos k vizinhos mais próximos, calculando a média dos k vizinhos para estimar o valor de um ponto desconhecido (Cover; Hart, 1967).

O KNN geralmente é utilizado para classificação, definindo rótulos para novos dados baseando-se nos conhecidos, mas também pode ser implementado para dados contínuos (o caso da regressão).

É desejado que k seja um número que minimize o erro na estimação de uma amostra desconhecida, mas também seja pequeno o suficiente (em relação à proporção

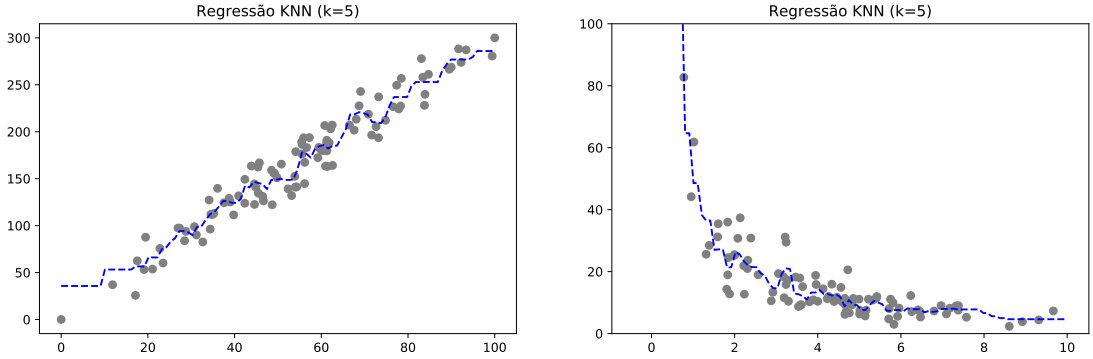


Figura 3 – Exemplo de regressão KNN.

dos números de amostras) para não utilizar informação de pontos mais distantes (Cover; Hart, 1967). A escolha do valor de k tem um efeito drástico pois define a flexibilidade dos limites entre os diferentes grupos de classificação. Um valor muito grande torna o modelo muito inflexível e os limites muito justos (JAMES et al., 2014).

Uma abordagem paramétrica supera a não-paramétrica quando a forma verdadeira em que os dados se comportam é próxima à forma paramétrica (JAMES et al., 2014). Por exemplo, para o KNN (figura 3) vemos que para os dados lineares (à esquerda) o modelo se ajusta muito aos pontos e apresenta um comportamento mais complexo e confuso que uma simples reta, como visto na regressão linear, onde os dados se comportam da mesma forma que a função paramétrica.

2.4 REGRESSÃO SIMBÓLICA

A regressão simbólica é uma técnica de regressão não-paramétrica. Isso significa que não se parte de um pressuposto inicial, e sim que os próprios dados ajudam o modelo a definir quais variáveis, constantes e funções serão utilizadas e como serão combinadas para descrever o comportamento observado (SMITS; KOTANCHEK, 2005).

Esse método de regressão costuma ser implementado através da programação genética, um algoritmo de otimização baseado em população que busca por expressões matemáticas, representando-as computacionalmente através de árvores sintáticas. Apesar de existirem outras abordagens para a regressão não-paramétrica, modelos populacionais e evolutivos são bastante estudados para essa aplicação, sendo a programação genética amplamente conhecida como um meio de realizar essa tarefa.

A representação por árvores sintáticas utiliza a representação de árvores binárias para compor funções, onde os **nós** representam funções e operações matemáticas, as arestas a aplicação de funções, e as folhas as constantes e variáveis. Chamamos de **nó filho de** v os nós no nível abaixo de v que possuem uma aresta os conectando a v; e **nó pai de** v o nó acima de v que possui uma aresta os conectando. Uma árvore sintática é uma estrutura de dados hierárquica, de forma que cada nó aplica sua função sobre os nós filhos. Um nó pode ter vários filhos, mas apenas um único pai. O nó mais alto da hierarquia é denominado **raiz** da arvore, e os nós mais baixos sem nenhum filho são denominados **folhas**.

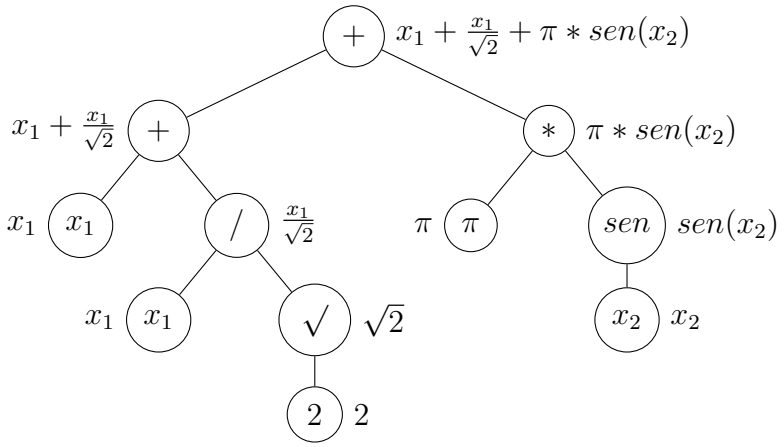


Figura 4 – Composição de uma função por meio de uma árvore. Ao lado de cada nó está a função que ele representa.

Na figura 4 temos uma árvore onde as folhas são as variáveis e constantes, e os demais nós uma função ou operação matemática, que é a aplicada sobre os seus nós filhos. Podemos expressar números racionais e irracionais, combinações lineares, frações, multiplicações e aplicações de funções trigonométricas com essa representação. Porém, a quantidade de funções que podem ser criadas é infinita, assim como a quantidade de funções equivalentes, fazendo com que a obtenção de um bom resultado seja desafiadora.

Na figura 5 temos o resultado de uma regressão simbólica por meios da programação genética, que utiliza árvores sintáticas para representação de expressões matemáticas[2]. A regressão simbólica oferece contribuições únicas, principalmente quando

[2]Para realizar esta regressão, foi utilizada a biblioteca GPLearn.

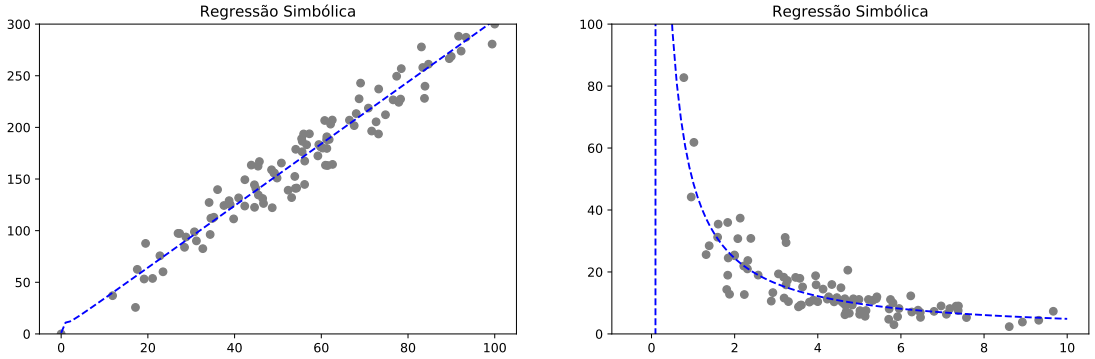


Figura 5 – Exemplo de regressão simbólica por meios da programação genética.

utilizada em um conjunto de dados não linear mas com um comportamento desconhecido (SMITS; KOTANCHEK, 2005).

A programação genética cria uma população de soluções aleatórias, e simula gerações onde as soluções passam por processos equivalentes ao *crossover* e a mutação biológica, recombinando as características genéticas de dois indivíduos para formar uma nova solução, com uma pequena chance de ocorrer alguma modificação aleatória no processo[3].

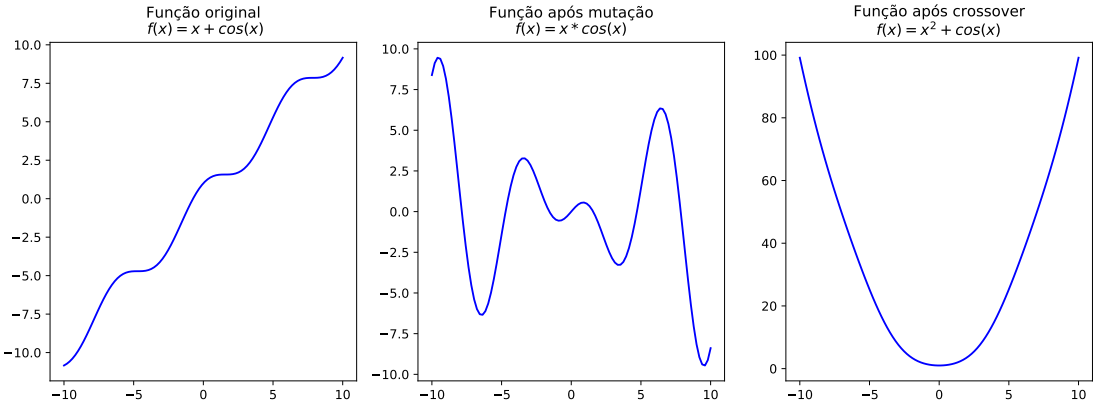


Figura 6 – Exemplo da instabilidade dos operadores. Adaptado de (FRANÇA, 2018).

Mas utilizar a representação de equações por árvores faz com que a recombinação e a mutação nem sempre resulte em uma nova função que mantenha relação com a função original (FRANÇA, 2018). Na figura 6 temos um exemplo onde uma mutação

[3] A mutação e o *crossover* serão apresentados com mais detalhes no capítulo 3.

modifica o nó de soma da expressão para um nó de multiplicação, e o *crossover* troca uma parte da árvore (x) por outra parte de uma expressão pertencente a população (no caso, x^2).

Nesse capítulo vimos um pouco sobre o problema de regressão e, especificamente, o de regressão simbólica utilizando programação genética. A área de estudos da regressão simbólica é recente, tendo origem como a principal aplicação da programação genética (KOZA, 1992). Esta área tem potencial promissor de encontrar soluções admissíveis para problemas das mais diversas áreas com a promessa de ajustar não só os coeficientes, mas também uma fórmula adequada (KORNS, 2011).

3 REVISÃO DE LITERATURA

Para entender as origens biológicas que inspiraram a computação evolutiva, este capítulo apresentará uma breve história do pensamento evolutivo, que se modificou bastante até chegar no que hoje é amplamente aceito como teoria da evolução. Também será feita uma introdução aos primeiros modelos de algoritmos evolutivos, e por fim um breve panorama sobre suas aplicações e outras técnicas muito utilizadas atualmente para realizar a tarefa de regressão.

3.1 O PENSAMENTO EVOLUTIVO

Até antes do século XIV, várias correntes de pensamento de naturalistas que buscavam explicar as diferentes espécies existiram. Essencialmente, tais teorias (chamadas de teorias criacionistas) acreditavam que cada espécie era única e tinha um propósito específico na natureza, de forma que uma espécie não evoluía para uma nova.

Na verdade, a ideia de evolução surgiu primeiramente com o trabalho de Jean Baptiste Lamark em 1802 - *Transformation Hypothesis* - onde foi apresentado uma teoria da evolução dos seres, que argumentava que as espécies não eram fixas, mas por pressão do meio ambiente os organismos se adaptavam, e dessa forma todos os organismos se direcionavam para uma maior complexidade. Através da transmissão das características adquiridas para seus descendentes era definido então o processo de evolução dos seres.

Charles Darwin, influenciado pelas ideias de Lamark e Thomas Malthus, realizou estudos que o levaram a postular a atualmente aceita teoria da evolução. A teoria de Darwin diz que indivíduos de uma mesma espécie são naturalmente diferentes (essas diferenças (ou variações) são intrínsecas ao mundo orgânico) e lutam pela sobrevivência. Essa luta se dá por meio da seleção natural, onde os diferentes traços de cada indivíduo influenciam na sobrevivência e reprodução do organismo em resposta ao ambiente, e esses traços particulares de cada indivíduo podem ser herdados por gerações futuras, se tornando mais frequentes os traços que melhor respondem ao ambiente.

Mas a teoria de Darwin não conseguiu explicar adequadamente como se dá o surgimento de diferentes traços e como esses são herdados pelas gerações futuras. Em

seu trabalho, a explicação da herança de características é explicada utilizando ideias Lamarkianas (SILVA, 2001). Contemporâneo de Darwin, Mendel veio posteriormente a responder tais questões, propondo um modelo teórico para explicar a herdabilidade.

Os dois pilares da teoria da evolução moderna foram então erguidos pelas observações de Darwin e pelo trabalho de Mendel.

A teoria de Darwin contrasta com Lamark no papel do meio ambiente e na forma que os traços são passados: para Lamark, o meio ambiente impõe um obstáculo para as espécies, que por sua vez vão gradativamente sendo forçadas a modificarem-se para continuar sobrevivendo, e essas modificações são transferidas para seus descendentes, de forma que a evolução seria uma escala de complexidade - enquanto Darwin acreditava que o meio ambiente exercia papel secundário, impondo uma pressão seletiva sobre as diferentes modificações que surgiam, favorecendo a sobrevivência dos mais aptos ao meio, que gerariam descendentes com suas características herdadas.

Com isso, podemos extrair da teoria de Darwin a ideia de mecanismos evolutivos, que serviram de inspiração para simular o processo de evolução computacionalmente. Temos, essencialmente, três mecanismos gerais responsáveis pela evolução:

- Variação - em uma população de uma mesma espécie cada indivíduo apresentará pequenas variações entre os demais, alguns com variações que apresentam maior aptidão ao meio, favorecendo seu sucesso de sobrevivência e reprodução;
- Seleção - os recursos são limitados, sendo necessário uma competição dentro da própria população, de forma que os mais aptos tenham uma maior chance de melhor resposta ao meio;
- Herança - Os indivíduos que forem capazes de se reproduzir podem ter seus traços herdados pelas futuras gerações.

Essas modificações genéticas implicam numa (possível) pequena melhoria a cada geração. Dado tempo suficiente, a consecutiva acumulação dessas pequenas melhorias através da herança pode gerar indivíduos muito diferentes dos ancestrais e totalmente únicos, criando novas espécies.

Ou seja, é por meio dos mecanismos evolutivos que ocorreu o surgimento de formas complexas, como uma constante seleção de modificações que favorecem o sucesso de sobrevivência e sucesso reprodutivo.

3.2 A EVOLUÇÃO NA COMPUTAÇÃO

A computação evolutiva é um dos paradigmas biologicamente inspirados que continua forte desde sua origem (BEYER; SCHWEFEL, 2002).

Os algoritmos evolutivos são um paradigma de busca, uma forma de gerar inteligência. Esta visa por meio dos mecanismos evolutivos encontrar um bom resultado para problemas que possam ser modelados computacionalmente. A ideia da computação evolutiva é simular o processo evolutivo para tentar encontrar soluções boas, utilizando populações adaptativas que são modificadas iterativamente com a aplicação de mecanismos evolutivos em busca de uma boa solução para o problema (BEYER; SCHWEFEL, 2002).

Na computação, a primeira ideia de aplicação de mecanismos genéticos para solução de problemas foi proposta por John Henry Holland (HOLLAND, 1975), que criou os *algoritmos genéticos*. Tais algoritmos se baseavam na teoria da evolução de Darwin combinados com operadores genéticos (recombinação e mutação) para buscar uma solução de um problema, onde as soluções são representadas como uma cadeia de caracteres (geralmente binários) (VIKHAR, 2016). Para a representação por árvores, o *crossover* realiza uma poda nas duas árvores que irão originar uma nova, e então trocam entre elas a seção podada; e a mutação substitui o valor de um nó (que pode ser um nó folha, com variáveis e constantes, ou um nó de função).

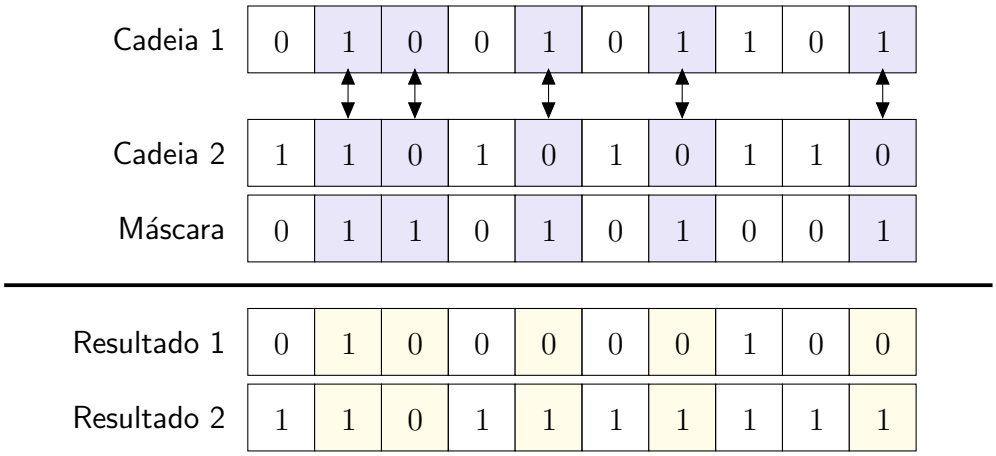


Figura 7 – Recombinação entre duas cadeias de caracteres utilizando uma máscara, com 2 resultados possíveis.

Na figura 7 temos um exemplo de como a recombinação (ou *crossover*) pode ser implementado. Neste exemplo, a informação das soluções é representada por uma cadeia de caracteres. O *crossover* ocorre entre duas soluções utilizando uma máscara que irá selecionar 50% dos elementos da cadeia de caracteres para realizar uma troca entre as soluções, resultando em uma (das duas possíveis) solução com as características herdadas dos dois pais.

Um tempo depois, John Koza criou a **programação genética** (KOZA, 1992), que se baseia na mesma representação de Holland, utilizando cadeias de caracteres para expressar um equivalente ao DNA das soluções, mas, em sua versão, a cadeia descreve árvores sintáticas, onde os nós são elementos utilizados para compor uma solução final. Podemos utilizar operações matemáticas e números para criar funções; operações lógicas para criar expressões booleanas; ou instruções em código para criar um programa.

Depois desses trabalhos, começaram a surgir várias outras famílias de algoritmos biologicamente inspirados para processos de otimização. Atualmente, o conjunto de famílias de algoritmos baseados em tais operadores evolutivos são parte de um grande grupo, chamado de *algoritmos evolutivos*.

Os **algoritmos evolutivos** são um subgrupo dos algoritmos de busca estocásticos e são uma metaheurística baseados em populações (VIKHAR, 2016). São algoritmos que **se baseiam no processo evolutivo** (gerações, aptidão, seleção, variabilidade (através da mutação)), **podendo considerar também fatores genéticos** (reprodução e troca de informação genética, herança de características). Esses 4 grupos do diagrama (figura 8) são os mais conhecidos, e a diferença entre eles é basicamente os operadores (evolutivos e genéticos) que cada um utiliza, e a forma que representam a solução.

Os algoritmos estocásticos realizam a busca de forma aleatória porém direcionada (no caso dos algoritmos evolutivos, pela função de aptidão, que deve ser determinada de forma que avalie a qualidade da solução e reflita-a de uma forma que possa ser comparada com outras), de forma que sejam exploradas apenas opções que se apresentam promissoras dentro do problema. Tal característica aleatória que torna os resultados interessantes e potencialmente promissores: ela possibilita explorar combinações novas, variações dessas novas combinações e espaços inimagináveis.

Este comportamento estocástico dá o poder de busca ao algoritmo, mas

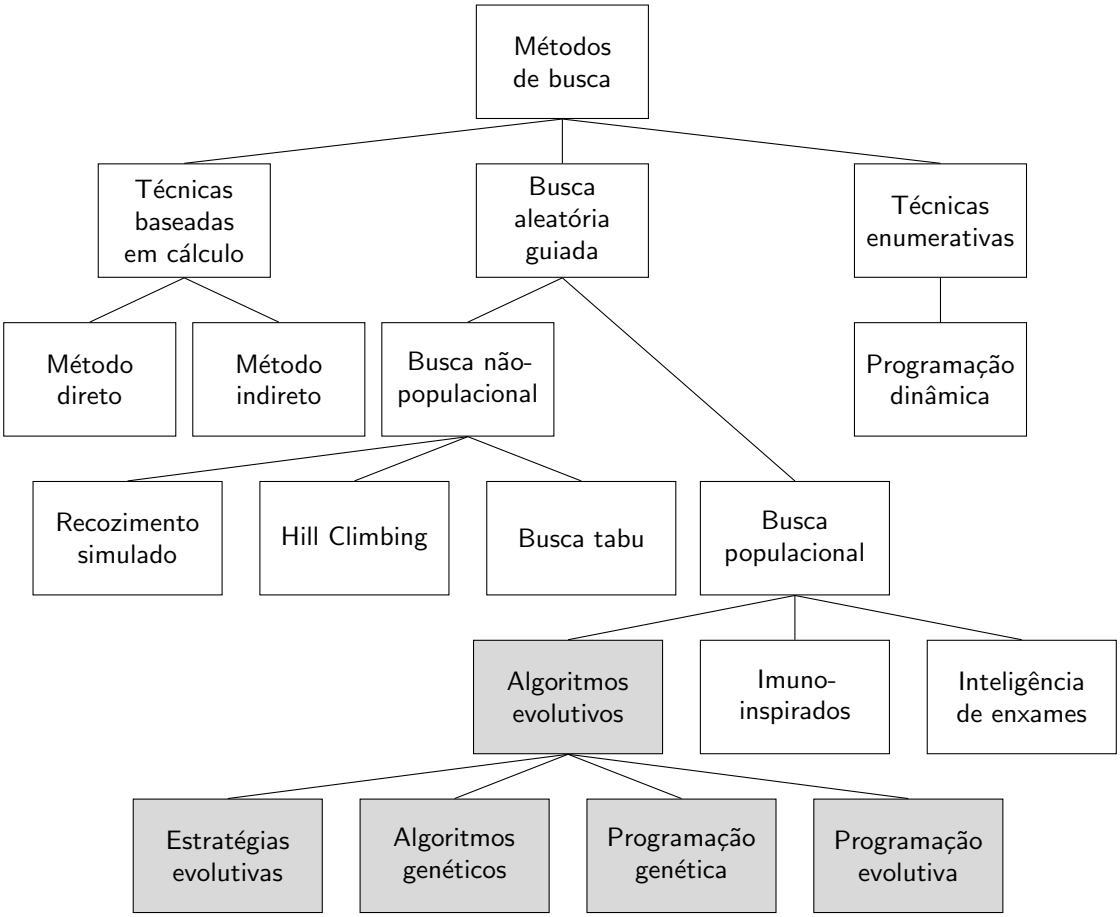


Figura 8 – Métodos de busca.

também adiciona uma incerteza ao resultado. Pode ser que seja necessário muito tempo para que o algoritmo explore as soluções e convirja para uma solução definitiva, ou um resultado obtido em uma evolução para solucionar um problema complexo seja único e não venha a ocorrer novamente em outras execuções. Não podemos assumir que o algoritmo encontrará sempre a mesma solução, ou que a solução encontrada será sempre ótima, mas eventualmente ele encontrará uma aceitável se essa existir na forma em que as representa.

Quando utilizados como modelos de otimização, os algoritmos evolutivos perdem em complexidade para os modelos *black box*, pois não ajustam parâmetros de um modelo pré definido, e sim criam iterativamente populações inteiras de modelos e os ajustam individualmente. Isso adiciona um maior grau de complexidade na busca por

ter mais parâmetros para ajustar, porém evitam que a busca fique presa em soluções locais.

As principais fontes de algoritmos evolutivos atualmente são: Programação evolutiva (*evolutionary programming*, **EP**), algoritmos genéticos (*genetic algorithms*, **GA**) e estratégias evolutivas (*evolution strategies*, **ES**) (BEYER; SCHWEFEL, 2002). Todas essas sub-áreas funcionam baseando-se nos mesmos princípios, mas diferem nas formas de implementações e aplicações (VIKHAR, 2016).

O ES foi proposto em 1973 por Rechenberg (VIKHAR, 2016), e se baseia principalmente no uso de mutação e seleção. Já o GA é a forma que apresenta um mapeamento mais coerente entre o processo de evolução natural e o processo de evolução simulado em computador (VIKHAR, 2016), pois a representação das soluções é feita utilizando-se uma cadeia de caracteres que representam os genes da solução. Isso se aproxima do mundo real por encorporar conceitos de fenótipo e genótipo, por ter processos de mutação (modificando aleatoriamente porções do material genético) e *crossover* (onde há troca de "fragmentos" de material genético) bastante similares aos nossos.

3.3 PROGRAMAÇÃO GENÉTICA

A programação genética é uma forma autônoma de inteligência, com capacidade de criar uma solução geral na forma de uma topologia parametrizada para um problema.

Um algoritmo genético (ilustrado pelo algoritmo 1) cria e modifica soluções computacionais para um problema especificado, modelando uma estrutura final de acordo com a forma que a aptidão (chamada de *fitness*, uma função para abstrair a qualidade da solução (VIKHAR, 2016)) de cada solução é calculada.

O algoritmo genético funciona simulando um processo de seleção natural por uma quantidade definida de gerações. Cada geração começa com uma população de soluções P, e termina com uma população de soluções P', onde P' é uma população resultante da aplicação de *crossover*, mutação e seleção na população P. Primeiro é aplicado entre os indivíduos da população uma recombinação, onde são trocados aleatoriamente "fragmentos" entre as soluções (um equivalente ao *crossover* que ocorre em seres vivos durante a reprodução sexuada). O *crossover* da programação genética pode aplicar mutações nas soluções, assim como ocorre biologicamente, adicionando

Algoritmo 1: Estrutura geral de um algoritmo genético.

Entrada: Parâmetros de execução, critério de parada
Saída: função simbólica f

P $\leftarrow$ [n Soluções aleatórias];
para g *Gerações* **faça**
  P' $\leftarrow$ Crossover(P);
  P' $\leftarrow$ Mutação(P');
  P $\leftarrow$ Seleção($P + P'$);
  se *critério de parada foi atingido* **então**
    Parar evolução;
  fim
fim
retorna maxFitness(P);

uma maior variabilidade na população. Após isso, para criar a população P' uma seleção é aplicada em P, de forma que as soluções com maior *fitness* tenham maior chance de serem selecionadas, onde o *fitness* é uma forma de quantificar o quão boa cada solução gerada é. Comumente, a seleção aplicada é a seleção por torneio, onde soluções aleatórias se "confrontam", sendo vencedora a solução de melhor *fitness*.

Por exemplo, no caso da regressão simbólica, se o objetivo é minimizar o erro de predição, a função de aptidão pode utilizar uma métrica que incorpore essa informação (como o *Mean Squared Error*, **MSE**) para que, durante um confronto, aquela que apresenta o menor erro seja considerada a vencedora do torneio.

A programação genética pode ser aplicada a qualquer problema, desde que os seguintes itens sejam bem definidos:

- Conjunto de entrada,
- Conjunto de busca,
- Parâmetros de execução,
- Critério de seleção (*fitness*),
- Critério de parada.

A combinação do conjunto de entrada e o conjunto de busca determinam a linguagem usada para a criação das soluções. O critério de seleção define como serão avaliadas a aptidão de cada solução - a evolução é orientada para a direção das soluções que maximizam o *fitness*. Os parâmetros de execução e o critério de parada são definidos

de acordo com o problema. É importante que a representação do problema seja feita de forma adequada computacionalmente - o propósito dessa representação é fazer uma ponte entre o mundo real e o mundo da computação evolutiva (VIKHAR, 2016).

Para a regressão simbólica, definimos o conjunto de entrada como variáveis e constantes, e o conjunto de busca como o conjunto de todas as operações matemáticas unárias e binárias que podem ser aplicadas. O *fitness* deve ser definido de forma a priorizar equações que se aproximem dos valores esperados.

As expressões resultantes são geralmente representadas por árvores sintáticas, onde as folhas das árvores são variáveis e constantes do conjunto de entrada, e os nós são as funções unárias e binárias, contidas no conjunto de busca (figura 9).

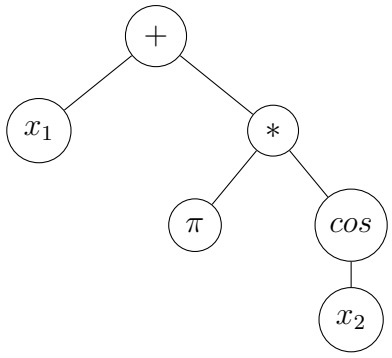


Figura 9 – Derivação da árvore sintática para a expressão $x_1 + \pi * cos(x_2)$.

A árvore da figura 9 é composta por duas variáveis (x_1 e x_2), uma constante (π), duas funções binárias ($+$, $*$) - que requerem 2 valores reais para operar - e uma função unária (cos) - que requer apenas um valor. A aridade da função de cada nó determina a quantidade de filhos que ele terá - variáveis e constantes não possuem filhos, funções unárias possuem um único filho, e funções binárias possuem dois. Para avaliar o resultado desta árvore, dados valores para x_1 e x_2, percorremos a árvore sintática partindo da raiz até atingirmos as folhas, que serão sempre números reais (constantes ou variáveis), então substituímos os valores dados para as variáveis e fazemos o caminho inverso, sempre resolvendo a função de cada nó aplicando-a sobre o resultado de seus nós filhos, até obtermos um único valor real que será o resultado da avaliação da árvore para a entrada fornecida.

A programação genética é aplicável quando há necessidade de obter soluções complexas em um amplo espaço de busca, por contar com uma otimização combinatória

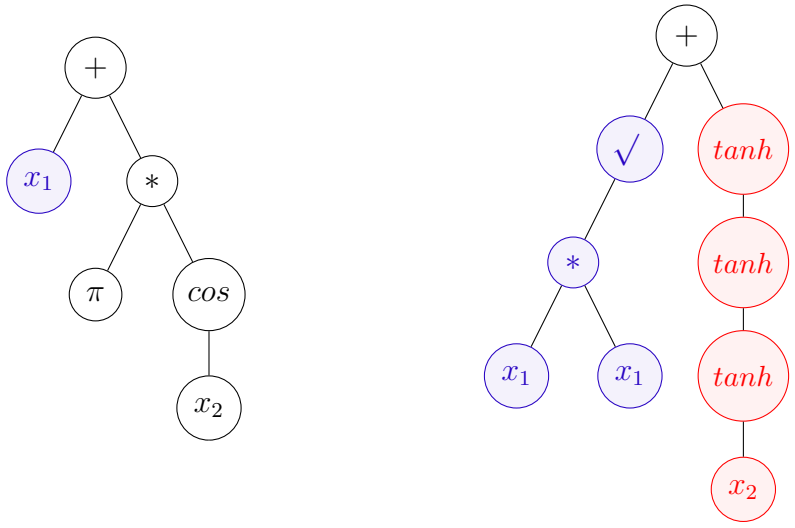


Figura 10 – Exemplo de equivalência (em azul) e não-interpretabilidade (em vermelho) que podem ocorrer em expressões representadas por árvores sintáticas.

das soluções no espaço de busca. Porém, na regressão simbólica, o espaço de busca é infinito, podendo dificultar a tarefa. Além disso, podem ocorrer equações equivalentes ou contendo redundâncias, assim como resultados não interpretáveis (ilustrados na figura 10), ao contrário de outros modelos de regressão, onde é possível ver a relação entre as variáveis. Dessa forma, a regressão perde a grande vantagem que tem sobre modelos *black box* de regressão, como as redes neurais ou o *support vector machines* (KORNS, 2011).

Para tornar mais viável a aplicação de algoritmos evolutivos, podemos realizar uma série de modificações. Algumas possíveis modificações são (FRANÇA, 2018):

- Restringir o espaço de busca;
- Priorizar funções simples;
- Mudar a construção e representação de expressões.

Também podemos adicionar penalidades na função de fitness (COELLO, 2002), o meio mais comum de restringir o espaço de busca do algoritmo. Essa penalidade serve para desencorajar o algoritmo a explorar regiões que não são de interesse para o objetivo. Mas até a penalidade apresenta problemas: como a solução é desconhecida, pode ser que a penalidade acabe limitando o algoritmo à uma área onde a solução ótima não se encontra. Além disso, definir uma penalidade muito ou pouco rígida pode

definir a velocidade como a exploração é feita, tornando o algoritmo evolutivo muito ineficaz por não conseguir encontrar uma solução em tempo praticável. Por isso, vários pesquisadores se dedicam ao estudo de heurísticas para definição de penalidades eficazes (COELLO, 2002).

3.4 APLICAÇÕES NOTÁVEIS

O uso de algoritmos genéticos para busca e otimização tem crescido nos últimos anos. Devido à simplicidade, flexibilidade e robustez, os algoritmos evolutivos se tornaram uma técnica popular de resolução de problemas nas áreas de aprendizado e otimização (VIKHAR, 2016). Um exemplo foi a utilização para busca de um design eficiente que atendesse uma série de requerimentos para uma antena, que seria utilizada em uma missão espacial da NASA em 2005 (HORNBY et al., 2006).

Também tem sido utilizada para buscar novas topologias de redes neurais, processo conhecido como neuro-evolução. Isso mostra a união entre as ideias da computação conexionista e evolucionista - o processo evolutivo molda a forma do cérebro, sua estrutura; e então o aprendizado é feito inspirado nas evidências neurofisiológicas que temos.

A computação evolutiva pode ser utilizada para descobrir novas estratégias em jogos digitais, em oposição às redes neurais, por apresentar uma melhor interpretabilidade (WILSON et al., 2018).

A regressão simbólica também tem sido aplicada no campo industrial para modelagem, motivada pela necessidade de obter *insights* de forma rápida e eficiente a partir de dados (KOTANCHEK; SMITS; KORDON, 2003).

Esses são alguns exemplos de aplicações práticas interessantes de algoritmos evolutivos que enfatizam as características apresentadas anteriormente: possuem uma melhor interpretabilidade, podem ser competitivos com redes neurais, são capazes de explorar espaços de busca complexos e podem nos ajudar a ter novas ideias.

3.5 TÉCNICAS UTILIZADAS ATUALMENTE

Devido ao crescente interesse no aprendizado de máquina e ao aumento do poder computacional dos computadores modernos, métodos de regressão passaram a

ser amplamente utilizados por serem aplicáveis em várias áreas práticas. Redes neurais e *support vector machines* por exemplo, não são modelos novos - seus conceitos existem desde a década de 60, mas só nos últimos anos que passaram a ganhar interesse por poderem ser implementados em tempo hábil.

Costumam ser escolhidos modelos que apresentam poucos hiper-parâmetros para serem ajustados. Entre os modelos de regressão amplamente utilizados atualmente, podemos destacar alguns:

Ordinary Least Squares (OLS) Modelo mais simples de regressão paramétrica que faz os ajustes dos coeficientes de uma combinação linear entre as variáveis utilizando o método de mínimos quadrados (OLS), buscando minimizar o erro médio quadrático (*mean squared error*, MSE).

Ridge Regression Regressão paramétrica que funciona de forma parecida à regressão linear, mas impõe uma penalidade nos coeficientes ao minimizar uma função de erro diferente, causando a redução de alguns coeficientes para valores próximos de zero. Essa penalidade resulta em um modelo com uma menor variância nas predições (JAMES et al., 2014).

Least Angle Regression (LARS) com Lasso Este modelo de regressão paramétrico funciona de forma parecida ao Ridge, mas na regressão Ridge os coeficientes tendem a zero sem nunca assumir esse valor, não necessariamente influenciando na acurária mas sim na interpretabilidade do modelo (JAMES et al., 2014). O LARS permite que os coeficientes sejam anulados, diminuindo o número de variáveis de $\widehat{f}$ (quando possível), gerando melhoras na interpretabilidade.

K-Nearest-Neighbors (KNN) Modelo de regressão não-paramétrico que utiliza a informação dos k vizinhos mais próximos (sendo a distância entre os vizinhos calculada por uma métrica de distância, comumente sendo utilizada a distância euclidiana) para predizer novos valores para dados desconhecidos.

Bayesian Ridge Regression (Bayesian) Este modelo também funciona como uma regressão linear, mas utiliza a inferência Bayesiana para realizar novas predições, assumindo que os valores de $\mathbf{y}$ vem de uma distribuição normal.

Extreme Gradient Boosting (XGBoost) O XGBoost faz parte da família de métodos de regressão baseados em árvores - algoritmos que envolvem a segmentação do espaço de predição em várias regiões mais simples por meio de árvores de decisão, produzindo múltiplas árvores que combinadas resultam em uma única

expressão. São modelos poderosos para interpretação por utilizarem árvores de decisão, que se assemelha ao processo de tomada de decisão humano. Apesar da interpretabilidade, não apresentam resultados competitivos contra os melhores algoritmos de aprendizado de máquina em termos de acurácia (JAMES et al., 2014) - mas com o uso de um número grande de árvores é possível contornar o problema de acurácia (FRIEDMAN, 2000), depreciando a interpretação. Para a regressão, o espaço de predição é dividido em regiões sem sobreposição, e, para cada observação que cair em uma região específica, é calculada a média dos valores conhecidos dela (JAMES et al., 2014).

Multi Layer Perceptron (MLP) Redes Neurais Artificiais (RNA) são modelos biologicamente inspirados na forma que o cérebro humano e de outros animais processa informações. Uma RNA é um grafo orientado de camadas que contém neurônios artificiais. Temos essencialmente 3 camadas:

1. *Input layer* (camada de entrada): esta camada contém um neurônio para cada variável imputada na rede, e seu papel é alimentar cada neurônio da próxima camada com o valor de entrada;
2. *Hidden layers* (camadas ocultas): são todas as camadas entre a camada de entrada e a camada de saída, exercendo um papel intermediário de "transformar a camada de entrada em algo que a camada de saída consiga utilizar". Cada camada escondida adicional aumenta o grau de complexidade da rede neural, permitindo a modelagem de funções cada vez mais complexas.
3. *Output layer* (camada de saída): é a camada responsável por aplicar uma função de ativação na informação vinda da camada anterior e fornecer como saída da rede um resultado.

Uma rede neural com apenas uma camada oculta e um único neurônio de saída tem capacidade de aproximação de qualquer função contínua limitada, com um erro ϵ pequeno, desde que a quantidade de neurônios na camada oculta seja suficiente (HORNIK, 1991). Porém nem toda configuração de parâmetros da rede apresentará um bom desempenho dependendo do problema de aprendizado em que a rede for submetida. Dessa forma, alguns problemas podem precisar de uma quantidade de neurônios considerável, tornando ainda mais obscuro o significado matemático da rede neural, ou precisar de uma quantidade tão grande de neurônios na camada oculta que torne o modelo inaplicável.

Neste capítulo foi apresentada a programação genética utilizando representação em árvores, e seus problemas relacionados à isso. Além disso, foram apresentadas possíveis aplicações da computação evolutiva e regressão simbólica e outros métodos que costumam ser utilizados atualmente, junto com suas características particulares.

4 ESTRUTURA INTERAÇÃO-TRANSFORMAÇÃO

Por conta dos problemas da programação genética realizada por meios de árvores de expressão apresentados anteriormente, onde a aplicação de uma mutação ou *crossover* em uma função pode gerar outra sem relação com o comportamento qualitativo da original, foi proposta em (FRANÇA, 2018) a utilização de uma nova representação, que restringe o espaço de busca priorizando por equações simples e de maior interpretabilidade, e apresentando uma menor volatilidade nos dados, com o objetivo de contornar os problemas da regressão simbólica por meios da programação genética canônica. Essa nova representação utiliza uma estrutura de dados nomeada Interação-Transformação (IT), que muda a forma como as expressões são construídas.

Junto à essa estrutura, foi introduzido um algoritmo, o *SymTree*, uma aplicação direta da IT para regressão simbólica. O *SymTree* realiza uma busca em largura em uma árvore onde a raiz é uma combinação linear das variáveis, e cada nó filho é uma expansão de seu nó pai (FRANÇA, 2018).

Os resultados mostraram que o algoritmo *SymTree* foi capaz de obter a expressão correta quando a função alvo pudesse ser descrita por meio da estrutura de dados (FRANÇA, 2018), e apresentar resultados competitivos quando a função alvo não pudesse ser descrita. Também se mostrou 100% preciso de acordo com a definição de precisão apresentada em (KORNS, 2011) nos testes realizados para equações da física e engenharia que podem ser representadas (ALDEIA; FRANCA, 2018), sendo uma prova de conceito da estrutura IT.

A estrutura IT descreve a aplicação de uma função de transformação sobre funções polinomiais das variáveis originais, e é utilizada como um bloco elementar para construção de equações, que são formadas a partir da combinação linear de várias estruturas IT.

Seja n o número de atributos do problema de regressão, $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ um ponto no domínio do problema, $\mathbf{k}$ um vetor de expoentes $\mathbf{k} \in \mathbb{Z}^n$ que possui um expoente para cada variável do problema, t uma função unária $t : \mathbb{R} \to \mathbb{R}$ (chamada de função de transformação), e seja $i(\mathbf{x}, \mathbf{k})$ uma função definida como o produto de cada

variável elevada ao expoente de $\mathbf{k}$ de mesmo índice (chamada de função de interação):

$$i(\mathbf{x}, \mathbf{k}) = \prod_{i=1}^{n} x_i^{k_i}. \tag{2}$$

Uma estrutura Interação-Transformação (IT) é uma tupla $(t, \mathbf{k})$ onde t é qualquer função matemática unária e $\mathbf{k}$ um vetor de dimensão n contendo expoentes aplicados às respectivas variáveis x_i que representa a força da interação entre as variáveis.

Dessa forma, uma estrutura IT representa uma função característica:

$$it(\mathbf{x}) = t(i(\mathbf{x}, \mathbf{k})). \tag{3}$$

Chamaremos de **termo IT** uma única tupla de Interação-Transformação, e **expressão IT** uma função composta por m termos combinados linearmente (equação 4).

$$Expr\ it(\mathbf{x}) = \sum_{j=1}^{m} w_j \cdot it_j(\mathbf{x}), \tag{4}$$

onde $\mathbf{w}$ representa um vetor de coeficientes (com elementos w_j), sendo cada elemento associado ao termo IT de mesmo índice.

Para ilustrar o uso dessa representação, tomemos como exemplo a expressão $f(x_1, x_2) = x_1 + \pi * cos(x_2)$ (apresentada na figura 9). São utilizados dois termos e um vetor de coeficientes $\mathbf{w}$:

$$\begin{aligned} it_1 &= (id, [1,0]), \\ it_2 &= (cos, [0,1]), \\ \mathbf{w} &= [1, \pi], \end{aligned} \tag{5}$$

onde *id* representa a função identidade.

A avaliação de it_1 e it_2 numa expressão linear de 2 termos IT resultaria na seguinte equação:

$$f(x_1, x_2) = 1 * id(x_1^1 * x_2^0) + \pi * cos(x_1^0 * x_2^1). \tag{6}$$

Dessa forma, representamos uma expressão que antes era construída com uma árvore sintática a partir da combinação linear de dois termos, onde toda não linearidade na expressão vem dos expoentes e da função de transformação.

Os escalares de cada termo são calculados utilizando modelos de regressão paramétricos, que ajustam apenas os coeficientes de funções de forma fixa. Em (ALDEIA; FRANCA, 2018), os autores utilizam o Método dos Mínimos Quadrados para ajuste desses escalares, um método de estimação de parâmetros que funciona minimizando a Soma Média Quadrática (*Mean Squared Error*) de uma função em relação a um conjunto conhecido de pontos. Com isso, os escalares são calculados por:

$$\hat{\mathbf{w}} = (\mathbf{I}^T\mathbf{I})^{-1}\mathbf{I}^T\mathbf{y}, \tag{7}$$

com $\mathbf{I} \in \mathbb{R}^{d\times m}$, onde d é a dimensão da base de dados e m o número de termos IT da expressão, em que cada elemento de $\mathbf{I}$ é a estrutura IT de índice i $(1 \leq i \leq m)$ avaliada para a amostra j da base $(1 \leq j \leq d)$: $\mathbf{I} = it_i(\mathbf{x}_j)$; e $\mathbf{y}$ o vetor composto pelos d valores da variável-alvo para toda a base.

A composição de funções por meio dessa estrutura permite a formação apenas de funções mais simples, contornando os problemas da regressão simbólica canônica. O objetivo em restringir o espaço de busca é fazer com que o algoritmo foque apenas no espaço de expressões simples, onde o conceito de simplicidade está relacionado com a capacidade de interpretação do resultado (FRANÇA, 2018). Mas note que esta restrição no espaço de busca não permite representar todas as equações possíveis - a equação 8 (resistência equivalente entre dois resistores paralelos) não pode ser expressada por meio de termos IT, pois não há combinação linear que resulte na equação. O quadro 1 apresenta alguns casos factíveis e infactíveis. Vale observar que, dentre expressões não representáveis, encontram-se, além de expressões simples e únicas, expressões com informação redundantes e com baixa interpretabilidade.

$$R_{eq} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}. \tag{8}$$

Um fator associado com a interpretabilidade das expressões que o autor em (SMITS; KOTANCHEK, 2005) destaca é a identificação de *metavariáveis*, termo descrito nas palavras dele por **combinação** ou **transformação** de variáveis. A representação IT naturalmente induz as expressões a incorporarem tanto combinações quanto trans-

Quadro 1 – Algumas funções que podem e não podem ser expressadas utilizando essa representação.

Expressões representáveis	Expressões não representáveis
$x_1 + \pi * cos(x_2)$	$\sqrt{x_1 * x_1} + tanh(tanh(tanh(x_2)))$
$x_1 + x_2 + x_1 * x_2$	$\frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$
$10 * log(\frac{I}{I_0}) + 5.0$	$\frac{sen(x_1 * x_2)}{x_3}$
$a^2 + a * b + b^2$	$\sqrt{a^2 + b^2}$

formações das variáveis, de forma que além de favorecer expressões mais simples, colateralmente contribui para a interpretabilidade do resultado em vários aspectos, como combinações lineares, interações e transformações evidentes, e restringindo o encadeamento consecutivo de várias funções matemáticas.

4.1 REPRESENTAÇÃO COMPUTACIONAL

A estrutura IT pode ser representada computacionalmente de forma simples e eficiente utilizando estruturas de dados comuns em todas as linguagens de programação atuais - listas. Como consequência, se aproxima da ideia dos algoritmos evolutivos que manipulam cadeias de caracteres, já que as expressões podem ser representadas por listas. Dessa forma, cabe aos operadores evolutivos modificar os termos e funções $(func, terms)$ para criar expressões; e ao método de regressão paramétrica utilizado ajustar os pesos e o valor do intercepto.

Sendo uma expressão IT composta por um ou mais termos IT, onde um único termo é uma tupla contendo uma função de transformação e um vetor de expoentes para ser aplicado às variáveis do problema, e como há a etapa adicional de realizar um ajuste de coeficientes da expressão IT, também é obtido um vetor de coeficientes de tamanho igual à quantidade de termos IT utilizados para compor a equação, e um

Quadro 2 – Representação computacional de expressões representáveis por meios da estrutura IT.

Expressão	Representação computacional

$x_1 + \pi * cos(x_2)$	`expressao1 {`
	`  terms = [[1, 0], [0, 1]]`
	`  funcs = [id, cos]`
	`  weights = [1.0, 3.1415]`
	`  intercept = 0.0`
	`}`

$x_1 + x_2 + x_1 * x_2$	`expressao2 {`
	`  terms = [[1, 0], [0, 1], [1, 1]]`
	`  funcs = [id, id, id]`
	`  weights = [1.0, 1.0, 1.0]`
	`  intercept = 0.0`
	`}`

$10 * log(\frac{I}{I_0}) + 5.0$	`expressao3 {`
	`  terms = [[1, -1]]`
	`  funcs = [log]`
	`  weights = [10.0]`
	`  intercept = 5.0`
	`}`

$a^2 + a * b + b^2$	`expressao4 {`
	`  terms = [[2, 0], [1, 1], [0, 2]]`
	`  funcs = [id, id, id]`
	`  weights = [1.0, 1.0, 1.0]`
	`  intercept = 0.0`
	`}`

valor real representando o intercepto da regressão paramétrica. Dessa forma, um único termo IT pode ser representado pela tupla $it = (func, terms)$ e uma expressão IT por $Expr\ it = ([it], weights, intercept)$, onde $[it]$ é uma lista de termos IT e *weights* é uma lista dos coeficientes respectivos à cada IT. O quadro 2 apresenta alguns exemplos de uma maneira adequada de representar uma expressão IT computacionalmente, onde cada expressão é um grupo de 4 estruturas:

- **terms**: uma lista de listas, que representam os vetores $\mathbf{k}$;

- **funcs**: uma lista de funções unárias;
- **weights**: uma lista de coeficientes;
- **intercept**: um valor real que representa o intercepto.

Note que as três primeiras estruturas são listas que devem ser do mesmo tamanho, onde cada elemento de índice i em cada uma dessas listas representa um termo IT e seu coeficiente.

4.2 PROPRIEDADES DA ESTRUTURA

Apesar da liberdade de escolha de qualquer função matemática unária como função de transformação, tanto como qualquer expoente, se o propósito for obter uma função contínua e diferenciável utilizando-se termos IT, devemos satisfazer duas condições:

- todo expoente deve ser maior ou igual a zero;
- toda função de transformação deve ser contínua.

Pode ser desejável em algumas situações obter um resultado diferenciável, como na busca por funções de ativação de redes neurais ou quando há necessidade de medir a taxa de variação do fenômeno estudado, encontrar o máximo e mínimo das funções, entre outros.

Definição 1 (Continuidade)**.** *Para que uma função f seja contínua num ponto a, devemos satisfazer as seguintes condições:*

- *Existe $f(a)$;*
- *Existe $\lim_{x \to a} f(x)$;*
- $\lim_{x \to a} f(x) = f(a)$*;*

Propriedade 1 (Operações matemáticas básicas)**.** *Sejam $g(x)$ e $h(x)$ funções contínuas em a. Então são contínuas em a também:*

- $g(x) + h(x)$*;*
- $g(x) - h(x)$*;*
- $g(x) * h(x)$*;*
- $\frac{g(x)}{h(x)}$*, desde que $h(x) \neq 0$;*

  Note que para que $\frac{g(x)}{h(x)}$ seja contínua, deve satisfazer-se a condição $h(x) \neq 0$.

Definido o conceito de continuidade e como ela se conserva nas operações matemáticas básicas, será apresentado o comportamento da continuidade para funções matemáticas e trigonométricas comumente utilizadas.

Propriedade 2 (Funções matemáticas)**.** *As seguintes funções apresentam continuidade em* $x \in \mathbb{R}$*:*

- *Seja* $g(x)$ *uma função polinomial,* $g(x)$ *é contínua para todo* $x \in \mathbb{R}$*;*
- *As funções* $g(x) = sen(x)$ *e* $h(x) = cos(x)$ *são contínuas para todo* $x \in \mathbb{R}$*;*
- *A função exponencial* $g(x) = x^a$ *(para* $a > 0$ *e* $a \neq 1$*) é contínua para todo* $x \in \mathbb{R}$*.*

Propriedade 3 (Funções matemáticas (cont.))**.** *Se existe uma função* $g(x)$ *definida e contínua num intervalo* I *e* $Im(g) = J$*, e* g *admite uma função inversa, então* $g^{-1} : J \to I$ *é contínua em todo o intervalo* J*.*

Como a inversa da exponencial é a função logarítmica, então a função logarítmica é contínua.

Logo, a função logarítmica $h(x) = log_a x$ *(para* $a > 0$ *e* $a \neq 1$*) é contínua.*

Propriedade 4 (Composição de funções)**.** *Se* g *é contínua em* a *e* h *é contínua em* $f(a)$*, então a função composta* $g \circ f$ *é contínua no ponto* a*.*

Quando for utilizada a representação IT para a tarefa de regressão, muito provavelmente as funções matemáticas e trigonométricas acima farão parte do conjunto de funções de transformação. Como a representação IT faz uma composição de funções, aplicando a transformação sobre a interação das variáveis, é importante avaliar a continuidade de funções compostas.

Apresentadas as propriedades, uma pequena análise de como conservar a continuidade utilizando a representação IT será feita à seguir.

Proposição 1. *Um termo IT é contínuo e derivável, desde que* $\forall k_i \in \mathbf{k}, k_i > 0$*.*

Seja it *um termo IT dado por* $(t, \mathbf{k})$*, sendo* t *uma função* $t : \mathbb{R} \to \mathbb{R}$ *e* $\mathbf{k} \in \mathbb{Z}^n$ *um conjunto de expoentes. Pela definição da IT,* it *será calculado pela composição* $t \circ i$ *(equação 2).*

Pela propriedade 4, $t \circ i$ *será contínua se* t *for contínua em* $i(\mathbf{x})$*,* $i(\mathbf{x}) : \mathbb{R}^{dim\ \mathbf{x}} \to \mathbb{R}$*, onde* $dim\ \mathbf{x}$ *é a dimensão do vetor de variáveis do problema* $\mathbf{x}$*.*

Vejamos que $i(\mathbf{x})$ é contínua. Pela propriedade 1 temos que o produto entre duas funções contínuas é contínuo. Para cada variável $x_i^{k_i}, x_i \in \mathbf{x}$, temos que $x_i^{k_i}$ é um polinômio (caso k_i seja ≥ 0), que pela propriedade 2 é contínuo.

Logo, uma estrutura IT é contínua quando $\forall k_i \in \mathbf{k}, k_i > 0$ e, portanto, derivável.

Proposição 2. *Uma combinação linear de termos IT contínuos é também uma função contínua.*

Seja f uma expressão linear composta por duas estruturas IT:

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = w_1 * t(x_1^{k_{1,1}} * \ldots * x_n^{k_{1,n}}) + w_2 * h(x_1^{k_{2,1}} * \ldots * x_n^{k_{2,n}}) \quad (9)$$

Visto que as funções representadas por expressões IT são contínuas, e a adição de duas funções é contínua (pela propriedade 1), e uma expressão IT com vários termos é o encadeamento consecutivo de somas, se cada termo IT for contínuo (ou seja, o conjunto de funções unárias utilizado para composição das expressões contenha exclusivamente funções contínuas e $\forall k_{i,j} > 0$), a função f denotada pela representação IT será contínua.

Nesse capítulo foi apresentada a estrutura IT que será foco do trabalho, tendo algoritmos que realizam regressão simbólica por meio dessa representação. Ela apresenta como vantagens sua simplicidade de representação computacional, restrição do espaço para equações simples e uma menor fragilidade diante de modificações de seus parâmetros; ao custo de um espaço de busca restringido, que pode ser uma limitação para alguns tipos de problemas onde as variáveis apresentam uma relação simples que está fora do espaço de busca.

As expressões são compostas de combinações lineares dessa estrutura, onde a não linearidade é introduzida pela função de transformação.

5 ALGORITMOS EVOLUTIVOS IMPLEMENTADOS UTILIZANDO A ESTRUTURA INTERAÇÃO-TRANSFORMAÇÃO

Para a implementação de algoritmos evolutivos baseados na estrutura IT é necessário ter métodos que manipulem a representação computacional da estrutura para simular o processo evolutivo. Para isso, foram pensados operadores que podem ser combinados de diferentes maneiras para permitir uma construção mais modular de algoritmos evolutivos. Neste capítulo, os operadores criados serão apresentados e, em seguida, os algoritmos desenvolvidos utilizando-os serão ilustrados e discutidos.

5.1 OPERADORES

Sendo um termo IT representado pela tupla $it = (term, funs)$ (como descrito na seção 4.1), os aqui chamados **operadores** são métodos que manipulam populações de expressões IT no nível estrutural de cada indivíduo da população, modificando suas listas de expoentes ou lista de funções de transformação, aplicando modificações com o objetivo de mimetizar alguns fenômenos evolutivos sobre a população de soluções.

Como tanto os coeficientes quanto o intercepto são parâmetros livres da expressão IT e são ajustados durante a criação de uma expressão tendo seus termos IT levados em consideração, são considerados uma otimização individual de cada termo - esse comportamento é chamado de aprendizado individual em algoritmos evolutivos (um equivalente ao fenótipo, que resulta da expressão das informações genéticas de um indivíduo) - e não são herdados pelas gerações futuras, portanto, os operadores atuam apenas sobre suas listas de expoentes ou lista de funções de transformação (equivalente ao genótipo).

Como propósito de criar alguns variantes de algoritmos evolutivos, foram feitos 4 operadores evolutivos: *(i)* operador de origem, *(ii)* operador de mutação, *(iii)* operador de *crossover* e *(iv)* operador de seleção.

5.1.1 OPERADOR DE ORIGEM

Para a criação de expressões no estágio inicial da evolução, é necessário um procedimento bem definido. O operador de origem trata de preencher uma população

inicialmente vazia com soluções criadas aleatoriamente, utilizando apenas expressões de baixa complexidade - tendo um número aleatório de termos dentro do intervalo $[1, 4]$ para cada solução, onde o expoente para cada variável do problema vem do intervalo $[0, 4]$. Estes limites no tamanho das expressões e nos valores dos expoentes tem como objetivo criar uma população inicial composta de soluções simples. Nesta etapa, todos os valores são sorteados com igual probabilidade.

O operador de origem faz o tratamento adequado para evitar a duplicata de termos, pois essa redundância aumenta o tamanho da expressão sem agregar novas características - ao se deparar com um termo repetido, caso a expressão não seja nula, ele é descartado. Também é feito o tratamento para não existir termos nulos $(\mathbf{k} = \mathbf{0}^n)$ ou uma expressão vazia. Caso uma solução seja criada aleatoriamente com um único termo onde todos os expoentes são 0 o algoritmo persistentemente irá criar um novo termo único, com os expoentes variando de 0 a 3. Isso evita soluções triviais na população. Esse operador é ilustrado pelo algoritmo 2.

Algoritmo 2: Operador de origem (método *oporigem*).

Entrada : pop_len: Tamanho da população
n_vars: Número de variáveis explicatórias
$funcs$: Conjunto de possíveis funções de transformação
Saída : P: População com soluções aleatórias

```
P ← [ ];
repita
    n_terms ← random([1, ..., 4]);
    terms ← [[random([0, ..., 4]) para _ ∈ n_vars] para _ ∈ n_terms];
    funcs ← [random(funcs) para _ ∈ n_terms];
    para (term, func) ∈ zip(terms, funcs) faça
        se (func = 0^n) ou se (func ∈ funcs − [func]) então
            terms ← terms - [term];
            funcs ← funcs - [func];
        fim
    fim
    se len(funcs) = len(terms) = 0 então
        P ← P + [(terms, funcs)];
    fim
até len(P) seja igual a pop_len;
retorna P;
```

Onde a função *random()* recebe uma lista de valores possíveis e seleciona um elemento dela com probabilidade uniforme, a função *len()* recebe um vetor e retorna seu número de elementos, e o símbolo "_" representa uma variável sem importância.

Como todos os sorteios aleatórios não levam em consideração nenhum fator à respeito de quantos elementos já estão na população, é esperado uma distribuição uniforme entre as possíveis combinações de número de termos, termos e funções possíveis, partindo assim de uma população com grande diversidade.

5.1.2 OPERADOR DE MUTAÇÃO

O operador de mutação opera sobre a população inteira, percorrendo cada uma das soluções e aplicando uma mutação disponível selecionada aleatoriamente com probabilidade uniforme (ilustrado pelo algoritmo 3). Cada mutação modifica algum aspecto da expressão IT, operando sobre uma cópia da expressão original. As mutações implementadas estão descritas abaixo:

Drop Term Mutation (Drop) mutação que remove um termo IT da expressão, selecionado aleatoriamente com mesma chance de seleção para todos os termos (uma modificação opcional seria dar a maior chance ao termo de menor coeficiente, pois esse representa o termo de menos significância para a regressão linear, onde são obtidos os coeficientes). Essa mutação só é aplicada se existir um número mínimo de termos na expressão atual, podendo ser controlado o tamanho mínimo da expressão. Um menor tamanho implica na existência de soluções mais simples.

Add Term Mutation (Add) Essa mutação adiciona um novo termo aleatório para a expressão. Da mesma forma que a anterior, há um limite de tamanho da solução, ajustável pelo usuário de acordo com a complexidade permitida para a busca. Esse novo termo é feito seguindo a estratégia adotada para o operador de origem.

Replace Interaction Mutation (Term) Essa mutação substitui um único valor do vetor de expoentes de um termo por um novo, sendo esse termo sorteado aleatoriamente da solução. Essa nova interação terá um único expoente modificado, substituindo o termo antigo. Caso o termo novo seja idêntico a um termo já existente, ele é descartado junto com o antigo.

Replace Transformation Mutation (Func) essa mutação modifica a função de transformação de um termo sorteado aleatoriamente para uma nova função

sorteada dentre as opções. Caso o termo novo seja idêntico a um já pertencente a equação, ele é descartado junto com o antigo.

Positive interaction mutation (Interp) Seleciona 2 termos aleatórios da expressão e realiza uma soma *element-wise* dos expoentes dos 2 termos sorteados. Então, utilizando a função de transformação do primeiro termo sorteado e o conjunto de expoentes resultante, adiciona esse termo à expressão, sem descartar nenhum anterior.

Negative Interaction mutation (Intern) Funciona de forma similar ao anterior, mas fazendo a subtração *element-wise*. Esse é o único operador que gera expoentes negativos.

Algoritmo 3: Operador de mutação (método *opmutacao*).

Entrada : pop: População
min_len: Tamanho mínimo da expressão
max_len: Tamanho máximo da expressão
Saída : $Childs$: População mutacionada

$Childs \leftarrow$ [ ];
$Mutations \leftarrow$ [Replace_Interaction, Replace_Transformation];
para $Expr\ it \in pop$ **faça**
  se *len*$(Expr\ it) > min\_len$ **então**
    $Mutations \leftarrow Mutations +$ [Drop_Term];
  fim
  se *len*$(Expr\ it) < max\_len$ **então**
    $Mutations \leftarrow Mutations +$ [Add_Term, Negative_Interaction, Positive_Interaction];
  fim
  $mutation \leftarrow$ random$(Mutations)$;
  $Childs \leftarrow Childs +$ [$mutation(Expr\ it)$];
fim
retorna $Childs$;

Dados os tipos de mutação apresentados, podemos categorizá-los em 4 grupos:

- **Expansores de expressão**: *Add*, *Interp*, *Intern*;
- **Redutores de expressão**: *Drop*;
- **Modificadores de expoentes**: *Term*;
- **Modificadores de termos**: *Func*.

O agrupamento das diferentes mutações em grupo permite um tratamento no controle do tamanho das expressões, de forma que nunca ultrapassem os limites definidos para a expressão final.

5.1.3 OPERADOR DE SELEÇÃO

O operador de seleção impõe sobre a população uma pressão seletiva onde as soluções que apresentam uma melhor aptidão tem maiores chances de serem selecionadas, sendo a aptidão definida pela métrica para qualificar as funções. Ela deve ocorrer de forma que favoreça os mais adaptados sem excluir os menos aptos completamente.

Para esse operador, foi utilizado o método de seleção por torneio: o algoritmo recebe uma população de tamanho n e um tamanho alvo para uma nova população m, e então iterativamente duas soluções aleatórias da população fazem um confronto onde sua aptidão é comparada, sendo vencedora aquela com melhor aptidão. Cada confronto tem sua solução vencedora adicionada à nova população, sem que ela seja retirada dos candidatos para os próximos confrontos, podendo acabar se repetindo. É esperado que esse comportamento favoreça melhores soluções, tornando-as mais presentes na população, mas não faça com que a convergência ocorra muito rápido, tendo em vista que a chance de serem selecionadas para confronto é a mesma para todas as soluções. Este operador é sumarizado pelo algoritmo 4.

Algoritmo 4: Operador de seleção (método *opselecao*).

Entrada : pop: População
$newpop\_len$: Tamanho da população selecionada
$tournament$: Função de torneio
Saída : $Selected$: População selecionada pelo torneio

$Selected \leftarrow [\,]$;
repita
  $winner \leftarrow tournament$(pop);
  $Selected \leftarrow Selected + [winner]$;
até *len(Selected) seja igual a* $newpop\_len$;
retorna $Selected$;

A função de torneio é o que determina qual solução será considerada a vencedora, podendo receber um ou vários indivíduos simultaneamente, e comparando

um ou vários critérios utilizando alguma heurística para determinar a aptidão de cada solução.

5.1.4 OPERADOR DE *CROSSOVER*

O operador de *crossover* simula a troca de informação genética da reprodução sexuada, requerendo duas funções para atuar. Tendo a população inteira como entrada, as soluções são selecionadas duas a duas através do operador de seleção para o processo de *crossover*.

Algoritmo 5: Operador de crossover (método *opcrossover*).

Entrada : pop: População
$newpop\_len$: Tamanho da população selecionada
$tournament$: Função de torneio
$cross\_rate$: Chance de herdar cada termo
Saída : $Childs$: População "filha"

```
Childs ← [ ];
repita
    (terms1, funcs1) ← tournament(pop);
    (terms2, funcs2) ← tournament(pop);
    picked_terms ← [ ];
    picked_funcs ← [ ];
    para term, func ∈ zip(terms1 + terms2, funcs1 + funcs2) faça
        se random() < cross_rate então
            picked_terms ← picked_terms + [term];
            picked_funcs ← picked_funcs + [func];
        fim
    fim
    Childs ← Childs + [(picked_terms, picked_funcs)];
até len(Childs) seja igual a newpop_len;
retorna Childs;
```

As duas soluções selecionadas serão percorridas termo a termo, onde cada vez que um termo é visitado é feito um sorteio de um número aleatório entre $[0, 1]$. Caso o número sorteado seja menor que o $cross\_rate$ (que deve ser um valor entre $[0, 1]$ indicando a chance de cada termo dos pais ocorrer na solução filha), esse termo será selecionado para compor a expressão filha. Note que aqui não há necessariamente uma

configuração de 50% do material de cada progenitor, e a solução gerada pode ser tão grande quanto os pais ou nula (no segundo caso, uma solução aleatória é retornada). Este processo é ilustrado pelo algoritmo 5.

5.2 ALGORITMOS IMPLEMENTADOS

Com base nos operadores apresentados, foram criados 3 algoritmos distintos, resultantes das possíveis combinações entre o **operador de mutação** e o **operador de crossover**. Os operadores de criação e seleção pertencem à qualquer implementação pois são essenciais para a criação, manutenção e convergência da população, sendo de interesse explorar apenas os operadores que dizem respeito às modificações nas estruturas e que influenciam tanto na convergência quanto na qualidade da solução.

Dessa forma, os três algoritmos criados são:

- **IT-MUT**: Algoritmo que utiliza apenas a mutação;
- **IT-CX**: Algoritmo evolutivo que utiliza apenas o *crossover*;
- **ITEA**: Algoritmo que utiliza a combinação da mutação e do *crossover*.

Para todos os algoritmos, a função de torneio sorteia duas soluções aleatórias da população e retorna aquela que minimiza a raiz quadrada do erro médio quadrático (*Root Mean Square Error*, RMSE) das soluções - sendo esta a função de *fitness*. O coeficiente de cada termo (sua aprendizagem individual) só é calculado durante a comparação entre soluções pois é quando faz-se necessário, então a função de torneio utiliza a base de dados, representada por $(\mathbf{X}, \mathbf{y})$ (variáveis explanatórias e variável alvo), e um método de regressão linear para ajuste dos coeficientes.

O algoritmo 6 ilustra a função de torneio. O método de regressão linear recebe uma matriz onde cada coluna representa o termo IT de mesmo índice avaliado para a matriz de $\mathbf{X}$ (como descrito no capítulo 4) e o vetor de variáveis alvo $\mathbf{y}$, para que os coeficientes ajustados pela regressão linear correspondam aos termos, e não as variáveis originais do problema. A função **min_RMSE** recebe duas expressões IT e retorna aquela que minimiza o RMSE, onde $\cdot$ representa o produto escalar dos coeficientes pelos termos.

Em outras palavras, a função de torneio seleciona duas soluções, realiza o aprendizado individual e compara-as, retornando a que apresenta melhor aptidão.

Algoritmo 6: Função de torneio (ftournament).

Entrada: pop: População de soluções
Saída : Expressão IT de menor RMSE

$\mathbf{X}$: Matriz com variáveis explicatórias;
$\mathbf{y}$: Matriz com variáveis alvo;

$linear_r$: Método de regressão linear para ajuste dos coeficientes;

$Expr\ it1 \leftarrow$ random(pop);
$Expr\ it2 \leftarrow$ random(pop);

$coefs1 \leftarrow linear_r$(Expr it1($\mathbf{X}$), $\mathbf{y}$);
$coefs2 \leftarrow linear_r$(Expr it2($\mathbf{X}$), $\mathbf{y}$);
retorna min_RMSE($coefs1 \cdot Expr\ it1, coefs2 \cdot Expr\ it2$);

5.2.1 IT-MUT

O primeiro algoritmo desenvolvido (algoritmo 7) utiliza a seleção como primeira etapa na criação de uma nova geração e, com os termos selecionados, aplica o operador de mutação sobre elas, sempre gerando uma variação diferente em cada solução.

Algoritmo 7: Algoritmo IT-MUT.

Entrada: pop_len: Tamanho da população
$gens$: Número de gerações
n_vars: Número de variáveis explicatórias
$funcs$: Conjunto de funções para busca
$ftournament$: Função de torneio
Saída : Expressão IT de menor RMSE

$min\_len = 3$;
$max\_len = 10$;
P $\leftarrow$ oporigem($pop\_len, n\_vars, funcs$);
para g *Gerações* **faça**
  P' $\leftarrow$ opmutacao(P, min_len, max_len);
  P $\leftarrow$ opselecao(P + P', len(P'), $ftournament$);
fim
retorna min_RMSE(P);

Como o operador de mutação funciona modificando a população passada como parâmetro, o resultado será uma população de mesmo tamanho; e o operador

de seleção retorna uma população de mesmo tamanho que a população mutacionada - dessa forma, a cada passo, a população sempre permanece com tamanho constante.

5.2.2 IT-CX

A segunda variação, **IT-CX** (algoritmo 8), funciona utilizando apenas o *crossover* antes de realizar a seleção, pois o próprio operador de *crossover* já utiliza a função de torneio para a criação de novas soluções.

Algoritmo 8: Algoritmo IT-CX.

Entrada: pop_len: Tamanho da população
$gens$: Número de gerações
n_vars: Número de variáveis explicatórias
$funcs$: Conjunto de funções para busca
$cross\_rate$: Taxa de *crossover*
$ftournament$: Função de torneio
Saída : Expressão IT de menor RMSE

$min\_len = 3$;
$max\_len = 10$;
P $\leftarrow$ oporigem($pop\_len, n\_vars, funcs$);
para g *Gerações* **faça**
  P' $\leftarrow$ opcrossover(P, $pop\_len, ftournament, cross\_rate$);
  P $\leftarrow$ opselecao(P + P', len(P'), $ftournament$);
fim
retorna min_RMSE(P);

Neste algoritmo a restrição do tamanho da expressão não é considerada (pois o controle do tamanho ocorre apenas na mutação), de forma que uma taxa de *crossover* muito grande possa levar a população à soluções sempre mais complexas, uma vez que é essa taxa que determina a quantidade de termos herdados dos pais em cada solução.

5.2.3 ITEA

O terceiro algoritmo, *ITEA* (algoritmo 9) utiliza a combinação dos dois operadores que modificam as soluções. Primeiro, é feito o *crossover*, após isso todas as soluções geradas passam pelo operador de mutação, onde a chance de ocorrência de mutações é sempre garantida.

Algoritmo 9: Algoritmo ITEA.

Entrada: pop_len: Tamanho da população
$gens$: Número de gerações
n_vars: Número de variáveis explicatórias
$funcs$: Conjunto de funções para busca
$cross\_rate$: Taxa de *crossover*
$ftournament$: Função de torneio
Saída : Expressão IT de menor RMSE

$min\_len = 3$;
$max\_len = 10$;
P $\leftarrow$ oporigem($pop\_len, n\_vars, funcs$);
para g *Gerações* **faça**
  P' $\leftarrow$ opcrossover(P, $pop\_len, ftournament, cross\_rate$);
  P' $\leftarrow$ opmutacao(P, min_len, max_len);
  P $\leftarrow$ opselecao(P + P', len(P'), $ftournament$);
fim
retorna min_RMSE(P);

Da mesma forma que o algoritmo anterior, o operador de *crossover* não restringe o tamanho das expressões dentro de um número de termos IT, mas o operador de mutação aplica essa restrição, e ocorre após o *crossover*, resultando em um controle parcial do tamanho da complexidade das soluções. A taxa de *crossover* é o fator determinante para o tamanho final das soluções, onde um valor alto leva à expressões com um número grande de termos IT, pois acaba selecionando a maioria dos termos dos dois pais.

Nesse capítulo vimos como foram implementados os algoritmos, que podem ser reproduzidos seguindo o que foi apresentado até este momento para o trabalho. Esquematizados os operadores e ilustrados como compõem os algoritmos, e criada uma função de torneio, o restante do trabalho se dedica em avaliar rigorosamente os resultados desses algoritmos propostos para então chegar à uma conclusão sobre o uso da representação Interação-Transformação para regressão simbólica por meio de algoritmos evolutivos.

6 METODOLOGIA

Para avaliar os algoritmos desenvolvidos, serão realizados testes com bases de dados conhecidas, comumente empregadas em testes de regressão e *machine learning*.

O *Mean Absolute Error* (MAE) e o *Root Mean Absolute Error* (RMSE) são duas métricas utilizadas para quantificar a precisão dos algoritmos. O MAE em particular desconsidera a direção dos erros, pois mede apenas a média dos erros absolutos. Trata-se de uma métrica linear, de forma que todos os erros são ponderados com o mesmo peso na média, sendo preferível quando o erro pode ser medido linearmente sem prejudicar a avaliação do algoritmo.

$$MAE = \frac{1}{d}\sum_{i=1}^{d} \left| y_i - f\left(\mathbf{x}_i\right) \right|, \tag{10}$$

sendo d o tamanho da série de medidas amostrais, y_i o valor alvo, e $f\left(\mathbf{x}_i\right)$ a solução avaliada para as variáveis explicatórias de índice i, $\mathbf{x} \in \mathbf{X}$.

Enquanto isso, o RMSE eleva os erros ao quadrado antes de utilizá-los no cálculo da média, dessa forma pontos que apresentam erros grandes (maiores que 1) são amplificados, enquanto erros pequenos (menores que 1) são reduzidos.

$$RMSE = \sqrt{\frac{1}{d}\sum_{i=1}^{d} \left(y_i - f\left(\mathbf{x}_i\right)\right)^2}. \tag{11}$$

O RMSE estimula modelos em que todos os erros sejam pequenos, sendo preferível quando não desejamos erros grandes.

Por fim, também será calculado o valor do *Normalized Mean Squared Error* (NMSE):

$$NMSE = \frac{1}{d}\sum_{i=1}^{d} \frac{(f(\mathbf{x}_i) - y_i)^2}{\overline{f(\mathbf{X})}\overline{\mathbf{y}}}, \tag{12}$$

onde $\overline{f(\mathbf{X})}$ representa a média dos valores preditos e $\overline{\mathbf{y}}$ a média dos valores esperados. Essa terceira métrica apresenta um valor baixo quando o erro é baixo em magnitude em

toda a base de teste; e um valor alto quando o erro tem magnitude alta em algumas ou em todas as bases de teste.

Para os testes, o método de regressão para ajuste dos coeficientes da expressão utilizado será o *linearRegression* da biblioteca *scikit-learn*, que implementa o método dos mínimos quadrados para a tarefa. Para a seleção de torneio foi definida uma função que compara soluções 2 a 2 e define como a vencedora aquela que apresentar um menor RMSE, ou seja, aquela que mais minimiza a função de erro - o estudo de heurísticas de seleção está fora do escopo deste trabalho, mas existem várias pesquisas na definição de tais e isso é um aspecto que pode ser explorado futuramente.

Os algoritmos possuem vários *hiper-parâmetros*, isto é, parâmetros que não são ajustados pelo modelo e que influenciam na busca e no resultado.

Quadro 3 – Algoritmos implementados e seus hiper-parâmetros. Em destaque estão os parâmetros específicos de cada algoritmo.

Algoritmo	Hiper-parâmetros
IT-MUT	Tamanho da população, Número de gerações, Conjunto de funções, **Mutações**
IT-CX	Tamanho da população, Número de gerações, Conjunto de funções, **Taxa de crossover**
ITEA	Tamanho da população, Número de gerações, Conjunto de funções, **Mutações**, **Taxa de crossover**

Para encontrar o melhor grupo de mutações, será testado cada grupo isoladamente e, após o teste de todos os grupos individuais, os dois grupos que apresentarem valores de mediana inferiores que os outros na maioria dos casos serão combinados e testados novamente. O processo será repetido até que todos os grupos tenham sido combinados.

Para a taxa de *crossover*, será feito o teste para os valores 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 e 0.7, onde essa taxa indica a chance para cada termo de cada expressão IT de ser selecionado para gerar o descendente.

Obtidos o melhor grupo de mutação e a melhor taxa de *crossover*, estes serão os parâmetros utilizados nos modelos para análise de seus desempenhos.

Os algoritmos também serão avaliados em termos de simplicidade dos resultados obtidos, tendo o tamanho médio das expressões convertidas para árvores sintáticas reportado, onde o tamanho será determinado pela quantidade de nós que a árvore possuir.

6.1 BENCHMARK

Serão utilizadas bases de dados comumente aplicadas para teste de algoritmos de regressão (tabela 1) para compor o *benchmark*. Todas as bases são dados coletados do mundo real, divididas em 5 partições pelos autores de (MARTINS et al., 2018).

Tabela 1 – Bases de dados com suas respectivas dimensões.

Nome da base	# Partições	# Atributos	# Amostras (treino/validação)
Airfoil	5	6	1202/301
Concrete	5	9	824/206
Energy Cooling	5	9	614/154
Energy Heating	5	9	614/154
Tower Data	5	26	3999/1000
Wine Red	5	12	1279/319
Wine White	5	12	3919/979
Yatch	5	7	246/62

Para chegar nas bases utilizadas, cada base originalmente é dividida em 5 partições de mesmo tamanho (cada partição é denominada *fold*) e então, para cada *fold* k toma-se a partição k como a partição reservada para validação (chamada de *fold-out*), enquanto as restantes são utilizadas para compor a base de treino (como ilustrado na figura 11), sendo que cada *fold* k utilizado como validação representa um

experimento. Nos dados fornecidos a divisão é feita reservando 80% para treino. Os modelos então serão qualificados pelos resultados obtidos no *fold-out*.

Para cada base de treino, o algoritmo passa pelo seu processo de aprendizagem, onde realiza o ajuste dos parâmetros do modelo para tentar encontrar a função f, enquanto a base de validação é exclusivamente reservada para avaliar como o modelo se comporta para dados diferentes dos usados nos treinos. Para obter o resultado final do desempenho, é calculada a média de todos os valores obtidos para cada base de validação, técnica chamada de *cross-validation*. Com os resultados obtidos utilizando o *cross-validation* para todas as bases, é possível analisar o desempenho de cada conjunto de hiper-parâmetros dos modelos, servindo de critério para definir as melhores configurações de hiper-parâmetros.

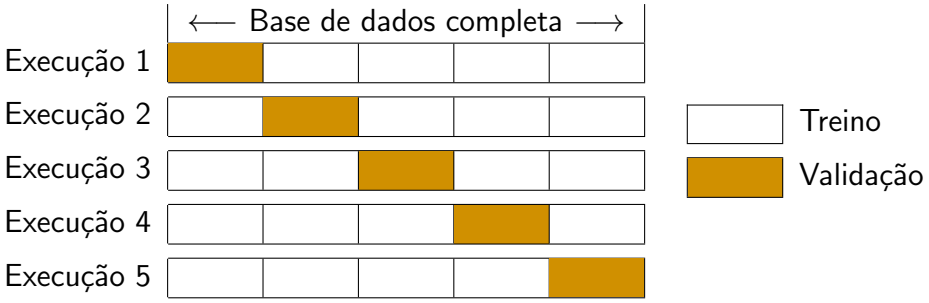


Figura 11 – Ilustração do processo de *cross-validation*.

Após dividir a base em 5 *folds*, cada um dos *folds* é tratado em uma execução como a partição para teste, e os outros são combinados para compor o grupo de treino. Esse processo resulta em k (no caso 5) estimativas de erro, de forma que a estimativa do k-*fold* CV é dada pela média obtida em todas essas execuções:

$$CV_k = \frac{1}{k} \sum_{i=1}^{k} ERRO_i. \tag{13}$$

Obtendo o valor CV para cada configuração de hiper-parâmetro, podemos determinar a melhor configuração como aquela que apresenta a menor média de erro (ou seja, minimiza o erro).

Cada algoritmo desenvolvido neste projeto passará pelo processo de treinamento e validação com cada base n vezes, tendo os valores das métricas NMSE, MAE e RMSE reportados para a base de teste em cada uma das execuções. Para obter o

valor de n, será feito um teste inicial considerando $n = 10$, indicando que cada base deve ser utilizada e ter seus valores reportados 10 vezes (como temos 5 *folds*, cada *fold* será executado 2 vezes). Após isso, o histograma da distribuição será analisado e, se apresentar um comportamento próximo de uma distribuição normal para todas as bases, o valor de n mantém-se o utilizado e valerá para todos os outros modelos utilizados para comparação. Caso contrário, n será incrementado em 10 e o processo será repetido, até que se obtenha uma distribuição próxima de uma distribuição normal.

6.2 COMPARAÇÃO COM OUTRAS TÉCNICAS

Por fim, algoritmos de regressão populares utilizados atualmente serão implementados para comparação com os resultados obtidos para os melhores hiper-parâmetros, submetendo os algoritmos de comparação a um processo de *grid-search* (uma busca exaustiva por todas as combinações possíveis de hiper-parâmetros) para ajustar seus hiper-parâmetros, otimizando seu desempenho para a comparação, sendo possível avaliar se os algoritmos propostos neste projeto apresentam um resultado competitivo se comparados com os modelos de regressão utilizados atualmente. Os métodos foram introduzidos na sessão 3.5, e seus hiper-parâmetros e valores de busca estão descritos no quadro 4. Todos os métodos possuem implementação disponibilizada na linguagem *Pyhton*, de forma que os métodos são executados na mesma linguagem e no mesmo ambiente, para as mesmas bases de dados.

Para todos os algoritmos comparados, os dados foram pré processados utilizando a normalização *z-score*, método para centralizar os dados, fazendo com que cada coluna da matriz de amostras $\mathbf{X}$ tenha média zero e desvio padrão 1. A matriz de variáveis explicatórias resultante $\mathbf{Z}$ tem mesmas dimensões, e cada coluna da matriz (z_j) é o resultado da equação 14.

$$\mathbf{z_j} = \frac{x_j - \mu_j}{\sigma_j}, \tag{14}$$

onde x_j corresponde à coluna original de $\mathbf{X}$, μ_j à média de x_j, e σ_j ao desvio padrão de x_j.

Para normalizar os dados de treino é feito o cálculo de cada μ_j e σ_j, e para a normalização dos dados de teste são utilizados os valores obtidos para os dados de treino. A normalização nunca deve ser feita antes da divisão em *folds*, para que os

Quadro 4 – Algoritmos comparados e seus hiper-parâmetros.

Método de regressão	Hiper-parâmetros
OLS	-
Ridge	• Alpha: $[1e{-}10, 1e{-}8, 1e{-}4, 1e{-}2, 1, 5, 10, 20]$ • Solver: $[svd, cholesky, lsqr, sparse_cg, sag, saga]$
LARS	• Alpha: $[1e{-}10, 1e{-}8, 1e{-}4, 1e{-}2, 1, 5, 10, 20]$
KNN	• K: $[3, 5, ..., 0.3 * len(base\ de\ dados)]$
Bayesian	• Alpha_1: $[1e{-}10, 1e{-}8, 1e{-}4, 1e{-}2, 1, 5, 10, 20]$ • Alpha_2 $[1e{-}10, 1e{-}8, 1e{-}4, 1e{-}2, 1, 5, 10, 20]$ • Lambda_1 $[1e{-}10, 1e{-}8, 1e{-}4, 1e{-}2, 1, 5, 10, 20]$ • Lambda_2 $[1e{-}10, 1e{-}8, 1e{-}4, 1e{-}2, 1, 5, 10, 20]$
XGBoost	• n_estimators: $[100, 200, 300, 400, 500]$ • max_depth: $[2, 3, 4]$ • learning_rate: $[1e{-}3, 1e{-}2, 1e{-}1, 1, 10]$
MLP	• Função de ativação: $[relu, tanh]$ • Número de neurônios nas camadas ocultas: $((50,), (100,), (500,), (50,50,), (100,50,), (500,50,))$

dados de teste sejam completamente isolados e não tenham participação no treino do algoritmo.

Essa normalização é feita em cada coluna pois, se considerado o conjunto total de dados, podemos ter atributos com valores em um domínio muito maior que outros, fazendo com que as características individuais de cada atributo sejam perdidas.

Apenas no caso do MLP os dados foram, além de centralizados, pré processados posteriormente com a normalização $min - max$ (equação 15), uma normalização que passa todos os valores para o intervalo de $[0, 1]$. Essa restrição é interessante por facilitar a convergência do método.

$$\mathbf{Z} = \frac{\mathbf{X} - minimo(\mathbf{X})}{maximo(\mathbf{X}) - minimo(\mathbf{X})}. \tag{15}$$

O *gridsearch* será feito para cada *fold* de treino, obtendo-se o melhor resultado para ele e aplicando a configuração de hiper-parâmetros encontrada; e então será executado para o *fold* de teste a mesma quantidade de vezes obtida para os algoritmos

aqui desenvolvidos (ou apenas uma vez no caso de algoritmos determinísticos), tendo as mesmas métricas e valores reportados que os algoritmos desenvolvidos.

A abordagem com os métodos comparados difere da abordagem com os métodos desenvolvidos no ponto de vista da otimização dos hiper-parâmetros: enquanto para cada método desenvolvido é feita uma análise variando aspectos dos operadores evolutivos que tem potencial de melhora na população para **todas as bases**, não é feito um *gridsearch* dos hiper-parâmetros do modelo; enquanto para os métodos comparados é feito um *gridsearch* para **cada base isoladamente**, de forma que os métodos executem em suas melhores configurações para cada caso. Uma otimização dos hiper-parâmetros dos métodos desenvolvidos está fora do escopo desse trabalho.

7 RESULTADOS E DISCUSSÃO

Neste capítulo, os métodos propostos utilizando a representação IT serão comparados, e aquele que apresentar o melhor desempenho será comparado posteriormente com os métodos utilizados atualmente descritos na sessão 3.5.

Para classificar o desempenho geral dos algoritmos, levando em consideração que foram utilizados 8 bases de dados diferentes, foi utilizada a seguinte estratégia:

- Todos os algoritmos tem sua métrica calculada para todas as bases de dados e então são classificados de acordo com o desempenho em cada base de dados obtendo-se a **pontuação individual**;
- Após isso, é obtido uma tabela com a classificação dos algoritmos em cada base de dados, e para cada algoritmo é calculada a média das classificações, obtendo uma nova **pontuação geral** para cada algoritmo que inclui o desempenho do mesmo em todas as bases de dados;
- Finalmente, obtida a pontuação geral, os algoritmos são classificados de acordo com estes valores.

Por exemplo, seja a tabela 2 a tabela onde as linhas 2-8 apresentam o RMSE médio dos algoritmos:

Tabela 2 – RMSE médio das melhores expressões encontradas pelo algoritmo IT-MUT, para diferentes tamanhos de população ou número de gerações.

Base de dados	IT-MUT (100 gen. 100 ind.)	IT-MUT (250 gen.)	IT-MUT (300 ind.)
Airfoil	2.904 ± 0.229	2.758 ± 0.433	2.763 ± 0.223
Concrete	7.008 ± 0.562	6.624 ± 0.31	6.705 ± 0.307
Energy Cooling	1.945 ± 0.306	1.692 ± 0.158	1.727 ± 0.175
Energy Heating	1.146 ± 0.402	0.687 ± 0.129	0.802 ± 0.208
Tower Data	31.28 ± 2.45	29.295 ± 2.043	29.876 ± 2.02
Wine Red	0.67 ± 0.258	0.676 ± 0.255	0.673 ± 0.253
Wine White	4.568 ± 36.051	0.745 ± 0.052	0.751 ± 0.145
Yacht	0.908 ± 0.429	0.904 ± 0.691	0.823 ± 0.146

Calculando a pontuação individual dos algoritmos em cada base de dados e,

com base nestas, a pontuação geral; obtemos a classificação final dos algoritmos, como ilustra a tabela 3.

Tabela 3 – Pontuação e classificação final do algoritmo IT-MUT para diferentes tamanhos de população ou número de gerações.

Base de dados	IT-MUT (100 gen. 100 ind.)	IT-MUT (250 gen.)	IT-MUT (300 ind.)
Airfoil	3	1	2
Concrete	3	1	2
Energy Cooling	3	1	2
Energy Heating	3	1	2
Tower Data	3	1	2
Wine Red	1	3	2
Wine White	3	1	2
Yacht	3	2	1
Pontuação geral	2.75	1.375	1.875
Classificação final	3	1	2

Por conveniência, apenas a tabela contendo os valores das métricas será apresentada, com a pontuação geral e classificação final incorporada, uma vez que a pontuação individual pode ser deduzida tendo-se as métricas. Para uma melhor leitura das tabelas, o primeiro colocado na pontuação individual de cada base de dados será grifado para facilitar a visualização panorâmica dos dados.

Apresentada a estratégia para classificar os algoritmos, o resto do capítulo irá discutir os aspectos dos algoritmos desenvolvidos e compará-los com as técnicas atuais.

7.1 DEFINIÇÃO DE HIPER-PARÂMETROS GERAIS

Os algoritmos desenvolvidos possuem uma série de hiper-parâmetros. Aqueles que pertencem à todos eles serão chamados de hiper-parâmetros gerais: número de gerações, limites de expoentes, tamanho da população, conjunto de funções para composição de estruturas IT. É preciso estabelecer um valor em comum para todos esses hiper-parâmetros, para que os resultados obtidos em cada um possa ser comparado em termos apenas dos parâmetros que diferem entre os modelos (destacados no quadro 3).

Como função de *fitness* foi definido o RMSE para quantificar o quão boa cada solução é, onde um menor valor indica um melhor resultado; e como função de torneio foi utilizada uma heurística simples, apenas considerando o RMSE da solução, sem incorporar informações adicionais como tamanho da expressão, linearidade, número de termos IT, entre outros possíveis.

Um teste inicial com 1000 gerações e 500 indivíduos foi feito para observar o desenvolvimento das expressões sem restrições de limites nos expoentes. Nesse cenário, as soluções apresentavam termos com potências na ordem de $10e2$. Tendo isso em vista, foi adicionado um limite de expoentes: $[-10,10]$. Ainda, foi adicionado um limitante no tamanho da expressão, no intervalo $[3, 10]$ que acabou por gerar resultados ainda melhores do que quando não havia tal restrição. A diminuição na complexidade das equações buscadas foi feita sem perda de generalidade, não impactando de forma significativa nos resultados finais mas fornecendo resultados mais interpretáveis.

Após isso, por questões do alto consumo de tempo requirido para executar todos os testes dos hiper-parâmetros específicos, o tamanho da população e o número de gerações foi fixado em 100. Para o conjunto de funções, será utilizado funções unárias comumente encontradas em equações da física e engenharia.

Dessa forma, temos para os hiper-parâmetros comuns aos três algoritmos, os valores:

- **Tamanho da população:** 100;
- **Número de gerações:** 100;
- **Conjunto de funções:** $\{id, sen, cos, tanh, \sqrt{}, log, log1p, exp\}$.

Definidos estes valores, os algoritmos foram então submetidos a uma série de testes para obtenção da melhor configuração de cada hiper-parâmetro. A próxima sessão apresentará os testes e discutirá seus resultados.

7.2 DEFINIÇÃO DE HIPER-PARÂMETROS ESPECÍFICOS

Como descrito anteriormente, dados dois diferentes operadores evolutivos (mutação e *crossover*), foram criados os possíveis algoritmos resultantes da combinação desses operadores.

Uma vez que o algoritmo IT-MUT utiliza apenas a mutação e o algoritmo IT-CX utiliza apenas o *crossover*, foi feita uma análise de sensibilidade apenas nos

operadores individualmente, e para o ITEA foi tomada como melhor configuração a combinação das melhores configurações obtidas para cada operador isoladamente. Vale notar que não necessariamente a melhor combinação dos dois operadores individualmente leva à melhor configuração deles combinados, mas uma análise considerando todas as variações possíveis está fora do escopo deste trabalho.

Tendo os hiper-parâmetros à serem variados, e utilizando os valores para eles como descrito na metodologia:

- Dados os grupos de mutação (descritos na subseção 5.1.2), cada um desses grupos terá o desempenho individual comparado e, após a comparação individual, iterativamente os 2 grupos de melhor desempenho serão combinados, até que seja formado o conjunto completo de mutação.
- Para a taxa de *crossover*, será feito o teste para os valores 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 e 0.7, onde essa taxa indica a chance para cada termo de cada expressão IT de ser selecionado para gerar o descendente.

Obtidos o melhor grupo de mutação e a melhor taxa de *crossover*, estes serão os parâmetros utilizados nos modelos para análise de seus desempenhos.

Para encontrar o número de execuções n para cada base de dados, à fim de obter um resultado estatisticamente significante para comparação, foi feito um teste com 10 execuções para cada base de dados e então o histograma foi visualmente analisado, na expectativa de encontrar uma distribuição gaussiana. Uma vez que, inicialmente, os valores não apresentavam uma distribuição como a esperada, foi-se incrementando n de 10 em 10 repetições para cada base de dados até que fosse obtido uma distribuição satisfatória.

Para ilustrar o processo de incremento de 10 em 10 execuções, a figura 12 mostra como a distribuição vai gradativamente tendendo a uma distribuição normal conforme o aumento do número de execuções, onde cada incremento de 10 execuções é sobreposto sobre o anterior. Cada cor indica a distribuição obtida em 10 execuções, e o gráfico completo é a distribuição obtida para 100 execuções, onde as cores mais escuras indicam as primeiras, enquanto as mais claras indicam as finais. A distribuição obtida apresenta um comportamento satisfatório para os fins das análises que seguem. Um maior número de execuções poderia ser utilizado, mas iria demandar muito tempo computacional.

Junto às distribuições para cada base de dados, foi traçado um gráfico da

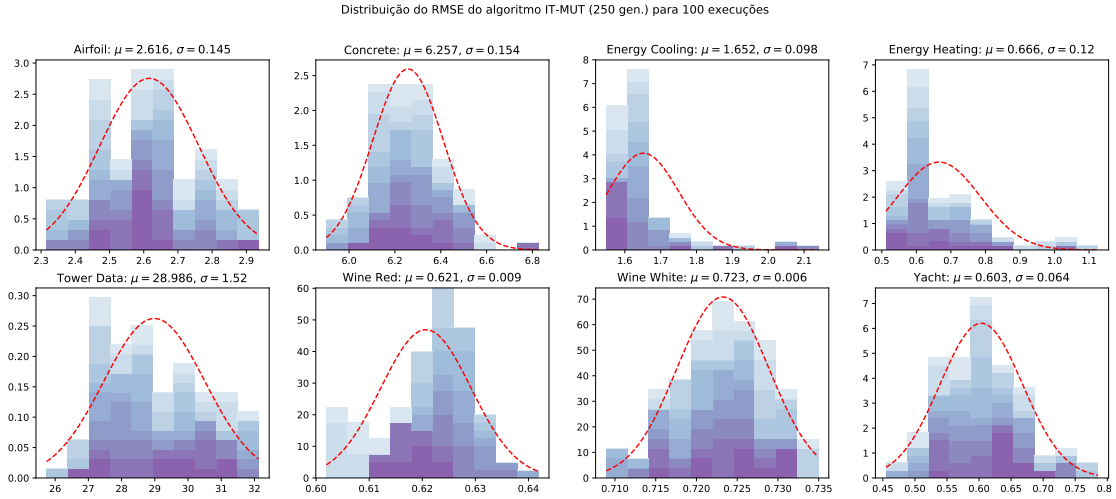


Figura 12 – Distribuição do RMSE das melhores expressões para 100 execuções, obtida a partir do empilhamento da distribuição de 10 em 10 execuções.

gaussiana correspondente à média (μ) e o desvio padrão (σ) das 100 execuções.

Definido o valor de $n = 100$, momento onde a distribuição se tornou aceitável visualmente, foi feito o teste de sensibilidade para encontrar os melhores grupos de mutação e melhor taxa de *crossover*, executando o algoritmo 100 vezes para cada base de dados.

Todos os resultados que seguem levam em consideração os dados para o *fold* de teste, pois utilizar o resultado obtido no treino pode apresentar problemas, como em casos de *overfit*, onde o valor para o treino pode ser bom mas a expressão encontrada pode falhar em predizer valores para dados não visitados.

Para determinar as melhores configurações foi utilizada como métrica a mediana, por ser uma métrica mais robusta a *outliers* (pontos muito distante das demais medidas, inconsistentes). Quando é feito um estudo de algoritmos estocásticos - especialmente os evolutivos - a ocorrência de *outliers* é inevitável. A mediana representa o valor central, teoricamente o pico da distribuição dos algoritmos. No uso de outras métricas, como o RMSE, *outliers* extremamente grandes podem afetar muito a média e o desvio padrão do resultado.

Para todos os gráficos de convergência que seguem nas subseções, a linha de convergência para cada base foi obtida calculando a média da melhor solução em cada

geração para as 100 execuções.

7.2.1 GRUPOS DE MUTAÇÃO

Para a definição do melhor grupo de mutação, primeiro foi feito a execução para cada grupo individualmente. Os resultados estão na tabela 4. Para isso, foi utilizado o algoritmo IT-MUT, já que esse se baseia apenas na mutação.

Tabela 4 – Medianas das melhores expressões para os grupos de mutação isolados.

Base de dados	Add e Drop	Term	Intern e Interp	Func
Airfoil	3.269	4.065	3.488	4.66
Concrete	7.018	10.402	11.148	12.988
Energy Cooling	2.178	3.104	2.88	4.355
Energy Heating	1.501	2.702	2.921	4.512
Tower Data	35.689	35.926	39.762	61.12
Wine Red	0.641	0.656	0.681	0.739
Wine White	0.741	0.766	0.778	0.84
Yacht	1.854	2.182	1.248	5.505
Pontuação geral	1.125	2.375	2.5	4.0
Classificação final	1	2	3	4

Após isso, foi percebido que o melhor grupo individual é $Add, Drop$ (sendo superado em apenas uma base de teste); seguido do grupo $Term$. Então, esses dois grupos foram combinados e o grupo obtido teve seu resultado reportado. Novamente, por ser um processo interativo, os dois melhores grupos foram $Add, Drop, Term$ e $Intern, Interp$, resultando na combinação dos dois, obtendo os resultados para as 100 execuções. Por fim, todos os grupos individuais foram combinados. Os resultados de cada etapa estão na tabela 5.

Dentre todas as possíveis combinações, a que apresentou o pior desempenho foi o grupo $Func$, sendo a última a ser incluída no grupo. Quando por fim foi incluída, resultou em uma queda no desempenho geral em todas as bases de dados. Desta forma, o melhor grupo de mutação obtido foi $Add, Drop, Term, Intern, Interp$. A consecutiva combinação dos grupos individuais foi gradativamente melhorando o desempenho do algoritmo, exceto quando o grupo incorporou a mutação *func*.

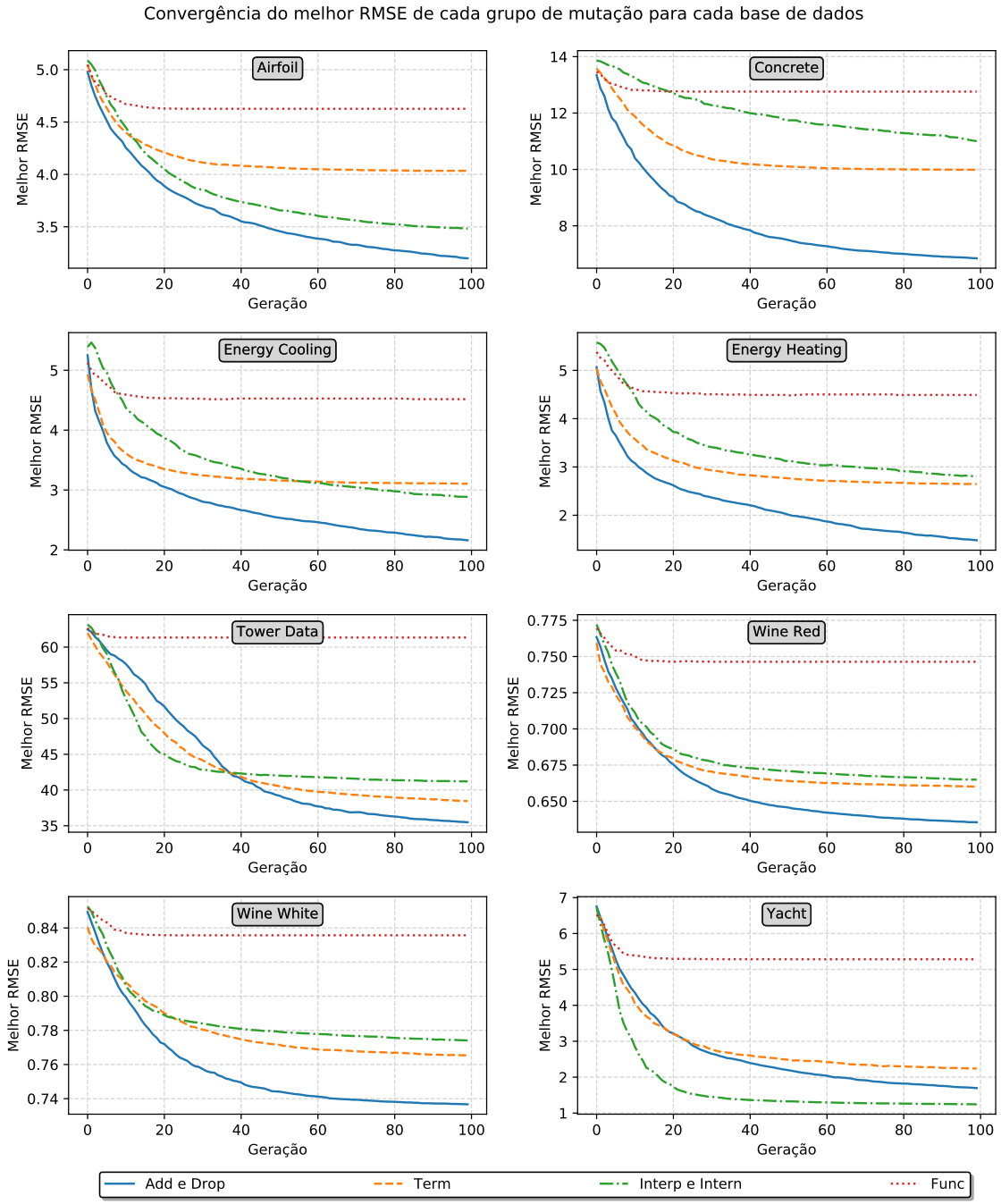


Figura 13 – Convergência da média do melhor RMSE de cada grupo de mutação individual sobre 100 execuções para cada base de dados.

Tabela 5 – Medianas das melhores expressões para os grupos de mutação combinados.

Base de dados	Add, Drop e Term	Add, Drop, Term, Intern e Interp	Todos os grupos
Airfoil	3.183	2.893	2.932
Concrete	6.95	6.908	6.962
Energy Cooling	1.99	1.888	1.851
Energy Heating	1.274	1.035	1.107
Tower Data	34.374	31.32	32.196
Wine Red	0.637	0.635	0.64
Wine White	0.736	0.737	0.735
Yacht	1.221	0.849	0.871
Pontuação geral	2.625	1.375	2.0
Classificação final	3	1	2

Analisando os gráficos de convergência dos grupos isoladamente (figura 13), vemos que, de fato, o grupo $Func$ é o de pior desempenho. Não apenas isso, esse grupo de mutação atinge um platô em todos os casos próximo das 20 gerações. Isso ocorre pelo fato de que a quantidade de diferentes funções que essa mutação pode gerar é muito limitada, visto que ela modifica apenas 1 aspecto dos termos IT (o que possui o menor intervalo, a função, que pode ser uma das utilizadas para o problema), sem modificar o tamanho das expressões.

O grupo $Term$ também apresenta uma estabilização na convergência, mas tem uma melhora no valor do RMSE muito grande durante as primeiras gerações.

O grupo $Intern, Interp$ apresenta fortes indícios de que está próximo de se estabilizar para todas as bases de dados.

Por fim, o grupo $Add, Drop$ apresenta o melhor desempenho de todos, com uma convergência (na maioria dos casos) mais rápida que todos os outros grupos.

Conforme os grupos vão sendo combinados, as convergências tendem a sempre apresentarem um comportamento parecido (figura 14). Apesar de individualmente o grupo $Add, Drop$ ter sido o melhor grupo, exceto quando combinado com $Func$, ele apresenta um melhor desempenho sem ser utilizado isoladamente.

Com base nos gráficos de convergência e na classificação dos grupos, conclui-se que o melhor grupo de mutação é $Add, Drop, Term, Intern, Interp$.

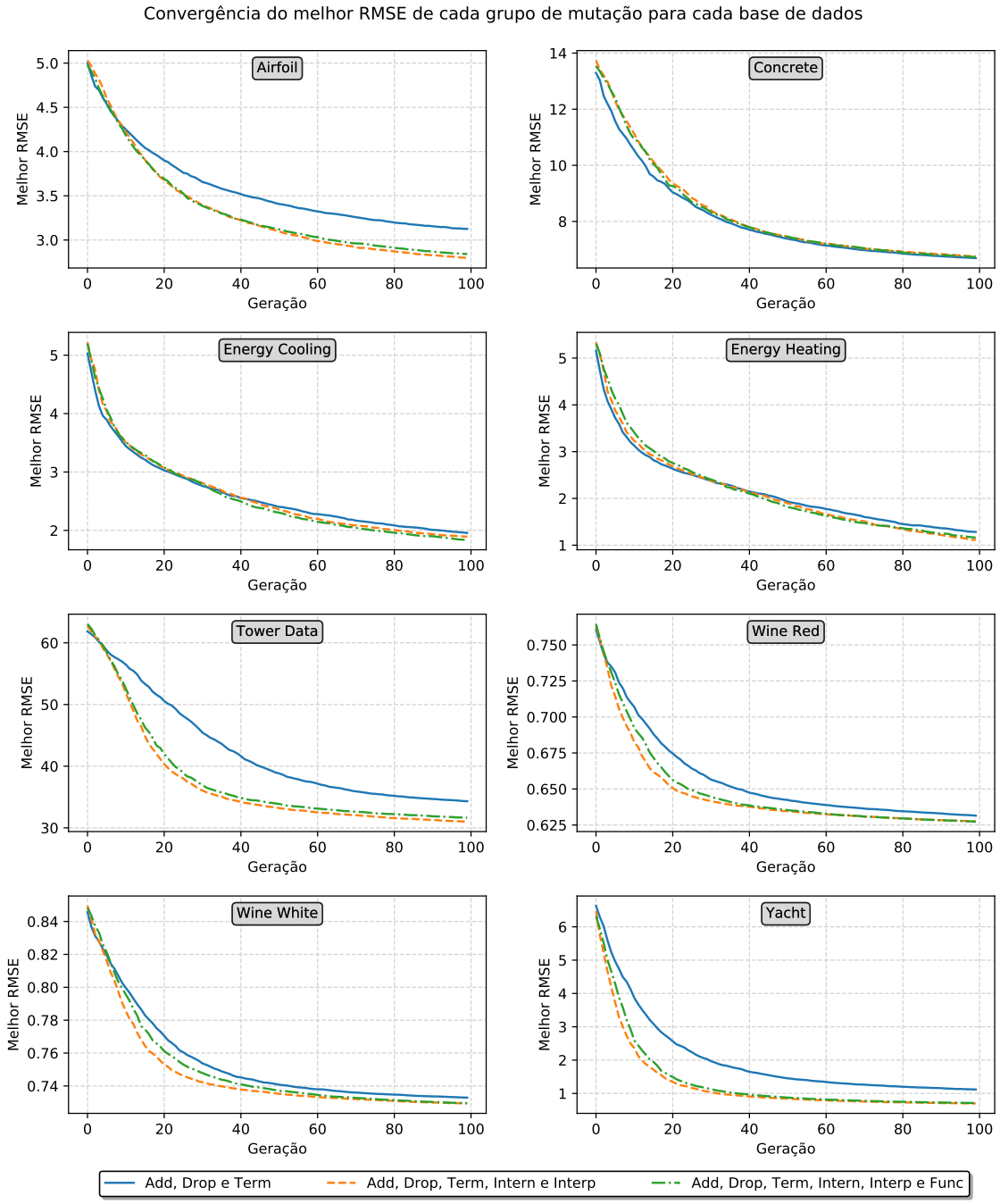


Figura 14 – Convergência da média do melhor RMSE de cada grupo de mutação composto sobre 100 execuções para cada base de dados.

7.2.2 TAXA DE *CROSSOVER*

Da mesma forma que os grupos de mutações foram analisados, a taxa de *crossover* também será, para que o algoritmo IT-CX também seja utilizado em sua melhor configuração, obtendo seu melhor resultado dentro dos valores fixos definidos para todos os algoritmos.

Para cada base de dados foi aplicada a mesma metodologia, com 20 execuções para cada *fold* (totalizando 100 execuções), tendo as medianas reportadas na tabela 6.

Tabela 6 – Medianas das melhores expressões para diferentes taxas de *crossover*.

Base de dados	Taxa 0.10	Taxa 0.20	Taxa 0.30	Taxa 0.40	Taxa 0.50	Taxa 0.60	Taxa 0.70
Airfoil	4.425	4.402	4.495	4.588	4.78	4.969	4.987
Concrete	11.817	11.503	11.889	12.36	13.05	13.46	13.772
Energy Cooling	3.781	3.747	3.708	3.945	4.082	4.648	4.779
Energy Heating	3.723	3.692	3.733	3.948	4.295	4.677	5.372
Tower Data	58.865	58.821	59.317	59.337	60.073	60.946	61.543
Wine Red	0.737	0.731	0.733	0.74	0.751	0.749	0.762
Wine White	0.834	0.831	0.835	0.841	0.845	0.85	0.851
Yacht	4.0	4.017	4.264	4.451	5.388	5.815	5.968
Pontuação geral	2.125	1.25	2.625	4.0	5.125	5.875	7.0
Classificação final	2	1	3	4	5	6	7

A taxa que obteve o melhor resultado foi 0.2 (20%), ou seja, dados duas expressões pais para a criação de uma nova expressão, a chance de cada termo IT ser herdado é de 20%. Pela classificação final, vemos que quanto maior a taxa menor a classificação final (com exceção da taxa 0.10).

A taxa de *crossover* tem um reflexo direto no tamanho da expressão filha, pois uma baixa taxa seleciona poucos termos, enquanto uma de taxa alta seleciona vários.

Uma taxa maior que 0.5 (50%) implica que mais da metade dos termos de cada pai será selecionado, de forma que a equação resultante possa ser maior que o limite definido de 10 termos IT caso os pais já estejam em uma configuração de tamanho máximo.

Para taxas pequenas, é esperado que poucos ou quase nenhum termo IT seja herdado. No primeiro caso, a expressão será menor que os pais e, uma vez que a seleção dos termos é feita de forma totalmente aleatória (sem considerar o coeficiente associado, que pode ser um indicador de importância do termo para a expressão final) não há

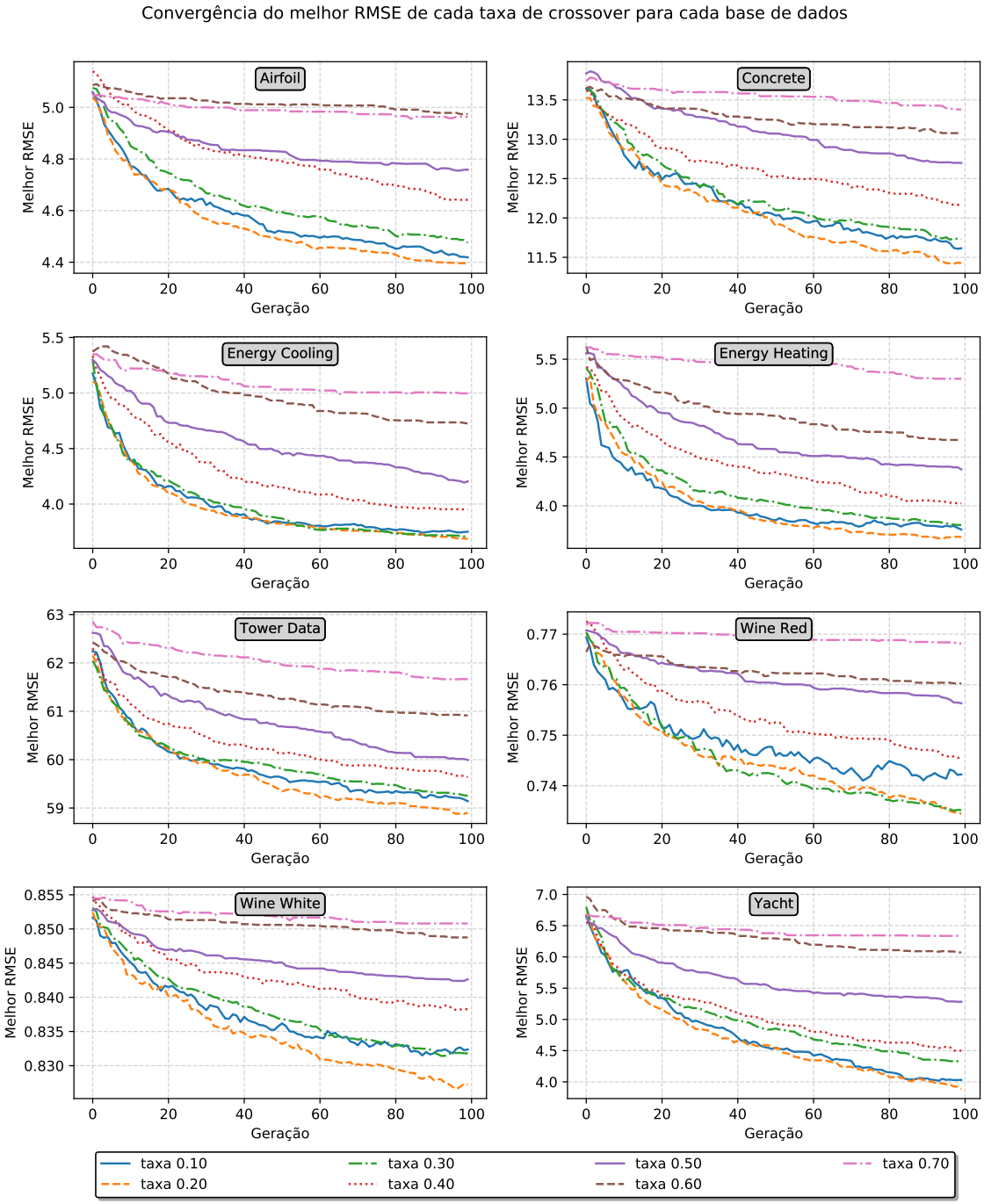


Figura 15 – Convergência da média do melhor RMSE de cada taxa de *crossover* sobre 100 execuções para cada base de dados.

garantia de que o resultado será melhor para todo *crossover* realizado. No segundo caso, onde a expressão filha é nula, o operador de criação é aplicado e retorna uma solução aleatória.

Analisando os gráficos de convergência (figura 15), vemos que quanto menor a taxa, mais turbulenta a conversão é, apresentando uma curva pouco suavizada; ao contrário das altas taxas de *crossover*, que apresentam uma curva mais suavizada, junto de uma convergência mais lenta. Se comparadas com as curvas de convergência da mutação (figuras 13 e 14), a convergência ocorre com um desempenho muito inferior, uma vez que as curvas de convergência do IT-MUT são bem suaves.

7.2.3 VARIAÇÕES NO TAMANHO DA POPULAÇÃO E NO NÚMERO DE INDIVÍDUOS

Definidos os melhores grupos de mutação e melhor taxa de *crossover*, uma análise das possíveis configurações de ambos juntos poderia ser feita para o algoritmo ITEA, mas demandaria um tempo indisponível em um prazo pequeno para experimentar todas as combinações possíveis - dessa forma, seu grupo de mutação e taxa de *crossover* é a combinação dos melhores resultados obtidos para o IT-MUT e IT-CX.

Determinados os melhores hiper-parâmetros para os algoritmos, uma pequena análise da variação do tamanho da população e do número de gerações foi feita.

Para o tamanho da população, foram utilizados os melhores hiper parâmetros obtidos pela análise, e todos os hiper parâmetros fixos foram mantidos em seus valores, exceto o tamanho da população, que passou de 100 para 300 indivíduos, tendo sua convergência e medianas comparadas, assim como as anteriores. Para o número de gerações foi utilizada a mesma estratégia, dessa vez, utilizando como número de gerações 250. A escolha desses valores foi feita de forma que, no mínimo, os valores iniciais fossem dobrados; mas sem aumentar muito o tempo necessário para execução. As medianas obtidas foram reportadas nas tabelas 7, 8 e 9.

É possível observar que as duas perturbações causadas nos algoritmos (aumentar o número de indivíduos ou o número de gerações) gera um resultado melhor em quase todos os casos, sendo a configuração com 100 indivíduos e 100 gerações nunca a primeira na classificação geral, resultado esperado pois permite uma maior variedade de soluções ou uma convergência mais longa.

Tabela 7 – Medianas das melhores expressões para o algoritmo IT-MUT, para diferentes tamanhos de população ou número de gerações.

IT-MUT	100 ind. e 100 gen.	100 ind. e 250 gen.	300 ind. e 100 gen.
Airfoil	2.893	2.72	2.726
Concrete	6.908	6.592	6.663
Energy Cooling	1.888	1.674	1.699
Energy Heating	1.035	0.654	0.751
Tower Data	31.32	28.842	29.538
Wine Red	0.635	0.635	0.629
Wine White	0.737	0.731	0.732
Yacht	0.849	0.8	0.816
Pontuação geral	2.875	1.125	1.875
Classificação final	3	1	2

Tabela 8 – Medianas das melhores expressões para o algoritmo IT-CX, para diferentes tamanhos de população ou número de gerações.

IT-CX	100 ind. e 100 gen.	100 ind. e 250 gen.	300 ind. e 100 gen.
Airfoil	4.402	4.362	4.364
Concrete	11.503	10.82	11.26
Energy Cooling	3.747	3.587	3.546
Energy Heating	3.692	3.514	3.421
Tower Data	58.821	58.299	58.262
Wine Red	0.731	0.72	0.712
Wine White	0.831	0.825	0.821
Yacht	4.017	3.088	3.086
Pontuação geral	3.0	1.75	1.25
Classificação final	3	2	1

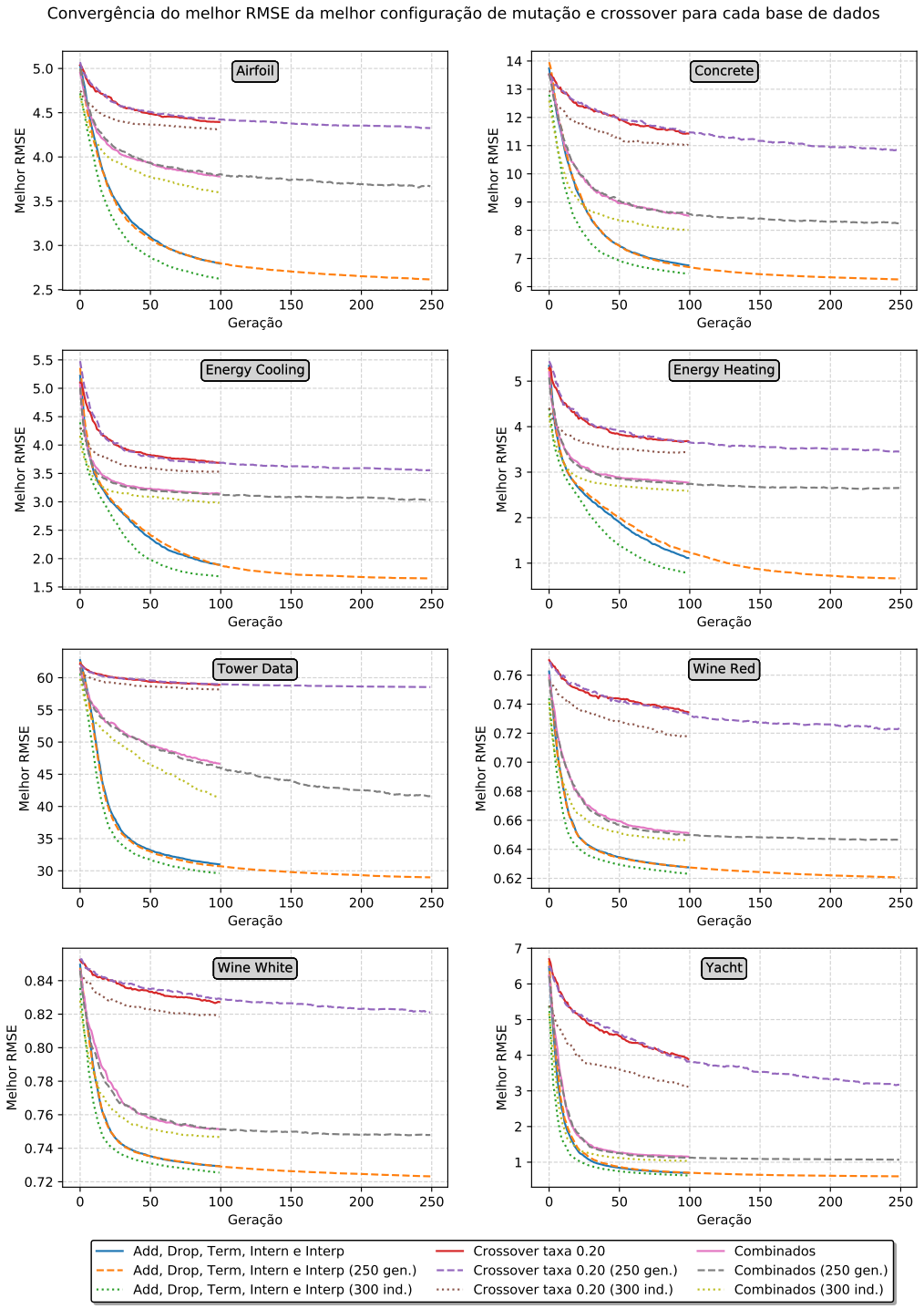


Figura 16 – Convergência da média do melhor RMSE para o IT-MUT, IT-CX e ITEA sobre 100 execuções, variando a população ou o número de gerações.

Tabela 9 – Medianas das melhores expressões para o algoritmo ITEA, para diferentes tamanhos de população ou número de gerações.

ITEA	100 ind. e 100 gen.	100 ind. e 250 gen.	300 ind. e 100 gen.
Airfoil	3.793	3.793	3.666
Concrete	8.578	8.578	8.103
Energy Cooling	3.164	3.164	3.032
Energy Heating	2.816	2.816	2.596
Tower Data	46.982	46.982	41.143
Wine Red	0.645	0.645	0.646
Wine White	0.753	0.753	0.748
Yacht	1.22	1.22	1.128
Pontuação geral	1.875	1.875	1.25
Classificação final	2	2	1

Vemos no gráfico de convergência (figura 16) que:

- O aumento do tamanho da população (linha pontilhada) desloca a convergência um pouco para baixo em **todos** os casos em relação à configuração original de 100 indivíduos e 100 gerações (linha contínua);
- O aumento do número de gerações (linha tracejada) sobrepõe a linha contínua e complementa-a por mais 200 gerações.

Ao fim dessa sessão temos definidos os melhores hiper-parâmetros, e concluímos que o algoritmo IT-MUT apresenta um melhor desempenho com 250 gerações; e os algoritmos ITEA e IT-CX com 300 indivíduos. Tendo esses resultados, serão essas as configurações utilizadas para analisar os resultados comparar com os algoritmos alvo do projeto.

A partir deste ponto, quando um algoritmo desenvolvido é referido, os resultados apresentados serão com o experimento de 20 execuções por *fold* para sua melhor configuração (e.g. quando lê-se IT-MUT os resultados apresentados serão referentes à 20 execuções por *fold* para cada base de dados utilizando 100 indivíduos, 250 gerações, e os grupos de mutação *Add*, *Drop*, *Intern*, *Interp* e *Term*).

7.3 ANÁLISE DOS RESULTADOS

Selecionadas as melhores configurações, os resultados serão analisados para elas de dois pontos de vista:

- O aspecto qualitativo dos resultados - relacionado à interpretabilidade e tamanho das expressões; e
- O aspecto quantitativo - relacionado ao desempenho dos algoritmos para as métricas descritas na metodologia, julgando os resultados e realizando uma análise estatística para determinar o desempenho dos algoritmos.

7.3.1 TAMANHO DAS EXPRESSÕES

A interpretabilidade está intimamente relacionada com o tamanho da solução. Nesta sub-sessão, serão analisados alguns resultados obtidos pelos algoritmos em suas melhores configurações.

Para calcular o tamanho de uma expressão IT é preciso convertê-la para uma árvore sintática e então contar a quantidade de nós resultantes. Uma expressão IT é a combinação linear de termos IT, que tem sempre um coeficiente associado (figura 17).

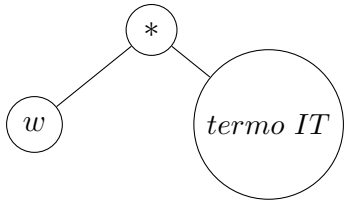


Figura 17 – Raiz de uma expressão IT, contendo o coeficiente e o termo IT.

O $termo\ IT$ é a aplicação de uma função de transformação sobre a interação das variáveis do problema, onde cada variável tem um expoente associado. Então, para cada variável, temos 3 nós: *i)* a variável, *ii)* o operador de exponenciação, e *iii)* o expoente. O conjunto desses 3 nós será denotado por **nó de interação**. O nó de interação implica que para cada variável do problema teremos um incremento de $+3$ no valor do tamanho da expressão, porém temos alguns casos especiais:

- No caso onde o expoente é 0, o resultado é sempre 1, elemento neutro da multiplicação, de forma que o nó de interação da variável tem tamanho indiferente para o tamanho da expressão, sendo desconsiderado; e

- No caso onde o expoente é 1 temos o equivalente à apenas a variável, tendo tamanho 1.

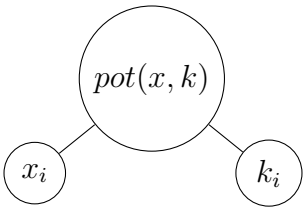


Figura 18 – Nó de interação genérico.

Como toda expressão IT combina linearmente vários termos IT, seja m o número de termos IT, teremos $m - 1$ nós de soma entre eles. O mesmo vale para as variáveis de interação, seja n o número de variáveis, teremos $n - 1$ nós de multiplicação entre cada nó de interação para cada termo.

Então, para uma termo IT, teremos seu tamanho calculado pela soma dos valores:

- $+3$, representando a multiplicação do coeficiente pelo termo;
- $+1$, representando a função de transformação;
- $+n - 1$, representando a interação das variáveis;
- Para os expoentes, temos:
  - $+1$ para todas as variáveis de expoente 1;
  - $+3$ para todas as variáveis de expoente diferente de 1;
  - -1 para todas as variáveis de expoente 0, equivalente à remoção da multiplicação entre o nó insignificante e o restante dos nós de interação.

E para uma expressão IT teremos a soma do tamanho de todos os termos que a compõem, somado de $m - 1$. Note que aqui não é preciso compensar os termos nulos de forma análoga como é compensado os nós de interação de expoente 0, pois os operadores evolutivos não permitem a existência de termos IT nulos.

Definida a estratégia para obter o tamanho da expressão, o tamanho médio de cada algoritmo para cada base de dados foi calculado e reportado na tabela 10.

É possível observar que o algoritmo IT-CX apresenta a melhor classificação, com uma grande diferença com o pior colocado, o algoritmo IT-MUT. A combinação da mutação com o *crossover* no algoritmo ITEA entrega um resultado intermediário. Como explicado anteriormente, uma baixa taxa de *crossover* resulta em uma menor

Tabela 10 – Tamanho médio das melhores expressões encontradas para cada algoritmo desenvolvido com suas melhores configurações.

Base de dados	IT-MUT	IT-CX	ITEA
Airfoil	144.64 ± 13.624	30.9 ± 11.228	64.26 ± 14.854
Concrete	181.01 ± 17.269	41.9 ± 16.572	71.48 ± 17.932
Energy Cooling	171.37 ± 20.722	43.77 ± 12.321	64.46 ± 18.696
Energy Heating	173.93 ± 18.829	47.53 ± 12.771	61.14 ± 15.062
Tower Data	738.1 ± 72.669	112.78 ± 45.133	188.73 ± 98.793
Wine Red	291.88 ± 33.127	64.44 ± 23.47	96.03 ± 36.084
Wine White	288.86 ± 33.362	70.53 ± 20.107	92.58 ± 29.14
Yacht	166.97 ± 21.292	23.38 ± 13.485	53.43 ± 14.911
Pontuação geral	3.0	1.0	2.0
Classificação final	3	1	2

quantidade de termos herdados e, consequentemente, em uma menor expressão[1].

Para ilustrar o impacto do tamanho das expressões na interpretabilidade do resultado, serão apresentadas algumas expressões encontradas para a primeira base de dados, *Airfoil*. Essa base foi escolhida por ser a base de menor número de atributos - como o interesse é analisar qualitativamente o resultado, torna-se desnecessário utilizar bases com grande número de atributos (como *Wine Red*), que ocupariam páginas devido à natureza já bastante complexa dos dados, para obter as mesmas conclusões. Convém ressaltar que, apesar de feita esta análise apenas para uma base de dados, ela se estende para todas as outras, e visto que a pontuação individual dos três algoritmos é exatamente a mesma para todas as bases (pois a pontuação geral e classificação final são as mesmas) a análise pode ser generalizada para os três algoritmos em todas as bases de dados.

As expressões apresentadas aqui são correspondentes ao resultado de menor RMSE encontrado, ou seja, não necessariamente são a menor obtida, mas a que mais minimiza o erro de predição. Como podem ocorrer *outliers* nos resultados, selecionando as de menor RMSE permite que a análise seja feita em um resultado prático e desejado quando se realiza uma regressão.

Primeiro, o algoritmo IT-MUT, que obteve a pior classificação, apresentou

[1] O teste estatístico está reportado no apêndice B.

como expressão de menor RMSE:

Expressão de menor RMSE (2.250) para o IT-MUT (tamanho=144)

```
 1.471525*log1p(x3^-3 * x4^-2) +
 0.607964*log1p(x3^7 * x4^-4) +
-0.954491*log1p(x0^-9 * x1^2 * x2^-7 * x3^3 * x4^-6) +
-0.513344*log1p(x0^8 * x2^7 * x3^-4 * x4^3) +
-1.42186*log1p(x0^-8 * x2^2 * x3^2 * x4^-9) +
 0.445444*log1p(x0^-8 * x1^3 * x2^-7 * x3^6 * x4^-3) +
-3.310411*log1p(x0^-2 * x2^-3 * x3^2 * x4) +
-2740.416799*log1p(x1 * x4^3) +
 0.011361*id(x0 * x4) +
-0.141906*sqrt(x0) +
 110.57095766780208
```

Já para o IT-CX, algoritmo de melhor classificação, temos um resultado muito superior em relação ao anterior, com uma interpretabilidade muito superior:

Expressão de menor RMSE (3.905) para o IT-CX (tamanho=27)

```
-2.440403*log1p(x0^3 * x2^3 * x4^3) +
 9.811232*log1p(x1^2 * x2^3) +
 128.62804941792177
```

E o ITEA, que obteve desempenho médio:

Expressão de menor RMSE (3.179) para o ITEA (tamanho=86)

```
 0.979865*log1p(x0^-6 * x2^-8 * x3^4 * x4^-8) +
-0.512595*log1p(x0^8 * x2^-4 * x3^-2 * x4^-4) +
-2.138797*log1p(x0^-8 * x2^-4 * x3^6 * x4^-4) +
-151537286637.53867*log1p(x2^2 * x3^-4 * x4^2) +
 1.438948*log1p(x0^2 * x2 * x3 * x4) +
 123.3527228794443
```

Onde $x0, x1, \ldots, x4$ são as variáveis explanatórias que predizem y.

Vemos que, para a melhor expressão encontrada, os tamanhos estão dentro do intervalo $[\mu - \sigma, \mu + \sigma]$ para os algoritmos IT-MUT e IT-CX, onde μ representa a média e σ o desvio padrão. Para o algoritmo ITEA o resultado está fora deste intervalo.

Com base nas expressões apresentadas, a base de dados escolhida para análise será brevemente apresentada. O *Airfoil* é uma base de dados disponibilizada pela NASA obtido a partir de uma série de experimentos realizados dentro de um túnel de vento (mais detalhes no documento original (BROOKS; POPE; MARCOLINI, 1989)), onde os parâmetros (adaptados para nossa representação) são:

- $x0$ - Frequência, em Hertz;
- $x1$ - Ângulo de ataque, em graus;
- $x2$ - Comprimento da corda, em metros;
- $x3$ - Velocidade do fluxo livre, em metros por segundo;
- $x4$ - Espessura do deslocamento lateral da sucção, em metros.

Dados essas variáveis explicatórias, o objetivo da regressão é encontrar uma expressão $\hat{f}(x0, x1, \ldots, x4)$ que prediz suficientemente bem a variável alvo y que representa o nível de pressão sonora escalado, em decibéis.

É possível realizar uma inferência com muito mais facilidade para a expressão encontrada pelo IT-CX, que apresenta maior simplicidade, porém esse é o resultado de maior erro associado. O resultado encontrado pelo IT-MUT apresenta o menor erro entre eles, mas uma função mais complexa. Claro que, se analisados as funções de transformação, os pesos associados à cada termo IT, e como os expoentes de cada variável se comportam em média, é possível obter algumas informações interessantes sobre como as variáveis se relacionam - mas isso demanda uma análise muito mais complexa que para o caso do IT-CX. Por fim, podemos ver que o ITEA entrega um resultado intermediário entre os dois, tanto no erro da predição quanto no tamanho, podendo ser uma evidência de que há um *payoff* entre minimização do erro e minimização da complexidade das expressões.

Esse é um resultado esperado, uma vez que maiores expressões garantem maior expressividade, mas também pode ser um indício de que a expressão está sofrendo *overfit*.

7.3.2 ERRO NA PREDIÇÃO

Para avaliar quantitativamente os algoritmos, todas as métricas citadas na metodologia (MAE, NMSE, RMSE) foram reportadas para cada base de dados.

Seguindo a mesma estratégia adotada para a análise qualitativa, foram utiliza-

dos os algoritmos em suas melhores configurações, cujo resultados estão reportados nas tabelas 11, 12 e 13.

Tabela 11 – RMSE médio das melhores expressões para cada algoritmo desenvolvido.

Base de dados	IT-MUT	IT-CX	ITEA
Airfoil	2.758 ± 0.433	4.335 ± 0.163	3.66 ± 0.209
Concrete	6.624 ± 0.31	11.099 ± 0.843	8.116 ± 0.434
Energy Cooling	1.692 ± 0.158	3.555 ± 0.199	3.023 ± 0.216
Energy Heating	0.687 ± 0.129	3.447 ± 0.32	2.605 ± 0.198
Tower Data	29.295 ± 2.043	58.27 ± 2.534	41.62 ± 4.541
Wine Red	0.676 ± 0.255	0.722 ± 0.039	5402.607 ± 53748.761
Wine White	0.745 ± 0.052	0.821 ± 0.019	0.75 ± 0.013
Yacht	0.904 ± 0.691	3.212 ± 1.158	1.166 ± 0.156
Pontuação geral	1.0	2.875	2.125
Classificação final	1	3	2

Tabela 12 – NMSE médio das melhores expressões para cada algoritmo desenvolvido.

Base de dados	IT-MUT	IT-CX	ITEA
Airfoil	0.0 ± 0.0	0.0 ± 0.001	0.0 ± 0.001
Concrete	0.003 ± 0.034	0.014 ± 0.098	0.005 ± 0.051
Energy Cooling	0.001 ± 0.004	0.002 ± 0.021	0.002 ± 0.015
Energy Heating	0.0 ± 0.001	0.005 ± 0.024	0.002 ± 0.013
Tower Data	0.001 ± 0.007	0.003 ± 0.029	0.003 ± 0.014
Wine Red	0.028 ± 0.013	0.002 ± 0.016	171184.593 ± 0.013
Wine White	0.003 ± 0.015	0.001 ± 0.019	0.001 ± 0.016
Yacht	0.031 ± 0.006	0.071 ± 0.08	0.003 ± 0.012
Pontuação geral	1.5	2.0	1.75
Classificação final	1	3	2

Vemos que, na grande maioria dos casos, o algoritmo de menor erro em todas as métricas foi o IT-MUT, seguido do ITEA, e o IT-CX foi o de pior desempenho, baseando-se na pontuação individual[2].

[2] O teste estatístico está reportado no apêndice C.

Tabela 13 – MAE médio das melhores expressões para cada algoritmo desenvolvido.

Base de dados	IT-MUT	IT-CX	ITEA
Airfoil	0.186 ± 2.009	0.115 ± 3.302	0.167 ± 2.813
Concrete	0.247 ± 5.082	0.731 ± 8.778	0.395 ± 6.332
Energy Cooling	0.149 ± 1.177	0.177 ± 2.664	0.131 ± 2.183
Energy Heating	0.102 ± 0.501	0.268 ± 2.634	0.161 ± 1.899
Tower Data	1.684 ± 21.206	2.192 ± 44.505	3.635 ± 31.508
Wine Red	0.031 ± 0.494	0.035 ± 0.562	3004.647 ± 0.507
Wine White	0.011 ± 0.574	0.014 ± 0.635	0.01 ± 0.586
Yacht	0.117 ± 0.518	0.624 ± 1.94	0.096 ± 0.743
Pontuação geral	1.625	2.5	1.875
Classificação final	1	3	2

Em particular, o algoritmo ITEA apresentou um único *outlier* durante as execuções com um erro tão grande que desconfigurou os valores de μ e σ obtidos. No geral, todos os valores σ são pequenos se comparados com μ, indicando que a ocorrência de *outliers* é baixa.

Vemos também que o algoritmo ITEA faz um equilíbrio entre acurácia e interpretabilidade, podendo ser promissor estudar e desenvolver mais essa variação específica, à fim de melhorar a interpretabilidade e diminuir o erro, características que o IT-MUT e IT-CX não conseguem apresentar simultaneamente.

É possível observar que o algoritmo baseado apenas na mutação é capaz de obter um bom desempenho, e o uso da mutação com o *crossover* em conjunto melhora o desempenho da variante que utiliza apenas o *crossover*.

Podemos concluir que o IT-MUT é um algoritmo robusto que permite obter resultados mais competitivos, quando o foco está na minimização de erro, seguido do algoritmo ITEA, e por fim do algoritmo IT-CX. Esse resultado é particularmente interessante devido à simplicidade de implementação de um algoritmo baseado apenas em uma mutação.

7.4 DESEMPENHO COMPARADO COM TÉCNICAS ATUAIS

Para obter uma visão panorâmica do desempenho do melhor algoritmo encontrado, ele será comparado com técnicas bem sucedidas e amplamente utilizadas atualmente. Esta comparação será feita em termos de minimização de erro, pois é um dos objetivos mais almejados atualmente.

O algoritmo proposto aqui com o melhor desempenho foi o IT-MUT, com as seguintes configurações:

Quadro 5 – Melhores hiper-parâmetros encontrados para o IT-MUT.

Hiper-parâmetro	Melhor configuração
Tamanho da população	100
Número de gerações	250
Conjunto de funções	$\{id, sen, cos, tanh, \sqrt{}, log, log1p, exp\}$
Mutações	Add, Drop, Intern, Interp, Term

Utilizando esses parâmetros, as outras técnicas serão comparadas pelo RMSE médio resultante de uma mesma quantidade de execuções utilizada para todas as outras análises (20 execuções por *fold* para cada base de dados, sendo que penas os métodos determinísticos foram executados uma única vez por *fold*). Apenas o RMSE será utilizado como métrica para alinhar os resultados de acordo com a literatura atual, onde esse é o valor amplamente utilizado para comparação de desempenho em vários artigos. Todos os métodos comparados tiveram seus hiper-parâmetros ajustados por um processo de *gridsearch* para cada base de dados, que faz uma busca exaustiva de todas as combinações e obtém aquela que melhor minimiza o erro. Essas execuções foram feitas para os mesmos *folds* de cada base de dados e tiveram seu desempenho médio reportado na tabela 14[3].

Vemos que o melhor método é o **XGBoost**, com uma pontuação geral muito distante do segundo colocado. A tendência do XGBoost é apresentar um erro cada vez menor quanto mais estimadores. Para as possíveis configurações, em muitos casos o melhor desempenho foi obtido com 500 estimadores de profundidade 4.

[3] O teste estatístico está reportado no apêndice D, junto com uma versão da tabela contendo todas as pontuações gerais.

Tabela 14 – RMSE médio dos melhores resultados para o IT-MUT e os modelos comparados.

Dataset	IT-MUT	Bayesian	LARS lasso	MLP
Airfoil	2.758 ± 0.433	4.819 ± 0.156	4.819 ± 0.156	8.893 ± 0.482
Concrete	6.624 ± 0.31	10.446 ± 0.432	10.447 ± 0.447	22.128 ± 1.398
EnergyCooling	1.692 ± 0.158	3.215 ± 0.1	3.215 ± 0.098	13.212 ± 0.599
EnergyHeating	0.687 ± 0.129	2.942 ± 0.167	2.942 ± 0.168	14.122 ± 0.241
TowerData	29.295 ± 2.043	30.666 ± 3.141	30.44 ± 3.271	120.821 ± 7.731
WineRed	0.676 ± 0.255	0.651 ± 0.026	0.651 ± 0.026	0.948 ± 0.037
WineWhite	0.745 ± 0.052	0.754 ± 0.011	0.756 ± 0.011	1.064 ± 0.065
Yacht	0.904 ± 0.691	8.998 ± 0.622	8.963 ± 0.612	20.757 ± 1.656
Pontuação geral	3.0	4.75	4.75	8.0
Classificação final	3	5	5	7
Dataset	OLS	Ridge	XGBoost	KNN
Airfoil	4.819 ± 0.156	4.82 ± 0.156	1.612 ± 0.131	2.684 ± 0.146
Concrete	10.445 ± 0.438	10.447 ± 0.442	4.3 ± 0.285	8.713 ± 0.748
EnergyCooling	3.211 ± 0.1	3.215 ± 0.099	0.857 ± 0.139	2.531 ± 0.054
EnergyHeating	2.935 ± 0.164	2.942 ± 0.168	0.356 ± 0.049	2.281 ± 0.147
TowerData	30.432 ± 3.274	30.542 ± 3.223	15.665 ± 0.698	14.064 ± 0.5
WineRed	0.652 ± 0.025	0.651 ± 0.025	0.613 ± 0.036	0.657 ± 0.037
WineWhite	0.754 ± 0.012	0.754 ± 0.011	0.667 ± 0.006	0.719 ± 0.012
Yacht	9.012 ± 0.606	8.996 ± 0.626	0.672 ± 0.086	8.952 ± 1.622
Pontuação geral	4.5	5.0	1.125	2.875
Classificação final	4	6	1	2

O segundo colocado é o **KNN**, um método baseado em instância, que funciona armazenando toda a informação disponível para treino e, no momento de fazer uma nova predição, utiliza todos os dados de treino para estimar o valor de y.

O melhor dos métodos propostos nesse trabalho, o **IT-MUT**, ficou em terceiro colocado, com uma pontuação geral próxima do segundo, mas distante do primeiro e quarto colocados. Ainda, nenhum método paramétrico superou o IT-MUT na pontuação geral.

Dos demais métodos, o **Bayesian**, **OLS**, **LARS** e **Ridge** obtiveram praticamente as mesmas pontuações gerais, desempenho médio e desvio padrão (sendo valores idênticos iguais em vários casos).

Os métodos **Bayesian** e **LARS lasso** empataram na pontuação geral, e ficaram muito próximos do 6º colocado.

Por fim, o **MLP** foi o último colocado, com a pior pontuação geral que se pode obter, indicando que não teve um bom desempenho em nenhuma das bases.

Levando em consideração que o IT-MUT não passou por um processo de *gridsearch*, enquanto todos os outros métodos tiveram seus hiper-parâmetros ajustados para cada base de dados individual, apesar de não ter sido o melhor, apresenta bons resultados. A grande vantagem do IT-MUT em relação ao XGBoost e ao KNN está no resultado que ele fornece - uma função simbólica - ao contrário de um conjunto grande de árvores complexas ou o uso de informações dos vizinhos próximos.

8 CONCLUSÃO

Neste trabalho foram aprofundados os estudos na aplicação da estrutura IT para regressão simbólica, entregando três algoritmos biologicamente inspirados nos processos de evolução, utilizando técnicas já bastante consolidadas na área de computação evolutiva para tal. O uso da estrutura IT tem como um dos principais objetivos realizar a tarefa de regressão simbólica retornando resultados mais interpretáveis do que os métodos atuais, tornando esse trabalho uma contribuição para sua aplicação.

Os algoritmos propostos tiveram seus parâmetros ajustados para melhorar seu desempenho, e foram então executados para bases de dados amplamente utilizadas no campo de aprendizagem de máquina, tendo seus resultados analisados entre si e depois comparados com técnicas bem sucedidas utilizadas atualmente.

Das três variantes introduzidas neste trabalho, a baseada em mutação (IT-MUT) se mostrou boa na minimização do erro de predição; a baseada em *crossover* (IT-CX) em minimização da complexidade dos resultados; e a híbrida (ITEA) apresentou um ponto intermediário entre as duas.

Os resultados obtidos são uma prova de que é possível realizar uma regressão simbólica em um espaço de busca restringido e ainda sim obter um desempenho competitivo, com a vantagem de ter uma maior interpretabilidade devido à maior simplicidade resultante da representação IT. Os três algoritmos apresentaram uma convergência rápida, levando em torno de 5 minutos para evoluir uma população pequena por poucas gerações e obter um resultado relevante.

8.1 TRABALHOS FUTUROS

O trabalho traz novos algoritmos baseados na estrutura IT e assim abre o caminho para o aprimoramento destes, podendo ser feito estudos nos diversos aspectos não tratados. Algumas possibilidades que se mostraram interessantes durante o desenvolver do projeto que merecem atenção estão listadas abaixo.

- Realizar um *gridsearch* nos hiper-parâmetros de forma mais abrangente, variando não só a taxa de *crossover* e o grupo de mutação como o número de gerações, grupo de funções de transformação, tamanho da população, limites do tamanho

da expressão e limites dos expoentes;

- Existem diversos métodos de seleção que poderiam ser testados para estudar sua influência na curva de convergência da população;
- Desenvolver uma heurística mais elaborada para a mutação no momento de selecionar o método, ponderando a chance de cada mutação ocorrer de acordo com o estado atual da expressão;
- O uso de outras técnicas de regressão para ajuste de coeficientes, como o Ridge ou LARS;
- O uso de heurísticas para a função de *fitness* que incorporem mais informações do que apenas o RMSE, mas também a complexidade da equação.

8.2 CONSIDERAÇÕES FINAIS

Apesar da limitação da estrutura, a não linearidade inserida pelos expoentes e pela função de transformação dá maior expressividade para os algoritmos, podendo não encontrar a equação que deu origem aos dados, mas uma que se aproxime de forma satisfatória - mas ainda existe espaço para muitas outras otimizações e ideias conceituais que possam melhorar o desempenho, a interpretabilidade e o tempo de execução.

Referências

ALDEIA, G. S. I.; FRANCA, F. O. de. Lightweight symbolic regression with the interaction - transformation representation. In: **2018 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2018. p. 1–8. Citado 4 vezes nas páginas 2, 4, 28 e 30.

BEYER, H.-G.; SCHWEFEL, H.-P. Evolution strategies – a comprehensive introduction. **Natural Computing**, v. 1, n. 1, p. 3–52, Mar 2002. ISSN 1572-9796. Disponível em: <https://doi.org/10.1023/A:1015059928466>. Citado 2 vezes nas páginas 17 e 20.

BROOKS, T. F.; POPE, D.; MARCOLINI, M. A. Airfoil self-noise and prediction. v. 1218, 08 1989. Citado na página 72.

COELLO, C. A. C. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. **Computer Methods in Applied Mechanics and Engineering**, v. 191, n. 11, p. 1245 – 1287, 2002. ISSN 0045-7825. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0045782501003231>. Citado 2 vezes nas páginas 23 e 24.

Cover, T.; Hart, P. Nearest neighbor pattern classification. **IEEE Transactions on Information Theory**, v. 13, n. 1, p. 21–27, January 1967. ISSN 0018-9448. Citado 2 vezes nas páginas 10 e 11.

FRANÇA, F. O. de. A greedy search tree heuristic for symbolic regression. **CoRR**, abs/1801.01807, 2018. Disponível em: <http://arxiv.org/abs/1801.01807>. Citado 8 vezes nas páginas , 2, 4, 9, 13, 23, 28 e 30.

FRIEDMAN, J. Greedy function approximation: A gradient boosting machine. **The Annals of Statistics**, v. 29, 11 2000. Citado na página 26.

HOLLAND, J. H. Book; Book/Illustrated. **Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence**. [S.l.]: Ann Arbor : University of Michigan Press, 1975. Includes index. ISBN 0472084607. Citado na página 17.

HORNBY, G. et al. Automated antenna design with evolutionary algorithms. **Collection of Technical Papers - Space 2006 Conference**, v. 1, 09 2006. Citado na página 24.

HORNIK, K. Approximation capabilities of multilayer feedforward networks. **Neural Networks**, v. 4, n. 2, p. 251 – 257, 1991. ISSN 0893-6080. Disponível em: <http:

//www.sciencedirect.com/science/article/pii/089360809190009T>. Citado na página 26.

JAMES, G. et al. **An Introduction to Statistical Learning: With Applications in R**. [S.l.]: Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370. Citado 6 vezes nas páginas 7, 8, 10, 11, 25 e 26.

KORNS, M. F. **Accuracy in Symbolic Regression**. New York, NY: Springer New York, 2011. 129–151 p. ISBN 978-1-4614-1770-5. Disponível em: <https://doi.org/10.1007/978-1-4614-1770-5_8>. Citado 4 vezes nas páginas 2, 14, 23 e 28.

KOTANCHEK, M.; SMITS, G.; KORDON, A. Industrial strength genetic programming. In: _____. **Genetic Programming Theory and Practice**. Boston, MA: Springer US, 2003. p. 239–255. ISBN 978-1-4419-8983-3. Disponível em: <https://doi.org/10.1007/978-1-4419-8983-3_15>. Citado 2 vezes nas páginas 4 e 24.

KOZA, J. R. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambridge, MA, USA: MIT Press, 1992. ISBN 0-262-11170-5. Citado 2 vezes nas páginas 14 e 18.

MARTINS, J. F. B. S. et al. Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. **CoRR**, abs/1804.06808, 2018. Disponível em: <http://arxiv.org/abs/1804.06808>. Citado na página 48.

ORZECHOWSKI, P.; CAVA, W. L.; MOORE, J. H. Where are we now?: A large benchmark study of recent symbolic regression methods. In: **Proceedings of the Genetic and Evolutionary Computation Conference**. New York, NY, USA: ACM, 2018. (GECCO '18), p. 1183–1190. ISBN 978-1-4503-5618-3. Disponível em: <http://doi.acm.org/10.1145/3205455.3205539>. Citado na página 2.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011. Citado na página 9.

SILVA, E. P. d. Uma breve história da teoria evolutiva. **História, Ciências, Saúde-Manguinhos**, scielo, v. 8, p. 671 – 987, 12 2001. ISSN 0104-5970. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-59702001000400009&nrm=iso>. Citado na página 16.

SIVANANDAM, S.; DEEPA, S. **Introduction to Genetic Algorithms**. Springer Berlin Heidelberg, 2007. ISBN 9783540731900. Disponível em: <https://books.google.com.br/books?id=wonrLjj2GagC>. Citado na página 10.

SMITS, G. F.; KOTANCHEK, M. Pareto-front exploitation in symbolic regression. In: _____. **Genetic Programming Theory and Practice II**. Boston, MA: Springer US,

2005. p. 283–299. ISBN 978-0-387-23254-6. Disponível em: <https://doi.org/10.1007/0-387-23254-0_17>. Citado 4 vezes nas páginas 10, 11, 13 e 30.

VIKHAR, P. A. Evolutionary algorithms: A critical review and its future prospects. In: **2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)**. [S.l.: s.n.], 2016. p. 261–265. Citado 5 vezes nas páginas 17, 18, 20, 22 e 24.

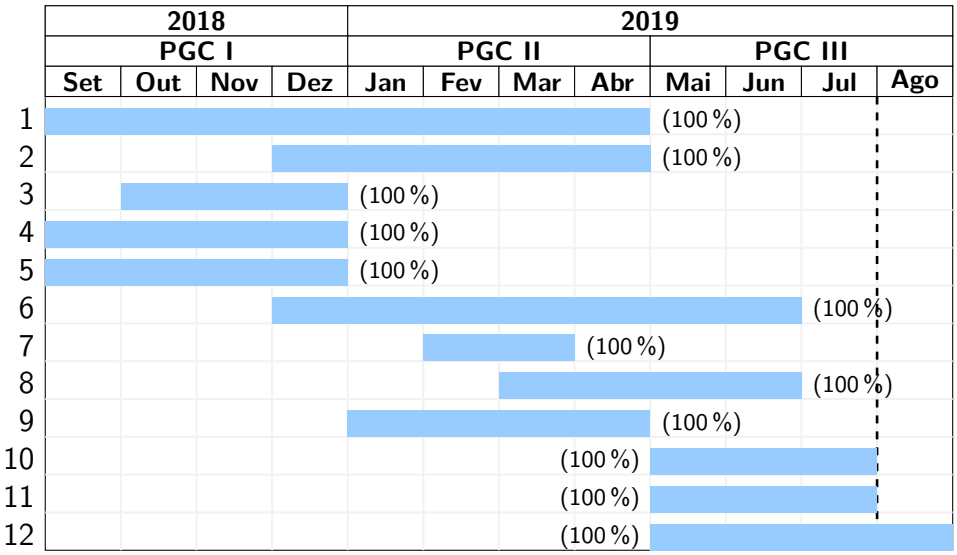
WILSON, D. G. et al. Evolving simple programs for playing atari games. In: **Proceedings of the Genetic and Evolutionary Computation Conference**. New York, NY, USA: ACM, 2018. (GECCO '18), p. 229–236. ISBN 978-1-4503-5618-3. Disponível em: <http://doi.acm.org/10.1145/3205455.3205578>. Citado na página 24.

Apêndices

APÊNDICE A – PLANEJAMENTO DE EXECUÇÃO

O Projeto de graduação em computação se divide em três disciplinas, que são cursadas ao longo de 1 ano.

Baseando-se nisso e nos objetivos estipulados, a execução do projeto é dividida em três grandes partes, permitindo que o progresso seja avaliado ao término de cada uma das disciplinas.



.33 Apresentação do projeto

Figura 19 – Diagrama de Gantt mostrando o planejamento de execução do projeto.

O diagrama (figura 19) ilustra a distribuição das tarefas à serem realizadas ao longo do desenvolvimento do projeto. De forma resumida, são apresentados abaixo cada etapa do projeto:

1. Revisão bibliográfica sobre o estado-da-arte dos algoritmos evolutivos, métodos de seleção e algoritmos de regressão simbólica atualmente.
2. Definição dos algoritmos evolutivos à serem estudados com a estrutura, baseados nos problemas conhecidos da regressão simbólica na literatura.
3. Desenvolvimento de uma metodologia para formulação e realização dos testes dos algoritmos.

4. Planejamento e escrita da proposta do projeto.
5. Implementação do algoritmo base à ser utilizado ao longo de todo o desenvolvimento em Python. A escolha da linguagem se deu pela diversidade de bibliotecas voltadas para ciência de dados e machine learning disponíveis atualmente - dessa forma, os algoritmos aqui propostos são executados e comparados com algoritmos no mesmo ambiente.
6. Implementação de diferentes algoritmos evolutivos baseados nas diferentes famílias apresentadas e estudadas. A implementação será feita em conjunto com a etapa 2.
7. Revisão bibliográfica para preparação do benchmark de testes, incluindo a seleção dos algoritmos que serão comparados.
8. Realização dos testes, coleta e análise dos resultados.
9. Escrita de um relatório parcial para avaliação.
10. Implementação (ou busca por implementações) de algoritmos de machine learning, regressão simbólica e algoritmos evolutivos levantados na etapa 7 para contrastar com os resultados obtidos.
11. Discussão dos resultados.
12. Escrita final do trabalho de graduação em computação.

APÊNDICE B – TESTE ESTATÍSTICO PARA O TAMANHO DAS EXPRESSÕES

O *p*-valor ajuda a determinar a significância da hipótese e dizer se a diferença entre as distribuições se relacionam ou se foi apenas acaso. Basicamente temos duas hipóteses que se contradizem: H_0, que diz que não há relação entre os dados (essa é considerada a hipótese nula), enquanto o que queremos é que exista tal relação (considerada a hipótese alternativa, H_a). O *p*-valor indica a probabilidade da ocorrência dos mesmos resultados observados se a hipótese nula for correta. Um baixo valor de *p*-valor é um forte indicador de que a hipótese nula está incorreta, mas não necessariamente indica que alternativa está correta. Costuma-se ser considerado um bom valor de p-valor contra a hipótese nula $p<0.05$. Para os dados que seguem, foi utilizado o teste de Wald para calcular o *p*-valor.

Sendo a hipótese nula a de que não há relação entre o tamanho médio das expressões encontrados para os algoritmos IT-MUT, IT-CX e ITEA, um teste de p-valor para todas as bases de dados podem sustentar essa hipótese ou apresentar indícios contra ela (não necessariamente confirmando a hipótese alternativa de que estão relacionados).

Tabela 15 – *p*-valor do tamanho das expressões para a base Airfoil.

Airfoil	IT-MUT	IT-CX
IT-CX	0.014*	
ITEA	0.002*	0.015*

Tabela 16 – *p*-valor do tamanho das expressões para a base Concrete.

Concrete	IT-MUT	IT-CX
IT-CX	0.104	
ITEA	0.069	0.269

O resultado desse teste estatístico favorece a hipótese alternativa, sugerindo que há sim uma relação entre utilizar apenas a mutação, apenas o *crossover* ou os

Tabela 17 – *p*-valor do tamanho das expressões para a base Energy Cooling.

Energy Cooling	IT-MUT	IT-CX
IT-CX	0.741	
ITEA	0.003*	0.003*

Tabela 18 – *p*-valor do tamanho das expressões para a base Energy Heating.

Energy Heating	IT-MUT	IT-CX
IT-CX	0.402	
ITEA	0.173	0.000*

Tabela 19 – *p*-valor do tamanho das expressões para a base Tower Data.

Tower Data	IT-MUT	IT-CX
IT-CX	0.709	
ITEA	0.668	0.001*

Tabela 20 – *p*-valor do tamanho das expressões para a base Wine Red.

Wine Red	IT-MUT	IT-CX
IT-CX	0.968	
ITEA	0.342	0.915

Tabela 21 – *p*-valor do tamanho das expressões para a base Wine White.

Wine White	IT-MUT	IT-CX
IT-CX	0.000*	
ITEA	0.000*	0.000*

Tabela 22 – *p*-valor do tamanho das expressões para a base Yacht.

Yacht	IT-MUT	IT-CX
IT-CX	0.623	
ITEA	0.025*	0.000*

dois combinados no tamanho médio obtido, sendo que a mutação favorece grandes expressões, o *crossover* favorece pequenas, e a combinação de ambos resulta em um ponto entre a performance dos dois operadores que apresenta um tamanho e um RMSE intermediário.

Ainda, da mesma forma que os três algoritmos aqui desenvolvidos apresentam uma relação entre seus operadores evolutivos utilizados e o tamanho médio da expressão obtido, o mesmo vale para diferentes taxas de *crossover*. Para suportar o argumento de que, apesar de apenas comparado com uma única taxa de *crossover*, será feito o teste estatístico entre todas as taxas, para mostrar que esse resultado pode ser estendido para todas as outras taxas, mas que foi utilizada apenas a de melhor desempenho para manter a coerência com o restante das análises. A figura 20 apresenta o tamanho médio em função da taxa de crossover para cada base de dados.

Tabela 23 – *p*-valor do tamanho das expressões para a base Airfoil para diferentes taxas de *crossover*.

Airfoil	taxa 0.10	taxa 0.20	taxa 0.30	taxa 0.40	taxa 0.50	taxa 0.60
taxa 0.20	0.000*					
taxa 0.30	0.000*	0.000*				
taxa 0.40	0.000*	0.002*	0.004*			
taxa 0.50	0.047*	0.043*	0.000*	0.196		
taxa 0.60	0.000*	0.002*	0.000*	0.033*	$0.044*$	
taxa 0.70	0.006*	0.145	0.004*	0.046*	0.103	0.033*

Tabela 24 – *p*-valor do tamanho das expressões para a base Concrete para diferentes taxas de *crossover*.

Concrete	taxa 0.10	taxa 0.20	taxa 0.30	taxa 0.40	taxa 0.50	taxa 0.60
taxa 0.20	0.858					
taxa 0.30	0.005*	0.022*				
taxa 0.40	0.031*	0.682	0.261*			
taxa 0.50	0.024*	0.470	0.062*	0.376		
taxa 0.60	0.487	0.286	0.727	0.664	0.455	
taxa 0.70	0.378	0.964	0.983	0.001*	0.173	0.658

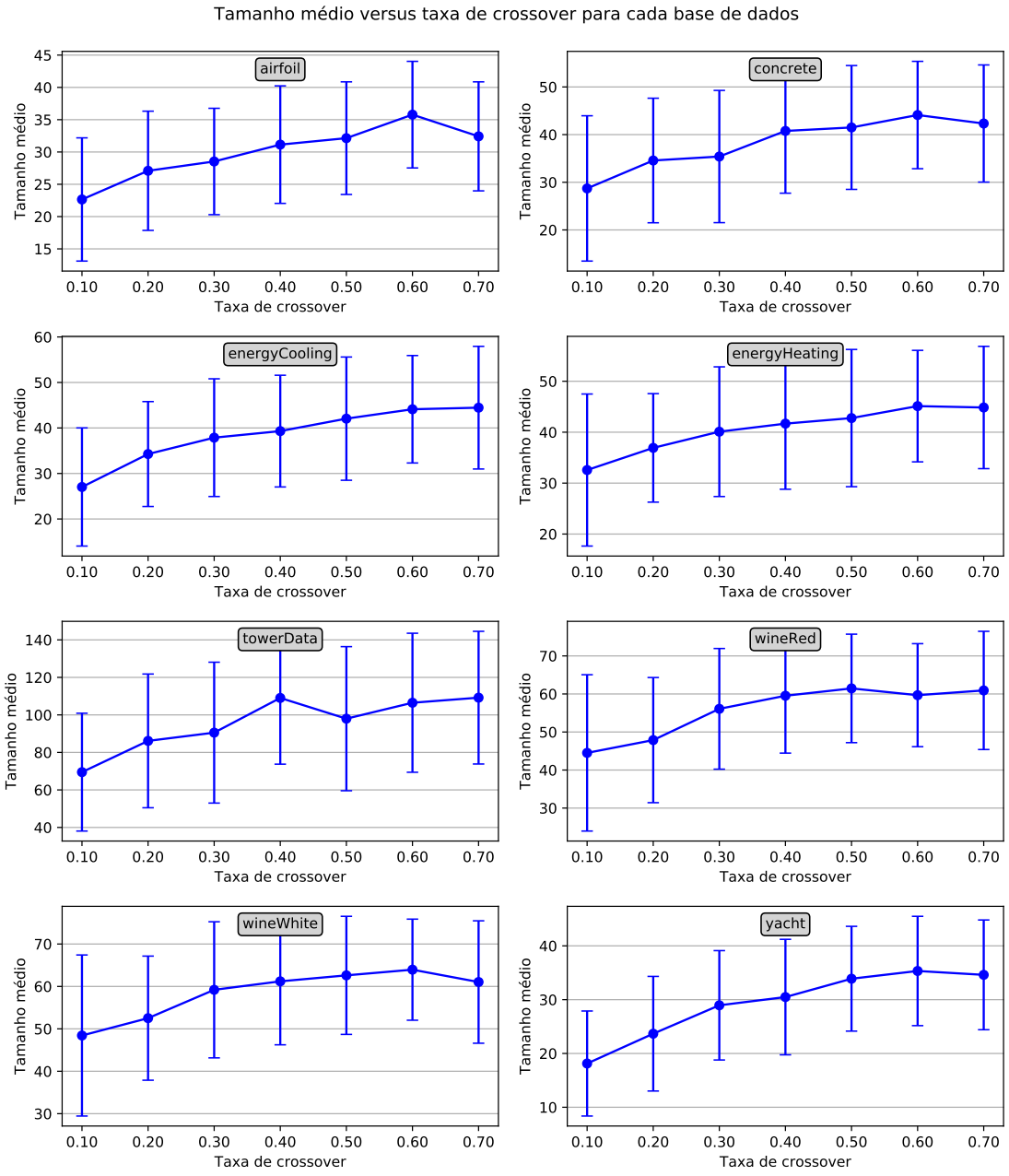


Figura 20 – Variação do tamanho médio das melhores expressões para cada base de dados com o aumento da taxa de crossover.

Tabela 25 – *p*-valor do tamanho das expressões para a base Energy Cooling para diferentes taxas de *crossover*.

Energy Cooling	taxa 0.10	taxa 0.20	taxa 0.30	taxa 0.40	taxa 0.50	taxa 0.60
taxa 0.20	0.407					
taxa 0.30	0.084*	0.262				
taxa 0.40	0.144	0.940	0.863			
taxa 0.50	0.290	0.585	0.036*	0.174		
taxa 0.60	0.174	0.668	0.202	0.585	0.180	
taxa 0.70	0.643	0.683	0.245	0.176	0.796	0.193

Tabela 26 – *p*-valor do tamanho das expressões para a base Energy Heating para diferentes taxas de *crossover*.

Energy Heating	taxa 0.10	taxa 0.20	taxa 0.30	taxa 0.40	taxa 0.50	taxa 0.60
taxa 0.20	0.008*					
taxa 0.30	0.630	0.021*				
taxa 0.40	0.147	0.046*	0.007*			
taxa 0.50	0.041*	0.239	0.508	0.373		
taxa 0.60	0.910	0.741	0.133	0.490	0.340	
taxa 0.70	0.790	0.507	0.734	0.518	0.137	0.513

Tabela 27 – *p*-valor do tamanho das expressões para a base Tower Data para diferentes taxas de *crossover*.

Tower Data	taxa 0.10	taxa 0.20	taxa 0.30	taxa 0.40	taxa 0.50	taxa 0.60
taxa 0.20	0.000*					
taxa 0.30	0.000*	0.000*				
taxa 0.40	0.000*	0.000*	0.000*			
taxa 0.50	0.000*	0.000*	0.000*	0.000*		
taxa 0.60	0.000*	0.000*	0.000*	0.000*	0.000*	
taxa 0.70	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*

Tabela 28 – *p*-valor do tamanho das expressões para a base Wine Red para diferentes taxas de *crossover*.

Wine Red	taxa 0.10	taxa 0.20	taxa 0.30	taxa 0.40	taxa 0.50	taxa 0.60
taxa 0.20	0.000*					
taxa 0.30	0.000*	0.000*				
taxa 0.40	0.000*	0.000*	0.000*			
taxa 0.50	0.000*	0.000*	0.000*	0.000*		
taxa 0.60	0.000*	0.000*	0.000*	0.000*	0.000*	
taxa 0.70	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*

Tabela 29 – *p*-valor do tamanho das expressões para a base Wine White para diferentes taxas de *crossover*.

Wine White	taxa 0.10	taxa 0.20	taxa 0.30	taxa 0.40	taxa 0.50	taxa 0.60
taxa 0.20	0.000*					
taxa 0.30	0.000*	0.000*				
taxa 0.40	0.000*	0.000*	0.000*			
taxa 0.50	0.000*	0.000*	0.000*	0.000*		
taxa 0.60	0.000*	0.000*	0.000*	0.000*	0.000*	
taxa 0.70	0.004*	0.000*	0.011*	0.001*	0.003*	0.005*

Tabela 30 – *p*-valor do tamanho das expressões para a base Yacht para diferentes taxas de *crossover*.

Yacht	taxa 0.10	taxa 0.20	taxa 0.30	taxa 0.40	taxa 0.50	taxa 0.60
taxa 0.20	0.000*					
taxa 0.30	0.046*	0.006*				
taxa 0.40	0.000*	0.000*	0.011*			
taxa 0.50	0.019*	0.024*	0.013*	0.006*		
taxa 0.60	0.001*	0.000*	0.283	0.001*	0.001*	
taxa 0.70	0.000*	0.000*	0.003*	0.003*	0.002*	0.000*

é possível ver que a ocorrência de p-valores menores que 0.05 é grande (utilizando 3 casas decimais de significância), o que indica que há forte indício matemático de que as distribuições estão relacionadas.

APÊNDICE C – TESTE ESTATÍSTICO PARA O ERRO DE PREDIÇÃO

Tabela 31 – *p*-valor do RMSE das melhores expressões encontradas para a base Airfoil.

Airfoil	ITEA-MUT	ITEA-CX
ITEA-CX	0.014*	
ITEA	0.002*	0.015*

Tabela 32 – *p*-valor do RMSE das melhores expressões encontradas para a base Concrete.

Concrete	ITEA-MUT	ITEA-CX
ITEA-CX	0.104	
ITEA	0.069*	0.269

Tabela 33 – *p*-valor do RMSE das melhores expressões encontradas para a base Energy Cooling.

Energy Cooling	ITEA-MUT	ITEA-CX
ITEA-CX	0.741*	
ITEA	0.003*	0.003*

Tabela 34 – *p*-valor do RMSE das melhores expressões encontradas para a base Energy Heating.

Energy Heating	ITEA-MUT	ITEA-CX
ITEA-CX	0.402	
ITEA	0.173	0.000*

Tabela 35 – *p*-valor do RMSE das melhores expressões encontradas para a base Tower Data.

Tower Data	ITEA-MUT	ITEA-CX
ITEA-CX	0.709	
ITEA	0.668	0.001*

Tabela 36 – *p*-valor do RMSE das melhores expressões encontradas para a base Wine Red.

Wine Red	ITEA-MUT	ITEA-CX
ITEA-CX	0.968	
ITEA	0.342	0.915

Tabela 37 – *p*-valor do RMSE das melhores expressões encontradas para a base Wine White.

Wine White	ITEA-MUT	ITEA-CX
ITEA-CX	0.000*	
ITEA	0.000*	0.000*

Tabela 38 – *p*-valor do RMSE das melhores expressões encontradas para a base Yacht.

Yacht	ITEA-MUT	ITEA-CX
ITEA-CX	0.623	
ITEA	0.025*	0.000*

APÊNDICE D – TESTE ESTATÍSTICO PARA PERFORMANCE COMPARADO COM TÉCNICAS ATUAIS

Tabela 39 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Airfoil.

Airfoil	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost
Bayesian	0.016*						
LARS lasso	0.017*	0.000*					
MLP	0.000*	0.000*	0.000*				
OLS	0.017*	0.000*	0.000*	0.000*			
Ridge	0.017*	0.000*	0.000*	0.000*	0.000*		
XGBoost	0.643	0.021*	0.019*	0.421	0.019*	0.021*	
KNN	0.232	0.000*	0.000*	0.015*	0.000*	0.000*	0.000*

Tabela 40 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Concrete.

Concrete	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost
Bayesian	0.017*						
LARS lasso	0.020*	0.000*					
MLP	0.226	0.519	0.515				
OLS	0.015*	0.000*	0.000*	0.510			
Ridge	0.017*	0.000*	0.000*	0.512	0.000*		
XGBoost	0.000*	0.028*	0.024*	0.205	0.038*	0.03*	
KNN	0.000*	0.000*	0.000*	0.465	0.000*	0.000*	0.001*

Tabela 41 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Energy Cooling.

Energy Cooling	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost
Bayesian	0.137						
LARS lasso	0.191	0.000*					
MLP	0.035*	0.000*	0.000*				
OLS	0.115	0.000*	0.000*	0.000*			
Ridge	0.195	0.000*	0.000*	0.000*	0.000*		
XGBoost	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*	
KNN	0.000*	0.001*	0.000*	0.276	0.001*	0.000*	0.209

Tabela 42 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Energy Heating.

Energy Heating	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost
Bayesian	0.195						
LARS lasso	0.195	0.000*					
MLP	0.135	0.116	0.111				
OLS	0.207	0.000*	0.000*	0.098			
Ridge	0.195	0.000*	0.000*	0.112	0.000*		
XGBoost	0.007*	0.000*	0.000*	0.012*	0.000*	0.000*	
KNN	0.058	0.000*	0.000*	0.022*	0.000*	0.000*	0.000*

Tabela 43 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Tower Data.

Tower Data	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost
Bayesian	0.001*						
LARS lasso	0.001*	0.000*					
MLP	0.488	0.998	0.994				
OLS	0.001*	0.000*	0.000*	0.997			
Ridge	0.001*	0.000*	0.000*	0.989	0.000*		
XGBoost	0.122	0.000*	0.000*	0.914	0.000*	0.000*	
KNN	0.000*	0.000*	0.000*	0.375	0.000*	0.000*	0.562

Tabela 44 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Wine Red.

Wine Red	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost
Bayesian	0.992						
LARS lasso	0.983	0.000*					
MLP	0.590	0.000*	0.000*				
OLS	0.991	0.000*	0.000*	0.000*			
Ridge	0.989	0.000*	0.000*	0.000*	0.000*		
XGBoost	0.684	0.000*	0.000*	0.000*	0.000*	0.000*	
KNN	0.808	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*

Tabela 45 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Wine White.

Wine White	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost
Bayesian	0.000*						
LARS lasso	0.000*	0.000*					
MLP	0.000*	0.000*	0.000*				
OLS	0.000*	0.000*	0.000*	0.000*			
Ridge	0.000*	0.000*	0.000*	0.000*	0.000*		
XGBoost	0.244	0.923	0.850	0.006*	0.897	0.991	
KNN	0.343	0.070*	0.060*	0.219	0.081	0.070*	0.025*

Tabela 46 – *p*-valor do RMSE das melhores soluções do algoritmo IT-MUT e os algoritmos comparados para a base Yacht.

Yacht	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost
Bayesian	0.047*						
LARS lasso	0.044*	0.000*					
MLP	0.132	0.000*	0.000*				
OLS	0.044*	0.000*	0.000*	0.000*			
Ridge	0.049*	0.000*	0.000*	0.000*	0.000*		
XGBoost	0.156	0.000*	0.000*	0.000*	0.000*	0.000*	
KNN	0.373	0.000*	0.000*	0.001*	0.000*	0.000*	0.000*

Dataset	IT-MUT	Bayesian	LARS lasso	MLP	OLS	Ridge	XGBoost	KNN
Airfoil	2.758 ± 0.433 3	4.819 ± 0.156 4	4.819 ± 0.156 4	8.893 ± 0.482 8	4.819 ± 0.156 4	4.82 ± 0.156 7	1.612 ± 0.131 1	2.684 ± 0.146 2
Concrete	6.624 ± 0.31 2	10.446 ± 0.432 5	10.447 ± 0.447 6	22.128 ± 1.398 8	10.445 ± 0.438 4	10.447 ± 0.442 6	4.3 ± 0.285 1	8.713 ± 0.748 3
Energy Cooling	1.692 ± 0.158 2	3.215 ± 0.1 5	3.215 ± 0.098 5	13.212 ± 0.599 8	3.211 ± 0.1 4	3.215 ± 0.099 5	0.857 ± 0.139 1	2.531 ± 0.054 3
Energy Heating	0.687 ± 0.129 2	2.942 ± 0.167 5	2.942 ± 0.168 5	14.122 ± 0.241 8	2.935 ± 0.164 4	2.942 ± 0.168 5	0.356 ± 0.049 1	2.281 ± 0.147 3
Tower Data	29.295 ± 2.043 3	30.666 ± 3.141 7	30.44 ± 3.271 5	120.821 ± 7.731 8	30.432 ± 3.274 4	30.542 ± 3.223 6	15.665 ± 0.698 2	14.064 ± 0.5 1
Wine Red	0.676 ± 0.255 7	0.651 ± 0.026 2	0.651 ± 0.026 2	0.948 ± 0.037 8	0.652 ± 0.025 5	0.651 ± 0.025 2	0.613 ± 0.036 1	0.657 ± 0.037 6
Wine White	0.745 ± 0.052 3	0.754 ± 0.011 4	0.756 ± 0.011 7	1.064 ± 0.065 8	0.754 ± 0.012 4	0.754 ± 0.011 4	0.667 ± 0.006 1	0.719 ± 0.012 2
Yacht	0.904 ± 0.691 2	8.998 ± 0.622 6	8.963 ± 0.612 4	20.757 ± 1.656 8	9.012 ± 0.606 7	8.996 ± 0.626 5	0.672 ± 0.086 1	8.952 ± 1.622 3
Pontuação geral Classificação final	3.0 3	4.75 5	4.75 5	8.0 7	4.5 4	5.0 6	1.125 1	2.875 2