UNIVERSIDADE FEDERAL DO TOCANTINS
CÂMPUS UNIVERSITÁRIO DE PALMAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

OTIMIZAÇÃO DE HIPERPARÂMETROS EM ALGORITMOS DE ÁRVORE DE DECISÃO UTILIZANDO COMPUTAÇÃO EVOLUTIVA

FELIPE REIS MACEDO BARBOSA

PALMAS (TO)
2018

FELIPE REIS MACEDO BARBOSA

OTIMIZAÇÃO DE HIPERPARÂMETROS EM ALGORITMOS DE ÁRVORE DE DECISÃO UTILIZANDO COMPUTAÇÃO EVOLUTIVA

Trabalho de Conclusão de Curso II apresentado à Universidade Federal do Tocantins para obtenção do título de Bacharel em Ciência da Computação, sob a orientação do(a) Prof.(a) Dr. Rafael Lima de Carvalho.

Orientador: Dr. Rafael Lima de Carvalho

PALMAS (TO)

2018

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da Universidade Federal do Tocantins

B238o Barbosa, Felipe Reis Macedo.

Otimização de hiperparâmetros em algoritmos de árvore de decisão utilizando computação evolutiva. / Felipe Reis Macedo Barbosa. – Palmas, TO, 2018.

52 f.

Monografia Graduação - Universidade Federal do Tocantins – Câmpus Universitário de Palmas - Curso de Ciências da Computação, 2018.

Orientador: Rafael Lima de Carvalho

1. Aprendizado de máquina. 2. Otimização de hiperparâmetros. 3. Árvores de decisão. 4. Algoritmo genético. I. Título

CDD 004

Elaborado pelo sistema de geração automatica de ficha catalográfica da UFT com os dados fornecidos pelo(a) autor(a).

FELIPE REIS MACEDO BARBOSA

OTIMIZAÇÃO DE HIPERPARÂMETROS EM ALGORITMOS DE ÁRVORE DE DECISÃO UTILIZANDO COMPUTAÇÃO EVOLUTIVA

Trabalho de Conclusão de Curso II apresentado à UFT – Universidade Federal do Tocantins – Câmpus Universitário de Palmas, Curso de Ciência da Computação foi avaliado para a obtenção do título de Bacharel e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Data de aprovação: 7 / 12 / 2018

Banca Examinadora:

Prof. Dr. Rafael Lima de Carvalho

Prof. Dr. Warley Gramacho da Silva

Prof. Me. Tiago da Silva Almeida

RESUMO

Alguns algoritmos em aprendizado de máquina são parametrizáveis, ou seja, permitem a configuração de parâmetros de maneira a aumentar o desempenho na tarefa utilizada. Na maioria dos casos, estes parâmetros são encontrados empiricamente pelo desenvolvedor. Outra abordagem é utilizar alguma técnica de otimização para encontrar um conjunto otimizado de parâmetros. Este projeto tem por objetivo a aplicação dos algoritmos evolutivos, Algoritmo Genético (AG), *Fluid Genetic Algorithm* (FGA) e *Genetic Algorithm using Theory of Chaos* (GATC) para otimizar a busca de hiperparâmetros em algoritmos de árvores de decisão. Este trabalho apresenta alguns resultados satisfatórios dentro do conjunto de dados testados, onde o algoritmo *Classification and Regression Trees* (CART) foi utilizado como algoritmo classificador para os testes. Nestes, as árvores de decisão geradas a partir dos valores padrão dos hiperparâmetros são comparados com os otimizados pela abordagem proposta. Buscou-se otimizar a acurácia e o tamanho final da árvore gerada, o que foram otimizadas com sucesso pelos algoritmos propostos.

Palavra-chave: Aprendizado de máquina. Otimização de hiperparâmetros. Árvores de decisão. Algoritmo genético.

ABSTRACT

Some algorithms in machine learning are parameterizable, they allow the configuration of parameters in order to increase the performance in some tasks. In most cases, these parameters are empirically found by the developer. Another approach is to use some optimization technique to find an optimized set of parameters. The aim of this project is the application of evolutionary algorithms, Genetic Algorithm (GA), Fluid Genetic Algorithm (FGA) and Genetic Algorithm using Theory of Chaos (GATC) to optimize the search for hyperparameters in decision tree algorithms. This work presents some satisfactory results within the data set tested, where the Classification and Regression Trees (CART) algorithm was used as a classifier algorithm for the tests. In these, the decision trees generated from the default values of the hyperparameters are compared with those optimized by the proposed approach. We has tried to optimize the accuracy and final size of the generated tree, which were successfully optimized by the proposed algorithms.

Keywords: Machine learning. Hyperparameter optimization. Decision trees. Genetic algorithm.

ABREVIATURAS

AG Algoritmo Genético

AM Aprendizado de Maquina

CART *Classification and Regression Trees*

CE Computação Evolucionária

DBN *Deep Belief Network*

DEMO *Differential Evolution for Multiobjective Optimization*

DL *Deep Learning*

DNNs *Deep Neural Networks*

EDA *Estimation of Distribution Algorithm*

FGA *Fluid Genetic Algorithm*

FN Falsos Negativos

FP Falsos Positivos

GATC *Genetic Algorithm using Theory of Chaos*

GS *Grid Search*

ID3 *Iterative Dichotomiser 3*

ML *Machine Learning*

MS *Manual Search*

NN *Neural Networks*

PSO *Particle Swarm Optimization*

RS *Random Search*

SVR *Support Vector Regression*

VN Verdadeiros Negativos

VP Verdadeiros Positivos

LISTA DE FIGURAS

Figura 1 – Exemplo de árvore de decisão verificando a qualidade de um atributo para classificar um nó 18
Figura 2 – Exemplo de uma Árvore de Decisão 19
Figura 3 – Estrutura básica de um algoritmo genético 25
Figura 4 – Cruzamentos de um e dois pontos de corte 27
Figura 5 – Exemplo de cromossomo do algoritmo FGA 28
Figura 6 – Exemplo de Cruzamento de um FGA 29
Figura 7 – *Esquema do Genetic Algorithm using Theory of Chaos* 30
Figura 8 – Exemplo de representação cromossomial do GATC. 31
Figura 9 – Exemplo de cruzamento do GATC 32

Figura 10 – Representação binária do cromossomo resultante 41
Figura 11 – Representação do indivíduo cromossomo do algoritmo FGA 41
Figura 12 – Representação do Cromossomo do algoritmo GATC 42

Figura 13 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Steel Plates 43
Figura 14 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Iris 44
Figura 15 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Wine 44
Figura 16 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Breast Cancer 45
Figura 17 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Digits 45
Figura 18 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Isolet 46
Figura 19 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Madelon . . 46

Figura 20 – Melhores indivíduos em 30 populações diferente para o algoritmo AG 47
Figura 21 – Melhores indivíduos em 30 populações diferente para o algoritmo FGA 47
Figura 22 – Melhores indivíduos em 30 populações diferente para o algoritmo GATC 48
Figura 23 – Comparação do Fitness obtido entre os algoritmos AG, FGA e GATC, e a utilização do algoritmo CART com os valores default 48
Figura 24 – Comparação da Profundidade da árvore final gerada, obtida entre os algoritmos AG, FGA e GATC, e a utilização do algoritmo CART com os valores default 49

LISTA DE TABELAS

Tabela 1 – Descrição de tipos de testes 17
Tabela 2 – Descrição variáveis da equação entropia para base de dados binárias 19
Tabela 3 – Exemplo de Matriz de confusão 23
Tabela 4 – Descrição dos componentes da equação Born-an-Individual 28
Tabela 5 – Processo de modificação da máscara utilizando a função caótica. . . 33

Tabela 6 – Parâmetros do Algoritmo CART. 37
Tabela 7 – Detalhes da base de dados *Stell Plates* 37
Tabela 8 – Discriminação das classes e seus registros da base de dados *Stell Plates* 38
Tabela 9 – Detalhes da base de dados *Breast Cancer* 38
Tabela 10 – Discriminação das classes e seus registros da base de dados *Breast Cancer* 38
Tabela 11 – Detalhes da base de dados *Wine* 38
Tabela 12 – Discriminação das classes e seus registros da base de dados *Wine* . . 38
Tabela 13 – Detalhes da base de dados *Iris* 39
Tabela 14 – Discriminação das classes e seus registros da base de dados *Iris* . . . 39
Tabela 15 – Detalhes da base de dados *Isolet* 39
Tabela 16 – Discriminação das classes e seus registros da base de dados *Isolet* . . 40
Tabela 17 – Detalhes da base de dados *Madelon* 40
Tabela 18 – Discriminação das classes e seus registros da base de dados *Madelon* 40

Tabela 19 – Descrição dos parâmetros utilizados no desenvolvimento do algoritmo FGA 43
Tabela 20 – Taxa de acerto dos algoritmos FGA, AG, GATC e o algortimo CART com seus valores padrões 46

SUMÁRIO

1 INTRODUÇÃO 14

1.1 Estrutura do trabalho 15

2 FUNDAMENTAÇÃO TEÓRICA 16

2.1 *Machine Learning* 16

2.2 O problema da otimização de parâmetros 16

2.3 Algoritmos para geração de árvores de decisão 17

2.3.1 ID3 18

2.3.2 C4.5 20

2.3.3 CART 21

2.4 Métricas de Avaliação 23

2.4.1 Acurácia 23

2.4.2 *Precision* 24

2.4.3 *Recall* 24

2.4.4 F1 *Score* 24

2.5 Computação Evolucionária 24

2.6 Algoritmos Genéticos 24

2.6.1 Inicialização 25

2.6.2 Função de Avaliação 26

2.6.3 Seleção dos Pais 26

2.6.4 Operadores Genéticos 27

2.7 *Fluid Genetic Algorithm* 27

2.7.1 Representação Cromossomial 28

2.7.2 *Born An Individual Function* 28

2.7.3 Inicialização, Avaliação e Seleção 29

2.7.4 Cruzamento 29
2.8 *Genetic Algorithm using Theory of Chaos* 30
2.8.1 Estrutura do GATC 30
2.8.1.1 Função Caótica 30
2.8.1.2 Representação cromossomial 30
2.8.1.3 Inicialização, Avaliação e Seleção 31
2.8.1.4 Cruzamento e Mutação 31
3 TRABALHOS RELACIONADOS 34
4 METODOLOGIA 36
4.1 Ferramentas 36
4.1.1 Python 36
4.1.2 *scikit-learn* 36
4.2 Bases de dados 37
4.2.1 *Steel Plates* 37
4.2.2 *Breast Cancer* 37
4.2.3 *Wine* 38
4.2.4 *Iris* 39
4.2.5 *Isolet* 39
4.2.6 *Madelon* 39
4.3 Desenvolvimento e avaliação 40
4.3.1 AG 41
4.3.2 FGA 41
4.3.3 GATC 41
4.3.4 Função Objetivo 42
5 ANÁLISES E RESULTADOS 43
6 CONSIDERAÇÕES FINAIS 50
6.1 Trabalhos Futuros 50
REFERÊNCIAS 51

1 INTRODUÇÃO

Machine Learning (ML), ou Aprendizado de Maquina (AM), é um campo de pesquisa que estuda como desenvolver programas que melhorem automaticamente com sua experiência, baseando-se em conceitos e resultados de vários campos da pesquisa, incluindo estatísticas, inteligencia artificial, teoria da informação, complexidade computacional, biologia e teoria de controle (MITCHELL et al., 1997).

Muitos algoritmos de ML possuem parâmetros cujos valores são encontrados empiricamente pelo desenvolvedor, esses valores influenciam diretamente no processo de aquisição, podendo gerar modelos diferentes (ROSSI, 2009). Geralmente, os algoritmos de ML não apresentam resultados ótimos sem terem sido configurados inicialmente, com isso, é necessário uma otimização na etapa de configuração dos parâmetros do algoritmo.

De acordo com Mitchell et al. (1997), existem técnicas de aprendizado, onde o aprendizado por hábito é a mais simples. Porém, estratégias de aprendizado por indução são as mais utilizadas entre os algoritmos de ML. Assim, o aprendizado indutivo pode ser classificado como supervisionado e não-supervisionado. No aprendizado não-supervisionado, o algoritmo não tem um conhecimento dos rótulos das possíveis classes a serem reconhecidas, trabalhando por meio de extrações de padrões para realizar agrupamento dos seus exemplos. No aprendizado supervisionado, cada registro da base de dados possui um conjunto de atributos, cujo os algoritmos têm por entrada, para treinar e inferir a classe a que esta instância pertence.

Para o ajuste de parâmetros tem-se uma entrada de dados para um determinado problema de classificação, o qual pode ser descrito pelo par ordenado (X, y), onde X é um vetor de atributos e y é a classe a qual aquele vetor pertence. Por exemplo, o algoritmo J48 (vide Seção 2.3.2) é um algoritmo utilizado para criar uma árvore de decisão, e a partir da árvore gerada é possível obter um resultado, que determina a classe a qual aquela entrada pertence. Esse algoritmo tem parâmetros que são configurados antes da fase de conhecimento da base de dados, com isso, a árvore gerada pode ser diferente de acordo com os valores adotados para o seus parâmetros.

De acordo Luo (2016), o problema de configuração automática de hiperparâmetros tem relação com vários campos que excedem a computação, onde todas essas áreas compartilham de um critério de qualidade específico comparando diferentes objetos, tendo como objetivo selecionar o objeto que melhor representa o conjunto de dados. Assim, modelos estatísticos e metodologia de otimização são implementados em ML com foco de selecionar os valores dos hiperparâmetros.

Uma grande gama dos problemas necessitam de uma configuração dos parâmetros para obtenção de um resultado mais satisfatório, com isso, é consumido muito tempo e

em alguns casos é necessário que um especialista estude a base de dados e o algoritmo utilizado, para assim, realizar a melhor configuração possível dos parâmetros do algoritmo.

A maior parte de problemas reais na área de otimização estão ligados a obtenção de diversas metas que devem ser atingidas simultaneamente. Normalmente existem conflitos e não existe solução única que otimize todas ao mesmo tempo. Segundo Ticona (2003), problemas dessa natureza são chamados de problemas de otimização multiobjetivo por envolverem minimizações ou maximizações simultâneas de um conjunto de objetivos satisfazendo a um conjunto de restrições.

Muitos pesquisadores estudam como contornar esse problema de configuração automática de parâmetros, buscando otimizar essa etapa. Uma abordagem é a utilização de algoritmo genético para ajuste de parâmetros. Segundo Yuan (2012), os resultados experimentais utilizando AG's na otimização de parâmetros do algoritmo *Support Vector Regression* (SVR) para obter melhor precisão em um planejamento orçamentário para volume de vendas, foram mais satisfatórios do que modelos de previsão tradicionais e redes neurais artificiais.

Em relação a algoritmos genéticos com recentes modificações estruturais, o trabalho proposto em Borges (2017) realizou uma revisão sistemática onde resultou na seleção de cinco algoritmos genéticos modificados.

Diante disto, neste trabalho propõe-se a utilização dos algoritmos genéticos adaptados levantados pelo trabalho em Borges (2017), aplicando-os em algoritmos de aprendizado de maquina parametrizados com o objetivo de otimizar seus parâmetros e verificar o impacto desta busca em seus respectivos desempenhos.

1.1 Estrutura do trabalho

O Capítulo 2 aborda os conceitos gerais necessários para essa pesquisa, sendo eles, *Machine Learning*, problematização da otimização na etapa de configuração dos hiperparâmetros, árvore de decisão e os principais algoritmos relacionados a essa pesquisa (ID3, C4.5 e CART) e abordagem sobre algoritmos genéticos e descrição dos algoritmos FGA e GATC. O Capítulo 3 lista uma série de pesquisas que abordaram sobre o tema de otimização e demonstram sobre a solução aplicada nesse trabalho. O Capítulo 4 descreve as principais ferramentas utilizadas para desenvolvimento desse trabalho, as bases de dados e os respectivos algoritmos, além das métricas envolvidas. O Capítulo 5 apresenta os resultados obtidos pelos algoritmos AG, FGA e GATC. O Capítulo 6 ressalta a descrição do problema, conclui a abordagem dos resultados e propõe o que será realizado em um trabalho futuro.

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo serão abordados os conceitos necessários para fundamentação dessa pesquisa, onde explica-se com mais detalhes o problema de configuração automática de hiperparâmetros dentro do campo de ML. Aqui também, descreve-se sobre árvore de decisão e computação evolucionária, com foco em algoritmos genéticos e suas variações.

2.1 *Machine Learning*

Machine Learning é um campo multidisciplinar, baseando-se em resultados de inteligência artificial, probabilidade, estatística, teoria da complexidade computacional, teoria de controle, teoria da informação, filosofia, psicologia, neurobiologia e outros campos. Mitchell definiu ML como: "Diz-se que um programa de computador aprende com a experiência E com relação a alguma classe de tarefas T e medida de desempenho P, se seu desempenho em tarefas em T, como medido por P, melhora com a experiência E" (MITCHELL et al., 1997).

Dentro da visão de ML, a classificação pode ser definida como a busca por uma função que mapeie os registros para suas respectivas classes (KOBLAR, 2012). Com isso, algum dos algoritmos classificadores possuem parâmetros para dinamizar o processo de classificação de acordo a base de dados.

2.2 O problema da otimização de parâmetros

Alguns algoritmos de ML para classificação possuem parâmetros que precisam ser configurados antes de executá-los para obtenção de um resultado mais satisfatório, tais parâmetros são comumente denotados como hiperparâmetros.

> "Existem muitos algoritmos de aprendizado de máquina, a maioria dos quais são complexos. Cada algoritmo de aprendizado de máquina possui dois tipos de parâmetros de modelo: parâmetros comuns que são automaticamente otimizados ou aprendidos em uma fase de treinamento de modelo e hiperparâmetros que são tipicamente definidos pelo usuário manualmente antes de um modelo de aprendizado de máquina ser treinado"(LUO, 2016).

O algoritmo pode ser utilizado com seus hiperparâmetros nos valores *default*, porém, realizando a configuração dos hiperparâmetros, é possível obter melhores resultados em relação a este. De acordo Probst, Bischl e Boulesteix (2018), o ajuste de hiperparâmetros, também chamado de otimização de hiperparâmetros, pode ser definido como

o processo de encontrar boas configurações de hiperparâmetros de um algoritmo de ML para um conjunto de dados específico.

Para realização da otimização dos hiperparâmetros, é necessário escolher uma estratégia de ajuste, onde os valores candidatos podem ser avaliados e escolhidos aleatoriamente a partir do espaço do hiperparâmetro ou usando estratégias de amostragem não aleatória. Após selecionar a estratégia de ajuste, é preciso definir quais parâmetros devem ser configurados.

O trabalho realizado por Probst, Bischl e Boulesteix (2018) demonstra a importância da configuração dos hiperparâmetros dos algoritmos de ML, onde sugerem que, as máquinas de vetores suporte, rede elástica, *gradient boosting* e árvore de decisão necessitam de uma boa configuração, já os algoritmos *random forest* e o *k-nearest neighbor* são menos sintonizáveis.

2.3 Algoritmos para geração de árvores de decisão

É uma das abordagens de aprendizado mais simples e de maior sucesso. Uma árvore de decisão tem como entrada um objeto ou um conjunto de atributos e como saída uma resposta, essa é dada a partir de uma sequência de testes.

Basicamente uma Árvore de Decisão permite dividir recursivamente um conjunto de dados de treino até que cada divisão forneça uma classificação para a instância. As Árvores de Decisão (ver Figura 2) consistem em nós que formam uma árvore, o que significa que, existe um nó-raiz que não tem ramos de entrada, ao contrário dos restantes nós. Cada nó intermédio específica um teste para o atributo, e cada ramo descendente desse nó corresponde ao valor possível desse atributo. Este conjunto de regras é seguido até ser atingido o nó-terminal ou folha (ROKACH; MAIMON, 2005), (ARONSON; LIANG; TURBAN, 2005).

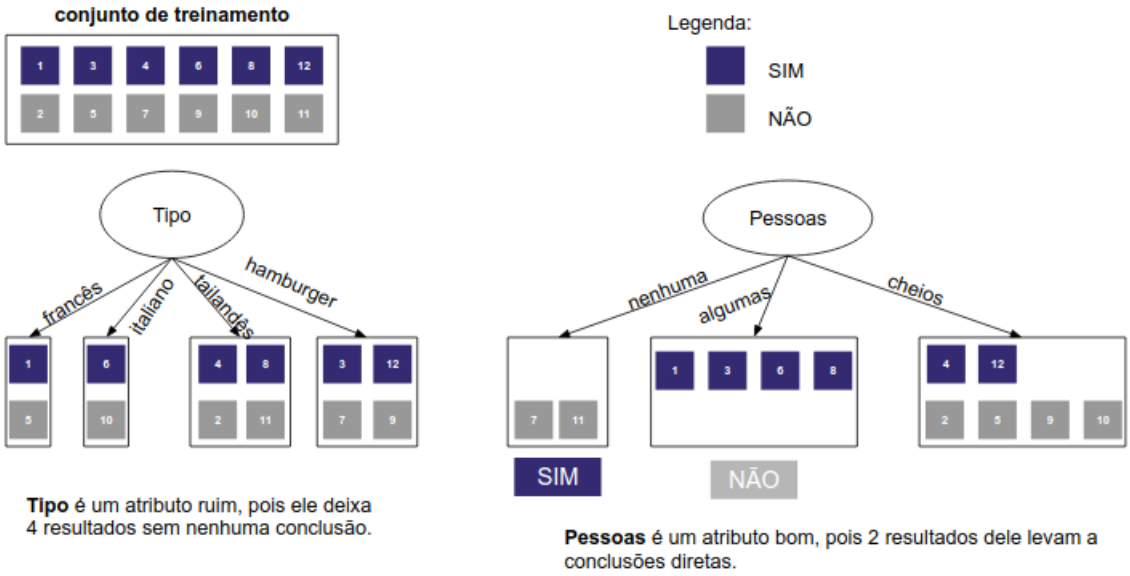
Tabela 1 – Descrição de tipos de testes

Tipo	Descrição	Exemplo
Exemplos Positivos	São aqueles que levam a uma resposta positiva	ẗvai esperar"= SIM
Exemplos Negativos	São aqueles que levam a uma resposta negativa	ẗvai esperar"= NÃO

É possível gerar uma árvore de decisão a partir de um conjunto de exemplos demonstrados na Tabela 1, com isso, a ideia inicial do algoritmo é testar os atributos mais importantes primeiro, aqueles que fazem maior diferença no processo classificatório, como exemplo a Figura 1.

O funcionamento da árvore de decisão e dado como:

Figura 1 – Exemplo de árvore de decisão verificando a qualidade de um atributo para classificar um nó



- Enquanto existirem exemplos positivos e negativos, deve-se escolher o melhor atributo para dividi-los;
- Se todos os exemplos restantes forem positivos ou todos negativos, então podemos responder SIM ou NÃO;
- Se não existirem exemplos restantes, retorna um valor padrão calculado a partir da classificação da maioria dos atributos do nó pai;
- Se não existirem atributo restantes, mas ainda existirem exemplos positivos e negativos temos um problema.

Quando não existem atributos restantes, mas ainda existem exemplos positivos e negativos, significa que alguns dos dados estão incorretos. Esse problema também ocorre quando os atributos não dão informação suficiente para descrever a situação completamente, ou quando o domínio é realmente não-determinístico. Uma possível solução para esse problema é a utilização de uma votação majoritária. Dentro dessa abordagem, exitem alguns algoritmos de árvores de decisão, sendo eles, *Iterative Dichotomiser 3* (ID3), C4.5, *Classification and Regression Trees* (CART).

2.3.1 ID3

O algoritmo ID3 foi desenvolvido por Ross Quinlan, na Universidade de Sydnei, Austrália em 1979. Ele tem como objetivo identificar os melhores atributos para construção da árvore de decisão mais eficiente, buscando inicialmente qual atributo deve ser

Figura 2 – Exemplo de uma Árvore de Decisão

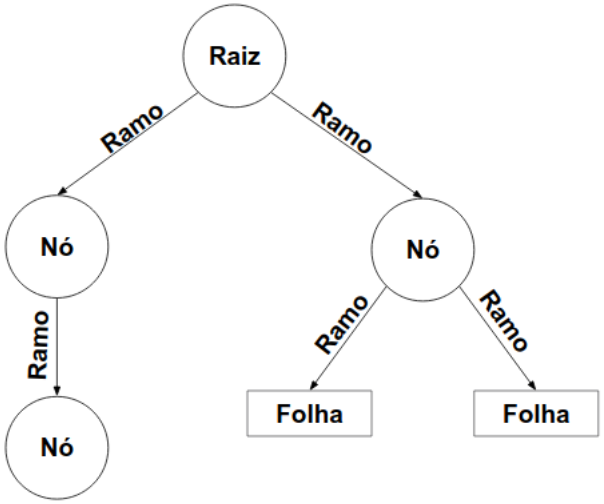


Tabela 2 – Descrição variáveis da equação entropia para base de dados binárias

$p_{\oplus}$	é a proporção de exemplos positivos em S.
$p_{\ominus}$	é a proporção de exemplos negativos em S.

testado na raiz da árvore. Para responder esta questão, cada atributo é avaliado para determinar quão bem ele sozinho classifica os exemplos de treinamento. Com isso, é realizado o cálculo da entropia, que, de acordo Mitchell et al. (1997), é necessária para definir precisamente o ganho de informação, que caracteriza a impureza de uma coleção arbitrária de exemplos. A fórmula da entropia é descrita a seguir, e suas varáveis estão descritas na Tabela 2.

$$Entropia\,(S) = -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus} \tag{1}$$

O algoritmo começa com o atributo de maior ganho de informação como o nó raiz, a cada iteração do algoritmo, a partir do conjunto de atributos não usados, é calculado a entropia e o ganho de informação desse atributo, para este nó da árvore é selecionado o atributo com maior ganho de informação. O algoritmo continua a recorrer em cada subconjunto, considerando apenas atributos nunca selecionados antes.

Caso tenha uma base de dados em que seus atributos possuam 'n' valores diferentes, sendo n $>$ 2 e P_i é a proporção do número de elementos referente ao valor pertencente a este atributo, para o número de elementos no conjunto total, a fórmula da entropia é dada por,

$$Entropia\,(S) = \sum_{i=1}^{n} -p_i log_2 p_i \tag{2}$$

Após o cálculo da entropia calcula-se o ganho de informação, que mede a efetividade em que o atributo classifica um conjunto de treinamento. A fórmula é dada a partir do cálculo de entropia do atributo classificador, menos o somatório da proporção do número de elementos em S_v para o número de elementos no conjunto total, multiplicado pela entropia do valor referente no atributo atual, a fórmula é dada por,

$$GI\left(S,A\right) = E\left(S\right) - \sum_{v \in Valores(A)} \frac{S_v}{S} E\left(S_v\right) \tag{3}$$

Como por exemplo, se temos uma base de dados com 14 registros e seu atributo classificador contém dois valores, SIM e NÃO, e essa base está dividida em 9 casos classificados em SIM e 5 classificados em NÃO e um atributo com valores Forte e Fraco, onde o atributo Forte contém 8 registros e Fraco contém 6 registros, o cálculo da entropia e ganho de informação é dado por,

$$\begin{gathered} E\left([9,5]\right) = -\tfrac{9}{14} log_2 \tfrac{9}{14} - \tfrac{5}{14} log_2 \tfrac{5}{14} = 0,940 \\ GI\left(S,A\right) = E\left(S\right) - \tfrac{8}{14} E\left(S_{fraco}\right) - \tfrac{6}{14} E\left(S_{forte}\right) \\ E\left(S_{fraco}\right) = -\tfrac{6}{8} log_2 \tfrac{6}{8} - \tfrac{2}{8} log_2 \tfrac{2}{8} = 0,811 \\ E\left(S_{forte}\right) = -\tfrac{3}{6} log_2 \tfrac{3}{6} - \tfrac{3}{6} log_2 \tfrac{3}{6} = 1,00 \\ GI\left(S,A\right) = 0,94 - \tfrac{8}{14} 0.811 - \tfrac{6}{14} 1.00 = 0,048 \end{gathered} \tag{4}$$

O Exemplo reduz muito pouco o nível de incerteza (entropia), logo, esse atributo não é bom para a raiz da árvore.

O algoritmo ID3 é pioneiro em indução de árvore de decisão, com isso, ele possui algumas limitações, tais como, só lidar com atributos discretos, ele também não trata valores desconhecidos, logo, todos os exemplos devem ter seus valores conhecidos para todos os atributos e não possui método pós poda, que consiste em realizar uma busca de baixo pra cima, transformando os nós que não aparentam nenhum ganho significativo em nós folhas. Com isso, alguns algoritmos surgiram como evolução do ID3, estes serão descritos nas seções seguintes.

2.3.2 C4.5

O algoritmo C4.5, também conhecido como J48, versão desenvolvida em JAVA pelo grupo WEKA. Ele é uma extensão do algoritmo ID3, essa evolução permitiu que pudessem ser lidos tanto atributos discretos, quanto atributos contínuos, ignorando também os valores desconhecidos. Há duas diferenças significativas, sendo que, este algoritmo possui método pós poda, e não aplica-se mais o ganho de informação como método de seleção do melhor atributo, gerando assim, árvores menos complexas passando a utilizar a razão

do ganho por,

$$RazaoDoGanho\,(S_v) = \frac{G(S)}{E(S_v)} \tag{5}$$

2.3.3 CART

O algoritmo *Classification and Regression Trees* foi desenvolvido no trabalho de Breiman Jerome Friedman e Olshen (1984). Os conceitos e o funcionamento do algoritmo CART apresentados são exclusivamente extraídos do livro Breiman Jerome Friedman e Olshen (1984) . Ele é semelhante ao C4.5, porém a diferença entre eles é, que o algoritmo CART suporta variáveis de destino numéricas (regressão) e não calcula conjuntos de regras.

A árvore de decisão resultante a partir do a algoritmo CART sempre é binária, o critério utilizado para calcular a impureza de um nó é o índice Gini, o qual mede a heterogeneidade dos dados. O cálculo é dado por,

$$G = 1 - \sum_{i=1}^{c} p_i^2 \tag{6}$$

Onde P_i é a frequência relativa de cada classe em cada nó e C é o número de classes.

Como citado nas seções anteriores sobre árvore de decisão e seus algoritmos, é preciso identificar qual dos atributos realiza a melhor divisão de uma possível árvore de decisão final. Com isso, na etapa de seleção de atributo onde é verificado a impureza do nó, pode-se definir o ganho de uma divisão como sendo a diferença entre o nó pai e a soma desta para com seus filhos, assim, obtém-se dois novos subconjuntos G_L e G_R (cálculos do índice Gini para os lados esquerdos e direitos) com proporções P_L e P_R (cálculos das proporções para os lados esquerdos e direitos). O Ganho da impureza é dado por,

$$\Delta G(S,T) = G(T) - ((P_L * G_L) + (P_R * G_R)) \tag{7}$$

O algoritmo CART considera os critérios Entropia e Gini para selecionar melhor divisão dos dados. O critério Entropia foi descrito na seção 2.3.1 e o critério Gini será descrito a seguir.

Observando a fórmula (6) e supondo que, por exemplo, um problema com 3 classes sendo a quantidade total de dados T = 150, C = 3 e $P_i = (C_i$ / T), onde para uma variável numérica esses casos T fossem distribuídos nas classes de maneira que, sejam (100|25|25), $C_1 = 100$, $C_2 = 25$ e $C_3 = 25$, as probabilidades das classes seriam:

$$p_1\left(\frac{c_1}{t}\right) = \frac{100}{150} = 0,666 \tag{8}$$

$$p_2\left(\frac{c_2}{t}\right) = \frac{25}{150} = 0,166 \tag{9}$$

$$p_3\left(\frac{c_3}{t}\right) = \frac{25}{150} = 0,166 \tag{10}$$

Após a realização da distribuição de probabilidades para cada classe, é possível calcular a impureza deste nó, neste caso o resultado do critério Gini, por,

$$G(S,T) = 1 - ((0,666)^2 + 2 * (0,166)^2) \tag{11}$$

$$= 1 - (0,443 + 0,055) \tag{12}$$

$$= 1 - 0,498 \tag{13}$$

$$= 0,502 \tag{14}$$

Nesse exemplo o Gini é G(S,T) = 0,502. Supondo que o cálculo apresentado pertence ao nó pai da árvore, e que foi subdividido em dois nós filhos, sendo o nó esquerdo contém 100 casos (100|0|0) e o no direito contém 50 casos (0|25|25), o cálculo do Gini para os nós filhos é dado por:

$$G(T_L) = 1 - \left(\left(\frac{100}{100}\right)^2 + 2 * \left(\frac{0}{100}\right)^2\right) \quad G(T_R) = 1 - \left(\left(\frac{0}{50}\right)^2 + 2 * \left(\frac{25}{50}\right)^2\right) \tag{15}$$

$$G(T_L) = 1 - 1 \qquad G(T_R) = 1 - 0,5 \tag{16}$$

$$G(T_L) = 0 \qquad G(T_R) = 0,5 \tag{17}$$

Tendo-se o resultado de Gini para os nós pai e filhos, é possível encontrar o ganho de impureza da divisão por,

$$\Delta G(S,T) = 0,502 - \left(\left(\frac{100}{150} * 0\right) + \left(\frac{50}{150} * 0,5\right)\right) \tag{18}$$

$$= 0,502 - 0,166 \tag{19}$$

$$= 0,336 \tag{20}$$

Esse cálculo é realizado para todas as variáveis do problema, escolhendo para divisão do nó a que possuir maior índice Gini. Após localizar o melhor ponto de divisão e, de fato, realizar a divisão do nó, caso não exista mais ganho para dividi-lo, faz-se uma associação de uma classe folha. Para associar uma determinada classe à folha o cálculo é

dado por,

$$\underset{c}{Max}\,(p_c) = \underset{c}{Max}\frac{N_c}{N} \tag{21}$$

Onde N é número total de exemplos na folha e N_c é o número de exemplos da classe c na folha. Após o término da construção da árvore, é adotada uma técnica chamada poda com o objetivo de remoção ou corte dos ramos e sub-árvores que não são relevantes para a resolução do problema.

2.4 Métricas de Avaliação

Após a execução do algoritmo classificatório é necessário que o desempenho do mesmo seja avaliado, com isso, existem algumas maneiras e métodos para realizar essa avaliação. Muitas métricas avaliativas utilizam a matriz de confusão, essa consiste numa tabela que demonstra o desempenho do classificador. É possível extrair quatro valores básicos de uma matriz de confusão, sendo eles, os verdadeiros positivos (V_P), os falsos positivos (F_P), os falsos negativos (F_N) e os verdadeiros negativos (V_N). Tendo a Tabela 3 como exemplo os 4 valores citados anteriormente são:

- A célula com 60 registros classificados corretamente são os V_P, pois o classificador o identificou como sim (positivo) e essa predição está correta (verdadeiro);
- Os 25 registros registros que estão classificados como sim (positivo) porém a predição está errada (falso) são os F_P;
- Os 11 registros classificados como não (negativo) mas a predição está errada (falso) são os F_N;
- Os 190 registros classificados como não (negativo) e a predição está correta (verdadeiro) são os V_N.

Tabela 3 – Exemplo de Matriz de confusão

	SIM	NÃO	TOTAL
SIM	**60**	25	85
NÃO	11	**190**	201
TOTAL	71	215	285

Algumas métricas serão descritas a seguir.

2.4.1 Acurácia

A acurácia é uma métrica simples que consiste em verificar a porcentagem de acerto do classificador, por exemplo, tendo 500 amostras dentro de uma base de dados e o classificador acertou 430 casos, obtém-se como acurácia final (430/500) * 100 = 86%

2.4.2 *Precision*

Essa métrica tem como objetivo identificar qual a porcentagem de acerto do algoritmo para as amostras classificadas positivamente. A fórmula é dada a seguir:

$$Precision \ = \frac{VP}{VP+FP} \ = \frac{60}{60+25} \ = 0,7059 \tag{22}$$

2.4.3 *Recall*

O *recall* é uma métrica semelhante a *precision*, onde realiza a função oposta, verificando a taxa de VP em relação a FN. A fórmula é dada a seguir:

$$Recall \ = \frac{VP}{VP+FN} \ = \frac{60}{60+11} \ = 0,8451 \tag{23}$$

2.4.4 F1 *Score*

F1 *score* depende das métricas, *recall* e *precision*, sendo assim, ela consiste em uma média harmônica entre elas, buscando um balanço entre as duas, dada por,

$$F1 \ = 2 * \frac{Precision*Recall}{Precision+Recall} \ = 2 * \frac{0,7059*0,8451}{0,7059+0,8451} \ = 0,7693 \tag{24}$$

2.5 Computação Evolucionária

Desde a década de 1950, vários cientistas da computação estudam os sistemas evolutivos com a ideia de que a evolução poderia ser usada como uma ferramenta para otimização de problemas. A ideia em todos esses sistemas era desenvolver uma população de soluções candidatas a um determinado problema, usando operadores inspirados pela variação genética natural e pela seleção natural (MITCHELL, 1998).

A Computação Evolucionária (CE) é uma metodologia baseada na teoria da evolução de Darwin, onde os algoritmos evolutivos se comportam da mesma maneira que o corpo humano, tendo uma população de cromossomos, os quais são as possíveis soluções de um determinado problema, para realização de reprodução, mutação e variação na competição de novos indivíduos. Dentro da CE estão algumas técnicas como, Estratégias Evolutivas, Programação Evolucionária e Algoritmos Genéticos (AG), sendo esse último o foco desse trabalho.

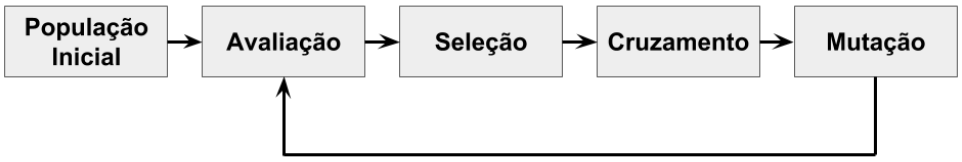
2.6 Algoritmos Genéticos

Os AG's são uma técnica de busca, tendo o seu foco analisar as possíveis soluções e encontrar um resultado perto da solução ótima quase sempre sem necessitar da interferência humana (LINDEN, 2006).

Algoritmos genéticos têm sido utilizados pela ciência e engenharia como algoritmos adaptativos para resolução de problemas práticos e também, como modelos computacionais de sistemas evolutivos naturais. Esta abordagem simples e acessível permite que os desenvolvedores implementem e experimentem algoritmos genéticos por conta própria. Ele se concentra em um conjunto de tópicos importantes e interessantes - particularmente no aprendizado de máquina e modelagem científica (MITCHELL, 1998).

A estrutura básica de um algoritmo genético normalmente é composta por populações de cromossomos, seleção de acordo com a aptidão, cruzamento para produzir novos filhos, e mutação aleatória destes novos filhos. Esses elementos estão representados na Figura 3.

Figura 3 – Estrutura básica de um algoritmo genético



Para compreender o funcionamento dos AG's faz-se necessário realizar uma analogia à evolução das espécies. Assim, o AG trabalha da seguinte forma:

- Inicialmente é gerado a população inicial, que consiste em uma quantidade n de cromossomos, os quais são possíveis soluções para o problema;
- Durante o processo evolutivo, cada indivíduo na população recebe uma nota referente ao valor obtido na função de avaliação;
- Uma porcentagem dos indivíduos são mais adaptados, fazendo com que os mais fracos sejam descartados;
- Os membros mantidos pela seleção podem sofrer modificações em suas características fundamentais por meio de cruzamentos *(crossover)*, mutações ou recombinação genética gerando descendentes para a próxima geração;
- Este processo, chamado geração, é repetido até que uma solução satisfatória seja encontrada ou o critério de parada seja alcançado.

2.6.1 Inicialização

O AG é inicializado com uma população, cujo a quantidade é relativa de acordo com a necessidade do desenvolvedor. Um cromossomo é um indivíduo que carrega as

características do problema proposto, sendo uma possível solução do mesmo. A representação desse cromossomo é completamente arbitrária, podendo ser formado por, uma sequência de bits, valores numéricos reais, lista de regras, entre outras, de acordo com o problema definido e da escolha do programador. Normalmente essa população inicial é gerada de maneira totalmente aleatória, implicando assim o conceito de diversidade presente no AG.

2.6.2 Função de Avaliação

Também conhecida como *fitness*, a função de avaliação é uma maneira utilizada pelos AG's para determinar a qualidade e quão relevante aquele indivíduo é dentro da população, pois assim, é possível obter as melhores soluções para o problema. O *fitness* deve ser determinado com bastante cautela, sendo necessário aplicar todo conhecimento sobre o problema no seu desenvolvimento. Pois, essa etapa é fundamental para bom funcionamento do AG.

2.6.3 Seleção dos Pais

O método de seleção de pais deve tentar simular o mecanismo de seleção natural que atua sobre as espécies biológicas, no qual, dois ou mais cromossomos são selecionados para gerar novos descendentes, que possivelmente serão mais aptos que seus geradores. Normalmente os cromossomos com função de avaliação mais alta possuem privilégios no momento de seleção, pois, espera-se deles uma geração mais apta, porém, para bom funcionamento de um AG é necessário analisar toda população cromossomial existente, sem desprezar os indivíduos menos aptos, com isso, existem técnicas de seleção, algumas delas serão descritas a seguir:

- Roleta: Nesta técnica, cria-se uma roleta virtual na qual cada cromossomo recebe um pedaço proporcional a sua avaliação. Deste modo, os cromossomos que possuírem um *fitness* maior ocuparão uma maior fração da roleta, enquanto que os cromossomos com *fitness* inferior ocuparão menores frações. Assim, roda-se a roleta para sortear os indivíduos que serão selecionados como pais de um novo indivíduo;

- Elitista: Essa técnica trabalha em conjunto com outras técnicas, mas ela garante que, primeiramente seja selecionado o indivíduo com *fitness* mais alto, para posteriormente selecionar os demais pares;

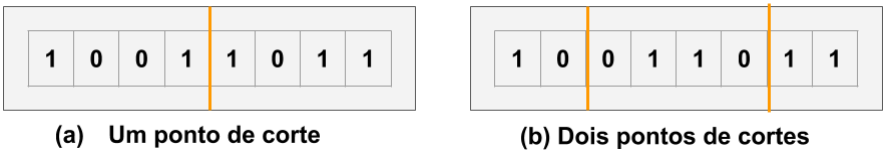
- Torneio: Os indivíduos são separados em N grupos, onde o indivíduo com maior *fitness* dentro do grupo vence a disputa, o mesmo é selecionado como pai. Normalmente os grupos são formados aleatoriamente ou são pré-selecionados usando algum outro métodos de seleção.

2.6.4 Operadores Genéticos

Cruzamento

É inspirado na recombinação biológica, realizando a troca de material genético entre os pais na geração dos filhos. O processo ocorre após a seleção dos pais, onde é necessário determinar a quantidade de pontos de corte para realizar o cruzamento, e assim, gerar um novo indivíduo. De acordo a Figura 4, temos dois exemplos de cortes, tendo o caso (a) com um ponto de corte, onde o primeiro filho recebe os genes mais a esquerda do primeiro pai e os genes mais a direita do segundo pai, esse processo é realizado de maneira inversa com o segundo filho, já no caso (b) onde ocorrem dois pontos de corte, fica a critério do desenvolvedor a metodologia de atribuição dos componentes para os novos filhos gerados.

Figura 4 – Cruzamentos de um e dois pontos de corte



(a) Um ponto de corte (b) Dois pontos de cortes

Mutação

Também sendo um operador inspirado na biologia, após obter-se a combinação dos novos indivíduos filhos, cada gene do novo cromossomo gerado é avaliado individualmente podendo sofrer alteração de acordo com o problema ou escolha do desenvolvedor. Esse operador é utilizado para implicar uma diversidade maior na população de cromossomos e evitar que haja uma convergência no AG. Porém é possível gerar um indivíduo que se distancie da solução do problema, sendo necessário uma certa cautela em relação ao uso desse operador.

2.7 *Fluid Genetic Algorithm*

Neste artigo, Jafari-Marandi e Smith (2017) apresentam um AG modificado e melhorado, denominado *Fluid Genetic Algorithm* - FGA. De acordo com os autores, em relação ao AG padrão, o FGA possui uma melhor taxa de sucesso, melhor controle de convergência e também pode ser aplicado a uma gama mais ampla de problemas, incluindo problemas multiobjetivo e multinível.

O algoritmo FGA possui duas grandes diferenças em relação ao AG padrão, a primeira diferença consiste no fato de não termos cromossomo e indivíduos como uma única entidade, com isso os autores afirmam que, biologicamente, o FGA está mais próximo da realidade do que acontece no mundo genético, onde cada cromossomo pode ter muitos

indivíduos diferentes associados a ele. Assim, no FGA um indivíduo é associado aleatoriamente a um cromossomo, atribuído pela função *born-an-individual*. A segunda diferença é que nessa nova proposta de AG não é necessário o operador mutação, pois, o algoritmo fornece uma diversidade populacional inteligente.

2.7.1 Representação Cromossomial

O cromossomo não é mais um conjunto de genes que representa uma possível solução para o problema, nessa nova abordagem cada um dos valores na célula contém um valor real normalizado entre 0 e 1, o qual representa uma probabilidade de o indivíduo ter o valor de 1 em vez de 0. Por exemplo, o indivíduo que será atribuído ao cromossomo na Figura 5 tem apenas 41% de chance de ter 1 em vez de zero na primeira célula da esquerda.

Figura 5 – Exemplo de cromossomo do algoritmo FGA

0.41	0.26	0.99	0.21	0.63	0.39	0.85

2.7.2 *Born An Individual Function*

Nessa função, cada cromossomo irá gerar um indivíduo com base no valor contido em cada célula. A função também possui a taxa de aprendizado global e a geração de *blue print* para produzir indivíduos. A geração *blue print* é, na verdade, um cromossomo com células preenchidas com o valor médio de todo o cromossomo em cada iteração, ela é iniciada com valor 0,5 para cada célula do cromossomo. A fórmula é dada por,

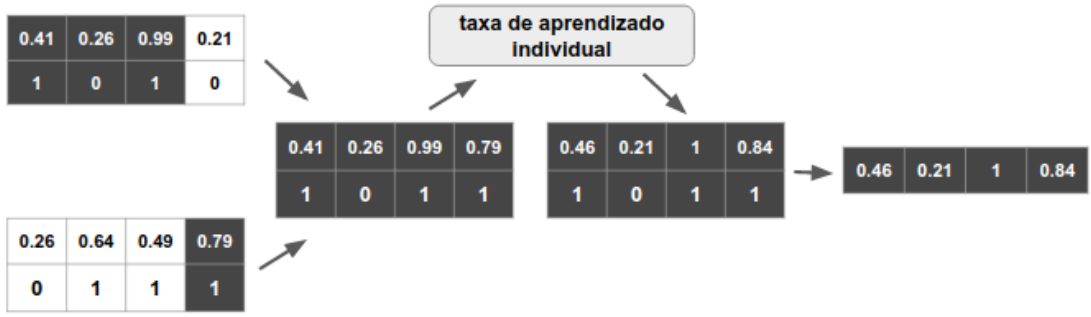
$$\begin{cases} N_g * PVB_i + (1 - N_g) * PVC_i < N_{dr} & , EPV_i = N_{dr} \\ N_g * PVB_i + (1 - N_g) * PVC_i > 1 - N_{dr} & , EPV_i = 1 - N_{dr} \\ \text{outro caso} & , EPV_i = N_g * PVB_i + (1 - N_g) * PVC_i \end{cases} \tag{25}$$

Tabela 4 – Descrição dos componentes da equação Born-an-Individual

Variáveis	**Descrição**
N_g	Taxa de aprendizado global
PVC_i	Valor percentual do cromossomo na posição i
PVB_i	Valor percentual do *blue print* na posição i
N_{dr}	Taxa de diversidade
EPV_i	Valor de probabilidade efetiva

Os parâmetros da função são, o cromossomo gerado anteriormente e a geração *blue print*, para serem aplicados na equação. A Tabela 4 descreve os componentes dessa função.

Figura 6 – Exemplo de Cruzamento de um FGA



A função limita o cromossomo ao ótimo local, sendo ele entre 0 e 1, ainda é utilizada a taxa de diversidade e a taxa de aprendizado global, as quais são duas constantes definidas pelo desenvolvedor.

2.7.3 Inicialização, Avaliação e Seleção

A população é iniciada de maneira aleatória e seus cromossomos, assim como o *blue print*, são instanciados com 0,5 em cada célula contida. A primeira geração não é afetada pela função *Born an individual*. As funções de avaliação e de seleção são semelhantes a de um AG, onde avalia-se o indivíduo gerado pela função *Born an individual* e atribui-se o *fitness*, e a técnica de seleção fica a critério do desenvolvedor.

2.7.4 Cruzamento

O cruzamento é realizado semelhante a um AG, utilizando-se a ideia de pontos de corte, onde o cromossomo filho recebe parte do cromossomo e do indivíduo de um pai e outra parte de um segundo pai, como detalhado na Figura 6. O que há de diferente em relação a um AG é a utilização da variável taxa de aprendizado individual, pois é verificado se o valor do indivíduo na posição referente a célula do gene no cromossomo for 1 é incrementado a taxa no novo cromossomo, se for 0, esse valor é decrementado. Por exemplo, na primeira célula da esquerda, o indivíduo referente a ela é 1, então é incrementado 0,05, valor usado pelos autores no artigo, na célula referente do cromossomo resultante, e na segunda célula da esquerda, como o indivíduo referente a ela é 0, é decrementado 0,05 da célula referente do cromossomo resultante.

Após a realização desse operador, é gerado um novo cromossomo, assim, verifica-se o critério de parada do algoritmo, caso ainda haja execução esse cromossomo gerado é levado a função *born an individual* para receber o indivíduo referente a sua lista de cromossomos e é adicionado na população para repetição desse ciclo.

2.8 *Genetic Algorithm using Theory of Chaos*

Neste artigo, Snaselova e Zboril (2015) propõem um AG modificado, baseado na teoria do caos de Edward Norton Lorenz, o *Genetic Algorithm using Theory of Chaos* (GATC). De acordo com os autores, a ciência lida com problemas não-lineares cujos comportamentos são impossíveis de prever ou controlar, como fluxo de fluido turbulento, padrões climáticos globais, ritmos cardíacos, sequências de codificação de DNA e assim por diante. Com isso, é possível considerar que os processos evolutivos também podem ser entendidos no sentido de caos. As adaptações do AG em suas etapas, serão apresentadas a seguir.

2.8.1 Estrutura do GATC

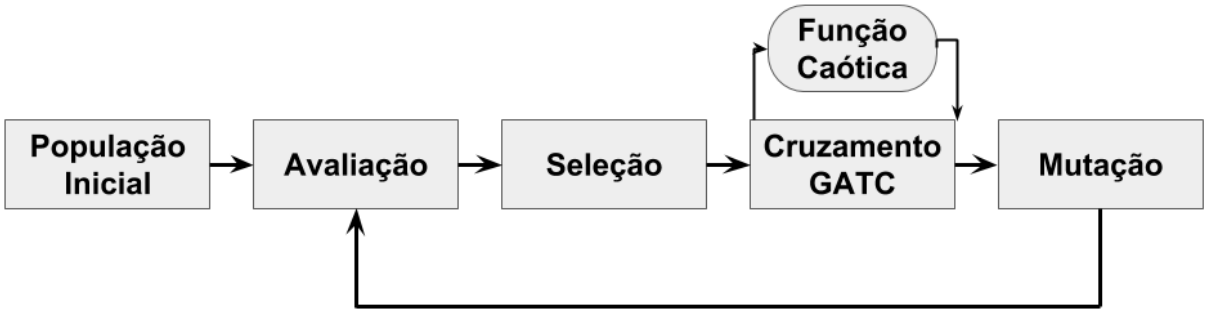
As principais diferenças do GATC para um AG padrão são, o processo de cruzamento que é aplicado pelo uso da função caótica, e a representação cromossomial que também sofre alteração na nova metodologia apresentada pelos autores, o esquema do GATC está representado pela Figura 7.

2.8.1.1 Função Caótica

Os autores utilizaram a função da teoria do caos para conduzir a evolução do algoritmo. A fórmula representa uma modificação na representação, onde os autores embutiram no cromossomo o parâmetro λ e a máscara uniforme de cruzamento,

$$z_{n+1} = \lambda z_n(1 - z_n) \tag{26}$$

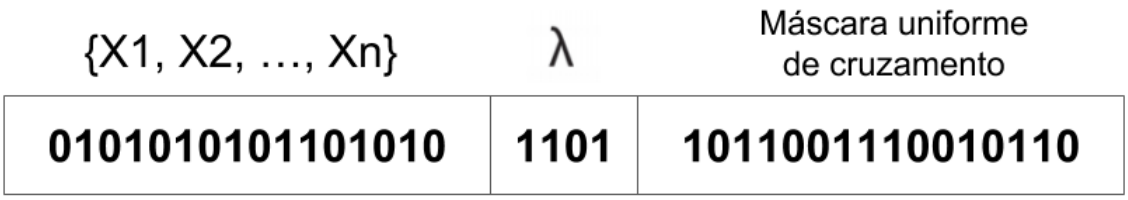
Figura 7 – *Esquema do Genetic Algorithm using Theory of Chaos*



2.8.1.2 Representação cromossomial

O cromossomo é representado pela Figura 8, este é divido em três partes: a solução do problema representada de forma binária, o valor da λ e a máscara de cruzamento uniforme.

Figura 8 – Exemplo de representação cromossomial do GATC.



O algoritmo GATC trabalha com problemas multimodais, sendo assim, a primeira parte do cromossomo compõe a solução do problema. A segunda parte é o valor de λ, no intervalo de 0 a 4. Sua representação binária é a proporção do valor real de acordo com quantidade de bits definidas pelo programador para λ. Os autores apresentam os conceitos de cromossomo convergente, periódico e caótico. Quando λ é menor que 3, o cromossomo é classificado como convergente, ou seja, o cromossomo tende a convergir para valores constantes nos processos de iteração do GATC. Caso o valor seja entre 3 e 3,56, o cromossomo é classificado como periódico, neste caso, o cromossomo tende a se tornar neutro para o algoritmo, ou seja, durante as iterações, seu valor pode convergir ou variar. E, por fim, se λ está entre 3,56 e 4, o cromossomo é caótico, assim, o cromossomo tem um comportamento evolutivo completamente variável durante as iterações.

A terceira parte do cromossomo é a mascara uniforme, representada de forma binária identificando se o gene herdado pertence ao primeiro ou segundo pai. A mascara é formada com a mesma quantidade de genes contidos na primeira parte, a qual representa a solução do problema.

Na etapa do cruzamento do GATC é utilizada a 26, com o objetivo de herdar aos filhos a terceira parte do cromossomo composta pela máscara uniforme. A máscara é representada em *Gray Code*. A utilização do conceito *Gray Code* e da função caótica serão explicados nas seções seguintes.

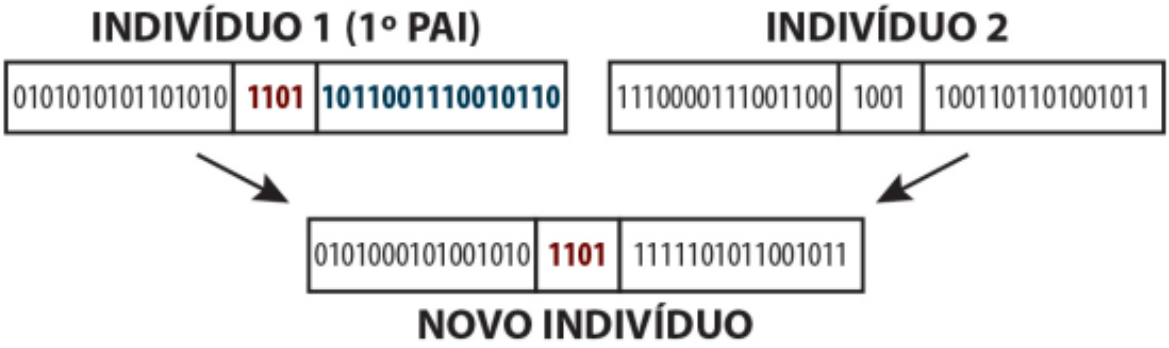
2.8.1.3 Inicialização, Avaliação e Seleção

Essas etapas são realizadas de maneira similar a de um AG padrão, respeitando a representação cromossomial do GATC, assim, os critérios de aleatoriedade são definidos pelo desenvolvedor. Uma diferença em relação a etapa de inicialização, é que os valores de λ podem ser apenas convergentes, periódicos ou caóticos, podendo optar também por mais de um tipo.

2.8.1.4 Cruzamento e Mutação

A máscara uniforme determina de qual pai o filho herdará o gene. Dessa forma, se no primeiro gene da máscara tiver o valor 1, o gene copiado para o primeiro filho será do primeiro pai e, se o valor for 0, ele herdará o primeiro gene do segundo pai, o segundo

Figura 9 – Exemplo de cruzamento do GATC



filho herdará o gene correspondente ao outro pai e assim sucessivamente.

A máscara utilizada é apenas a do primeiro pai, da mesma forma, a λ dos filhos gerados serão, também, a do primeiro pai.

A máscara uniforme herdada pelos filhos será obtida a partir da modificação da máscara do primeiro pai, utilizando a função caótica apresentada anteriormente. Essa etapa definida pelos autores procede da seguinte forma:

- O valor binário em *Gray Code* é convertido para valor binário natural. Cada subconjunto de bits que representam uma solução são manipulados separadamente.
- Os valores obtidos em forma binária natural são transformados em decimal natural.
- É realizado a normalização do valor decimal obtido.
- O valor de λ do primeiro pai passa pelo mesmo processo de transformação para número real, porém, utilizando o intervalo (0,4)
- Os valores de cada parte de solução da máscara e o valor de λ são inseridos na função caótica
- Estes valores modificados da máscara, correspondentes as partes de solução passam então pelo procedimento inverso para a representação *Gray Code*.
- A máscara uniforme de cruzamento resultante, modificada pela função caótica será então herdada pelos dois filhos.

O Processo descrito acima é detalhado pela Figura 9 e pela Tabela 5, onde demonstra-se como é produzido um novo indivíduo a partir dos pais selecionados e do uso da máscara uniforme, a aplicação da função caótica e o processo de geração da nova máscara uniforme que será herdada pelos filhos.

Após todo o processo de cruzamento realizado pelo algoritmo, os autores optaram pela mutação aleatória, assim, é escolhido um gene aleatório e seu valor é alterado de 1

Tabela 5 – Processo de modificação da máscara utilizando a função caótica.

Máscara	**x1**	**x2**
Gray Code	10110011	10010110
Binário	11011101	11100100
Decimal	221	228
No intervalo (0,1)	0,8667	0,8941
Modificada pela função caótica	0,3374	0,2765
Modificada em decimal	86	70
Modificada em binário	10101101	10001101
Modificada em Gray Code	11111010	11001011

para 0 ou 0 para 1, a diferença do GATC para o AG padrão é que, a alteração realizada no cromossomo resultante também é realizada no gene equivalente da máscara uniforme.

3 TRABALHOS RELACIONADOS

O trabalho realizado por Koblar (KOBLAR, 2012) propõe a otimização da configuração dos parâmetros do algoritmo J48 utilizando o algoritmo evolutivo *Differential Evolution for Multiobjective Optimization* (DEMO). A ideia consiste em uma otimização multi-objetiva, onde é necessário melhorar o resultado da acurácia e diminuir o tamanho da árvore gerada. Ele optou por configurar cinco parâmetros do algoritmo J48, sendo eles:

- M – Número mínimo de instâncias em uma folha;
- U – Uso de árvores não podadas;
- C – Fator de confiança usado na pós-poda;
- S – Operação de levantamento de sub árvores pós poda;
- B – Uso de divisões binárias.

A otimização dos parâmetros do algoritmo J48 com o DEMO foi realizada em três domínios de aprendizagem: *commutators, emulsion* e *steel plates*. No primeiro domínio os testes realizados foram satisfatórios, onde obteve-se o mesmo resultado de acurácia, em relação ao teste com o algoritmo em seus valores *default*, e um tamanho de árvore menor. No segundo domínio, tanto o valor de acurácia quanto o tamanho da árvore gerada foram melhores que o algoritmo executando com parâmetros *default*. No terceiro domínio, obteve-se mesmo resultado de acurácia com um árvore de tamanho significativamente menor, em comparação com a árvore de decisão construída com valores de parâmetros padrão.

O trabalho realizado por Mantovani et al. (2016), tem por objetivo, realizar uma configuração automática dos hiperparâmetros do algoritmo árvore de decisão J48. Para isso, utilizou-se 4 técnicas de otimização, *Random Search* (RS), Algoritmos genéticos, *Particle Swarm Optimization* (PSO) e *Estimation of Distribution Algorithm* (EDA). Foi utilizado a acurácia como metodologia de avaliação de resultados. Os testes foram realizado em 102 bases de dados distintas, essas disponíveis em OpenML [1]. Os resultados dos algoritmos testados com os valores *default* são eficientes, porém, a otimização empregada na etapa de configuração dos hiperparâmetros resultam em uma melhoria na maioria conjuntos testados, onde, dentre as 102 bases de dados, 70 delas obtiveram um desempenho classificatório significativos. Portanto, concluiu-se que para tarefas mais simples os valores *default* podem ser utilizados sem tantos problemas, mas mesmo assim, a otimização dos hiperparâmetros são mais indicados.

[1] <https://www.openml.org/>

No trabalho de Lorenzo et al. (2017), fui utilizado *Particle Swarm Optimization* (PSO) como algoritmo de busca para otimização dos hiperparâmetros do algoritmo *Deep Neural Networks* (DNNs). É ressaltado que, o desempenho de uma DNNs depende diretamente dos hiperparâmetros, que, normalmente, são selecionados por um especialista. Porém, essa otimização continua sendo um obstáculo para criação de uma DNNs. Nesse contexto, a solução proposta pelos autores consegue melhorar o desempenho das arquiteturas existentes, observando também, que o PSO é uma técnica eficaz para otimização do processo de seleção de hiperparâmetros. Contudo, utilizou-se RS e *Grid Search* (GS) como comparação dos resultados. Assim, os resultados obtidos pelo PSO se aproxima da solução ótima, percorrendo o espaço da solução de maneira íntegra fornecendo resultados consistentes e de alta qualidade em configurações experimentais diferentes, ultrapassando claramente a experiência humana ao otimizar uma arquitetura DNN.

Em Qolomany et al. (2017), os autores abordaram a otimização de parâmetros de modelos de *Deep Learning* (DL) usando PSO. O foco da pesquisa é a otimização de dois parâmetros chaves, o número de camadas ocultas e o número de neurônios em cada camada, onde os testes foram realizados usando um conjunto de dados coletados de uma rede Wi-Fi, com foco de treinar modelos DL para prever o número de ocupantes e suas localizações. Os autores optaram por realizar suas comparações com o algoritmo de busca *Grid Search* e a métrica de avaliação escolhida foi a acurácia. Os resultados mostram que o algoritmo PSO é útil no processo de treinamento de modelos de DL, assim, os tempos de treinamento diminuíram consideravelmente em comparação com o método GS.

No trabalho publicado em Bergstra e Bengio (2012), os autores realizaram experimentos com uma Rede Neural do tipo *Deep Belief Network* (DBN). Uma DBN é um modelo gráfico multicamada com componentes direcionados e não direcionados, e sua parametrização ocorre de forma similar a uma rede neural multicamada. Pois, um classificador DBN possui muitos parâmetros. A ideia geral da pesquisa é comparar o uso de *Grid Search*, *Manual Search* (MS) e *Random Search* para o processo de otimização de hiperparâmetros de uma DBN. Foram utilizadas 8 base de dados para realização da classificação, e comparando, GS e MS, a RS sobre o mesmo espaço de configuração de 32 dimensões encontrou desempenho estatisticamente igual em 4 bases de dados e desempenho superior em 1, obtendo desempenho inferior em 3 base de dados. Os autores concluíram que RS geralmente não é tão boa quanto a combinação sequencial de GS e MS, porém, no caso do problema de busca 32-dimensional da otimização da DBN, RS obteve bons resultados.

4 METODOLOGIA

Como base do estudo, utilizou-se técnicas de inteligência artificial, tais como, aprendizado de maquina, computação evolucionária englobando algoritmos genéticos e algumas variações. Para desenvolvimento do trabalho foram selecionadas algumas ferramentas, descritas a seguir.

4.1 Ferramentas

4.1.1 Python

Python é uma linguagem de programação poderosa, ele possui estruturas de dados eficientes de alto nível e uma abordagem simples, mas eficaz, para programação orientada a objetos. A sintaxe simples e a digitação dinâmica do Python, junto com sua natureza interpretada, tornam-no uma linguagem ideal para *scripts* e desenvolvimento rápido de aplicativos em muitas áreas na maioria das plataformas. O interpretador Python e a extensa biblioteca padrão estão disponíveis gratuitamente em formato fonte ou binário para todas as principais plataformas no site da Web em Python [1], e podem ser distribuídos gratuitamente (FOUNDATION, 2018).

Todos os códigos desenvolvidos nesse trabalho foram realizados com a linguagem Python de programação.

4.1.2 *scikit-learn*

scikit-learn é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python, seu desenvolvimento é projetado para interagir com as bibliotecas Python numérica *NumPy* e científica *SciPy* (SKLEARN, 2018).

Como dito anteriormente, é uma biblioteca de ML poderosa, ela inclui vários algoritmos de classificação, regressão e agrupamento incluindo máquinas de vetores de suporte, *Random Forest*, *Gradient Boosting*, *k-means*, *Decision Tree*. Esta última classe de algoritmos foi adotada como algoritmo de classificação no desenvolvimento deste trabalho.

Como foi relatado nos objetivos, tem-se como foco, a otimização dos hiperparâmetros do algoritmo árvore de decisão *Classification and Regression Trees*, descrito na Seção 2.3.3. Este algoritmo também está implementado na biblioteca *scikit-learn*. Optou-se por manipular os parâmetros do algoritmo CART presentes na Tabela 6.

[1] <https://www.python.org/>

Tabela 6 – Parâmetros do Algoritmo CART.

Variáveis	**Descrição**
criterion	A função para medir a qualidade de uma divisão
splitter	A estratégia usada para escolher a divisão a cada nó
max_depth	A profundidade máxima da árvore
min_samples_split	O número mínimo de amostras para dividir um nó interno
min_samples_leaf	O número mínimo de amostras para estar em um nó folha
min_weight_fraction_leaf	A fração ponderada miníma da soma total dos pesos necessários em um nó folha
presort	Se deve pre-ordenar os dados para acelerar a descoberta das melhores divisões no ajuste

4.2 Bases de dados

Para a etapa de aplicação do algoritmo CART, escolheu-se um total de 7 (sete) bases de dados, obtidas gratuitamente nos sites UCI *Machine Learning Repository* [2] e OpenML [3], sendo elas, *Steel Plates*, *Breast Cancer*, *Wine*, *Iris*, *Isolet* e *Madelon*.

4.2.1 *Steel Plates*

Esta base de dados contém dados de falhas na produção de placas de aço, sendo 1941 exemplos descritos por 27 atributos e classificados em 7 classes. Esses atributos descrevem as propriedades físicas das placas de aço, como tamanho do defeito, posição da falha, refletância da luz da superfície, tipo de material, etc. Todos os atributos são numéricos.

Tabela 7 – Detalhes da base de dados *Stell Plates*

Características do Atributo:	Inteiro, Real
Número de Registros:	1941
Número de Atributos:	27
Valores faltantes:	Não há

4.2.2 *Breast Cancer*

Nesta base de dados relata-se sobre a ocorrência ou não do câncer de mama, estão incluidos 201 instâncias de uma classe e 85 instâncias de outra classe. As instâncias são descritas por 9 atributos, alguns dos quais são lineares e alguns são nominais.

[2] <https://archive.ics.uci.edu/ml/datasets.html>
[3] <https://www.openml.org/>

Tabela 8 – Discriminação das classes e seus registros da base de dados *Stell Plates*

Classe	**Número de Exemplos**
Pastry	158
Z_scratch	190
K_scratch	391
Stains	72
Dirtiness	55
Bumps	402
Other faults	673

Tabela 9 – Detalhes da base de dados *Breast Cancer*

Características do Atributo:	Categórico
Número de Registros:	286
Número de Atributos:	9
Valores faltantes:	Não há

Tabela 10 – Discriminação das classes e seus registros da base de dados *Breast Cancer*

Classe	**Número de Exemplos**
Ocorrência	85
Não Ocorrência	201

4.2.3 *Wine*

Está base de dados mostram os resultados de uma análise química de vinhos mas derivados de três diferentes cultivares. A análise determinou as quantidades de 13 constituintes encontrados em cada um dos três tipos de vinhos. Foram registadas 178 instancias.

Tabela 11 – Detalhes da base de dados *Wine*

Características do Atributo:	Inteiro, Real
Número de Registros:	178
Número de Atributos:	13
Valores faltantes:	Não há

Tabela 12 – Discriminação das classes e seus registros da base de dados *Wine*

Classe	**Número de Exemplos**
Tipo 1	59
Tipo 2	71
Tipo 3	48

4.2.4 *Iris*

Este é talvez o banco de dados mais conhecido encontrado na literatura sobre reconhecimento de padrões. O conjunto de dados contém 3 classes de 50 instâncias cada, onde cada classe se refere a um tipo de planta da íris.

Tabela 13 – Detalhes da base de dados *Iris*

Características do Atributo:	Real
Número de Registros:	150
Número de Atributos:	4
Valores faltantes:	Não há

Tabela 14 – Discriminação das classes e seus registros da base de dados *Iris*

Classe	**Número de Exemplos**
Iris Setosa	50
Iris Virginica	50
Iris Versicolour	50

4.2.5 *Isolet*

Este conjunto de dados foi gerado da seguinte forma. 150 voluntários falaram o nome de cada letra do alfabeto duas vezes

Tabela 15 – Detalhes da base de dados *Isolet*

Características do Atributo:	Real
Número de Registros:	1560
Número de Atributos:	617
Valores faltantes:	Não há

4.2.6 *Madelon*

Este conjunto de dados contém pontos de dados agrupados em 32 agrupamentos colocados nos vértices de um hipercubo de cinco dimensões e rotulados aleatoriamente com +1 ou -1. As cinco dimensões constituem 5 características informativas, assim, 15 combinações lineares desses recursos foram adicionadas para formar um conjunto de 20 recursos informativos.

Tabela 16 – Discriminação das classes e seus registros da base de dados *Isolet*

Classe	Número de Exemplos
1	60
2	60
3	60
4	60
5	60
6	60
7	60
8	60
9	60
10	60
11	60
12	60
13	60
14	60
15	60
16	60
17	60
18	60
19	60
20	60
21	60
22	60
23	60
24	60
25	60
26	60

Tabela 17 – Detalhes da base de dados *Madelon*

Características do Atributo:	Real
Número de Registros:	600
Número de Atributos:	500
Valores faltantes:	Não há

Tabela 18 – Discriminação das classes e seus registros da base de dados *Madelon*

Classe	Número de Exemplos
1	300
-1	300

4.3 Desenvolvimento e avaliação

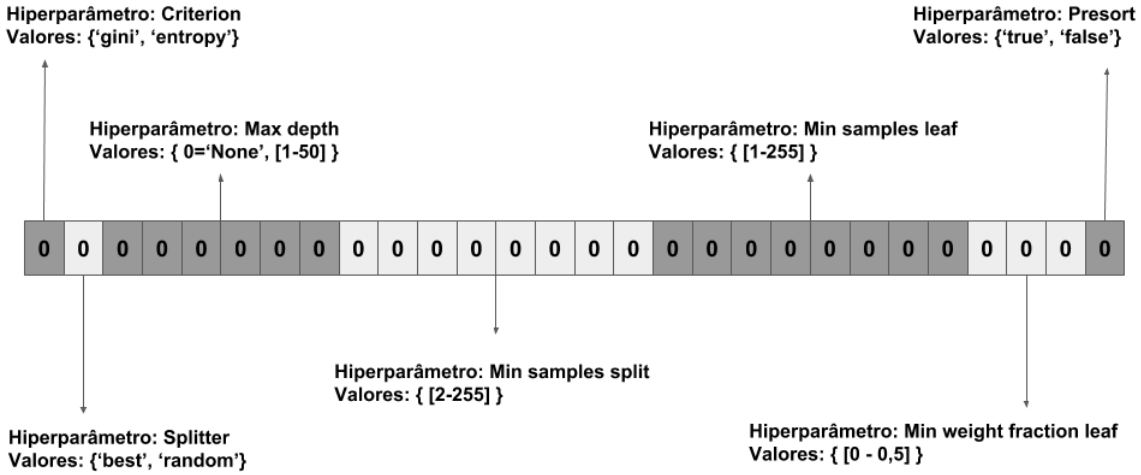
O desenvolvimento foi dado pela utilização dos algoritmos AG, FGA e GATC como estrátegia de otimização dos hiperparâmetros do algoritmo CART. A representação

cromossomial, o funcionamento dos algoritmos e a função objetivos serão descritos a seguir.

4.3.1 AG

O AG seguiu os padrões descritor na Seção 2.6, foi utilizado a representação binária, cujo o cromossomo resultante é descrito pela Figura 10.

Figura 10 – Representação binária do cromossomo resultante

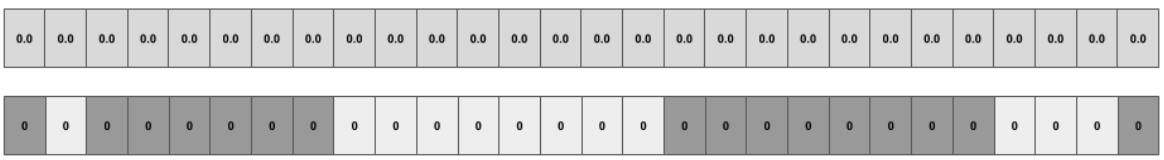


4.3.2 FGA

Como descrito sobre o FGA na Seção 2.7, o algoritmo possui uma abordagem diferente dos AG's comuns, onde sua representação cromossomial e a atribuição da função *born an individual* são os pontos de mudança e requerem uma manipulação específica para as mesmas, assim, o FGA precisou ser adaptado.

Foi utilizado o cromossomo na representação binária, tendo o indivíduo discriminado entre os hiperparâmetros selecionados e descritos na seção 4.1.2. O cromossomo ficou assim representado de acordo com a Figura 11.

Figura 11 – Representação do indivíduo cromossomo do algoritmo FGA

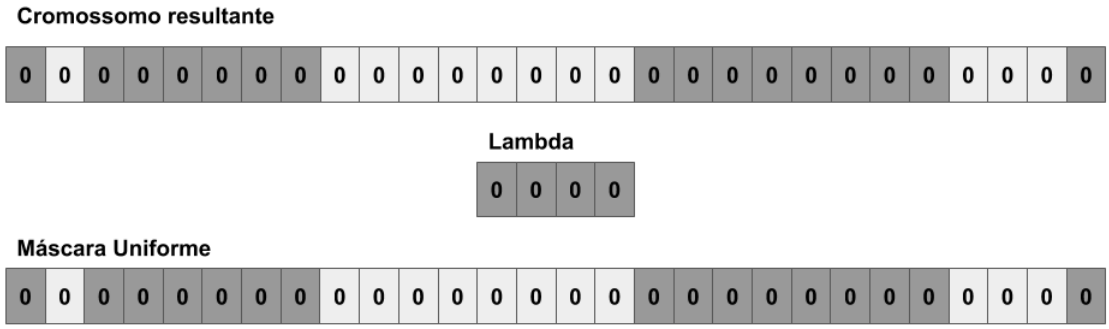


4.3.3 GATC

Como descrito sobre o GATC na Seção 2.8, o algoritmo possui uma abordagem diferente dos AG's comuns, onde sua representação cromossomial e a atribuição da função

caótica são os pontos de mudança e requerem uma manipulação específica para as mesmas, assim, o GATC precisou ser adaptado. O cromossomo ficou representado de acordo a Figura 12.

Figura 12 – Representação do Cromossomo do algoritmo GATC



4.3.4 Função Objetivo

Quando utilizado um classificador, é possível medir e quantificar a taxa de acerto, essa medida é atribuída por uma métrica de avaliação, descritas na Seção 2.4. Além disso, um algoritmo de árvore decisão possibilita reduzir o tamanho da árvore final gerada. Nesse contexto, tendo como foco desse trabalho a configuração automática dos hiperparâmetros do algoritmo CART, depara-se com um problema de otimização multi-objetivo, onde é possível reduzir o tamanho da árvore final gerada e aumentar o resultado obtido na avaliação.

Com isso, para cálculo do *fitness* dos algoritmos, utilizando o tamanho da árvore e a acurácia obtida, é preciso diminuir o tamanho e aumentar a acurácia, assim, escolheu-se maximizar o valor obtido pelo tamanho, e a fórmula usada foi descrita por,

$$fitness = \left(\frac{1}{tamanho}\right) + acurcia \tag{27}$$

Dessa forma, foi aplicado os algoritmos AG, FGA e GATC ao algoritmo CART.

5 ANÁLISES E RESULTADOS

Utilizou-se os algoritmos AG, FGA e GATC para otimizar da etapa de configuração de parâmetros do algoritmo de ML CART. Para realização dos testes e comparação dos resultados foi utilizado as bases de dados *Stell Plates*, *Wine*, *Breast Cancer*, *Digits*, *Iris*, *Isolet* e *Madelon*.

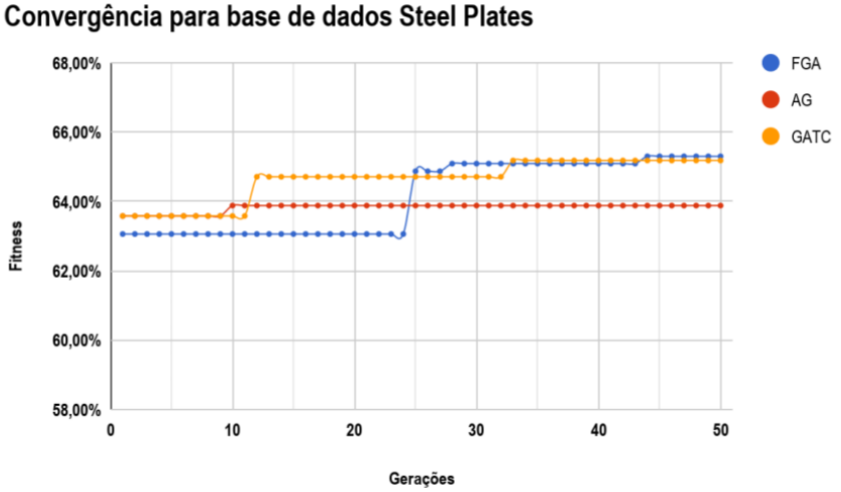
Como visto sobre o algoritmo FGA na Seção 2.7, foi adotado os valores da Tabela 19 para as constantes presentes no algoritmo. Para os algoritmos GATC e AG, vide Seções 2.6 e 2.8, foi adotado o valor da população inicial igual a 100 (cem).

Tabela 19 – Descrição dos parâmetros utilizados no desenvolvimento do algoritmo FGA

Variáveis	**Valores**
Taxa de aprendizado global : N_g	0,1
Taxa de aprendizado individual : N_i	0,1
Taxa de diversidade : N_{dr}	0
População Inicial	100

Durante a realização dos testes, verificou-se a convergência dos algoritmos entre 40 e 50 gerações, com isso, adotou-se como critério de parada o valor de 50 gerações. E como métrica de avaliação foi escolhido a acurácia. As Figuras 13, 14, 15, 17, 16, 18 e 19 demonstram os melhores indivíduos de cada geração, explicitando a convergência dos algoritmos, baseando-se na função de avaliação vista na Seção 4.3.4.

Figura 13 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Steel Plates



Utilizou-se 30 populações iniciais diferentes para execução dos testes dos 3 algoritmos, assim, as Figuras 20, 21 e 22, mostram o melhor indivíduo de cada execução.

Figura 14 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Iris

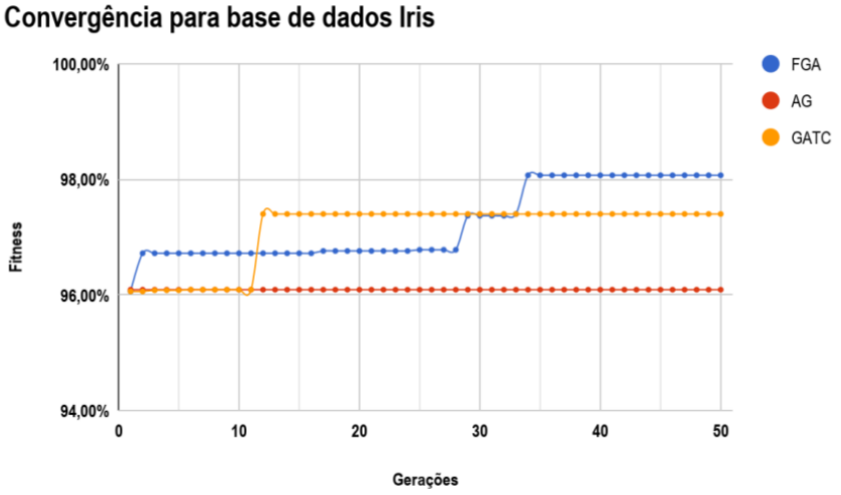
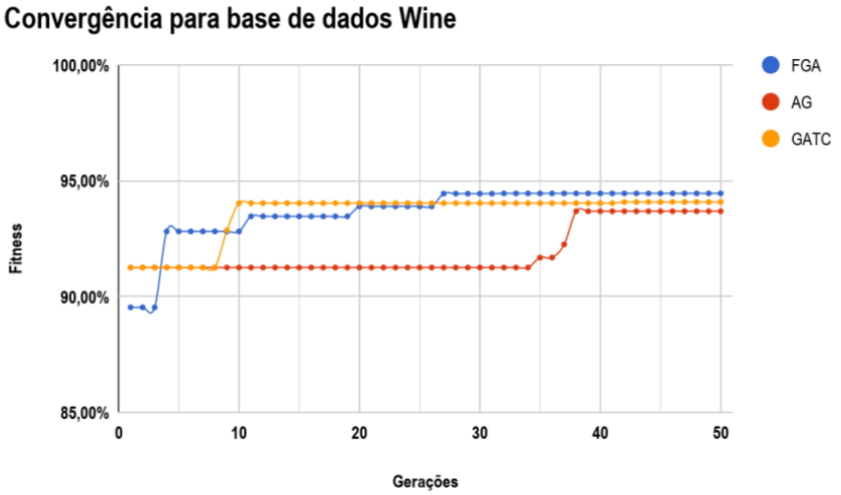


Figura 15 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Wine



As bases de dados possuem dificuldades diferentes, onde possibilita que, em cada base, tenha um algoritmo que obtenha melhor resultado. A Figura 23 compara os resultados obtidos pelos algoritmos aplicados ao algoritmo CART, e faz comparação com o algoritmo CART aplicando seus valores *default*. A Figura 24 compara a profundidade das árvores finais geradas entre os resultados dos algoritmos e o algoritmo CART com seus valores *default*.

A Tabela 20 descreve a taxa de acerto do uso do FGA, AG e GATC utilizando a métrica de avaliação acurácia. É possível notar que os resultados obtidos foram melhores, comparado com o algoritmo CART com seus valores padrões. O uso das variações do AG padrão, traz, em sua maioria, um resultado melhor do que o obtido pelo AG. Exempli-

Figura 16 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Breast Cancer

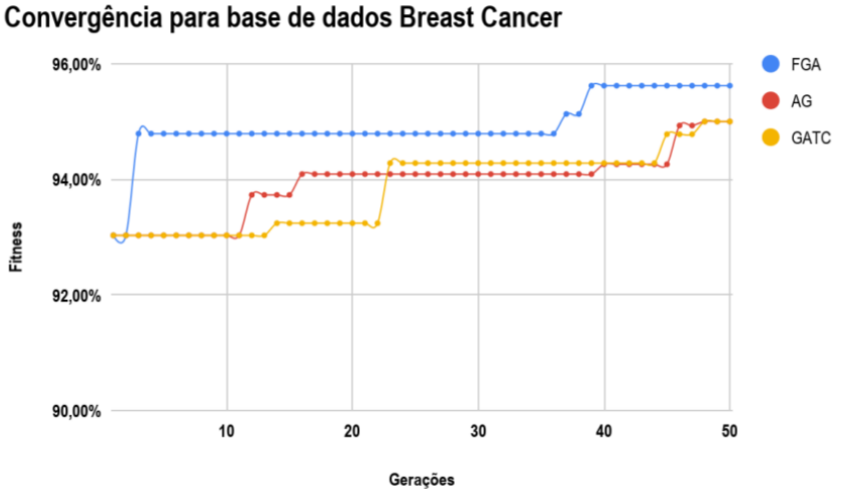
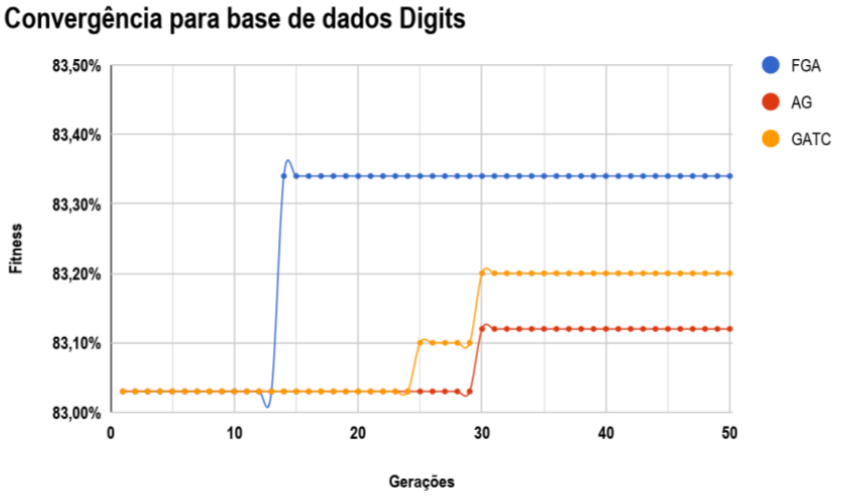


Figura 17 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Digits



ficando, nas base de dados *Steel Plates*, *Iris*, *Wine*, *Breast Cancer*, *Digits* e *Madelon*, o FGA possui melhores resultados do que os outros algoritmos, já na base de dados *Isolet*, o AG padrão obteve melhor resultado comparando com as suas variações FGA e GATC. O algoritmo GATC, obteve, na maioria dos testes, melhor resultado que o AG padrão, porém, se mostrou inferior ao FGA dentro do conjunto de dados testados no aglortimo CART.

Por exemplo, a base de dados Iris já possuía uma acurácia relativamente alta, porém, os algoritmos evolutivos conseguiram melhorar essa avaliação.

A Figura 24 descreve o tamanho da árvore final gerada. O uso dos algoritmos evolutivos foram suficiente em todas bases de dados, exceto com a *Digits*, onde o AG obteve

Figura 18 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Isolet

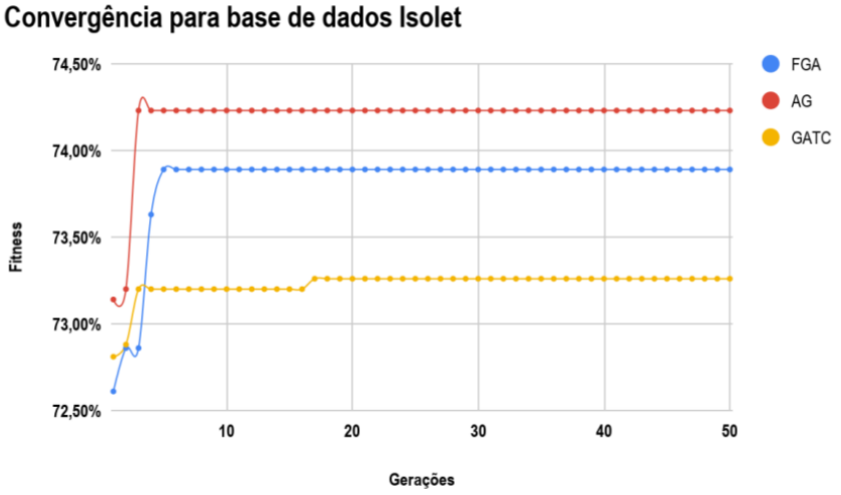


Figura 19 – Gráfico com o melhor indivíduo a cada geração, demonstrando a convergência dos algoritmos em relação a base de dados Madelon

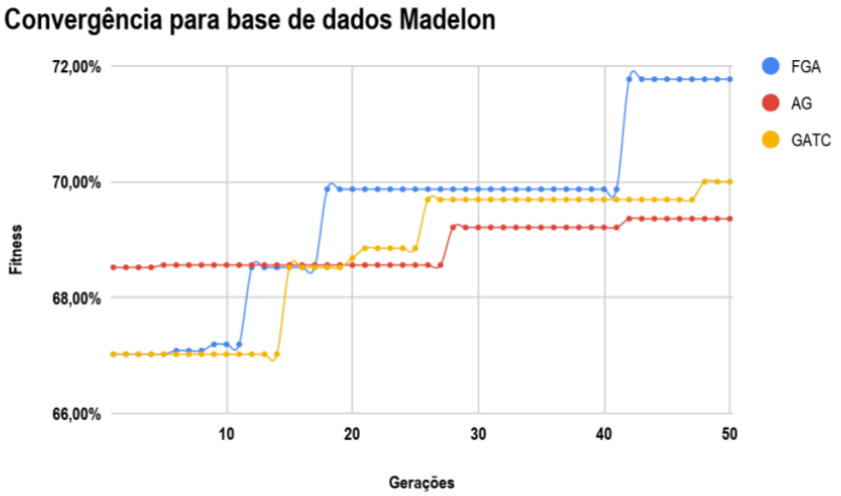


Tabela 20 – Taxa de acerto dos algoritmos FGA, AG, GATC e o algortimo CART com seus valores padrões

	CART Default	FGA	AG	GATC
Stell Plates	56,49%	**65,30%**	63,96%	64,97%
Breast Cancer	91,76%	**95,62%**	95,00%	95,00%
Iris	96,00%	**98,00%**	97,30%	97,52%
Wine	89,47%	**94,24%**	93,68%	94,08%
Digits	83,03%	**83,34%**	83,12%	83,18%
Isolet	72,61%	73,99%	**74,02%**	73,93%
Madelon	65,50%	**71,77%**	69,40%	70,00%

Figura 20 – Melhores indivíduos em 30 populações diferente para o algoritmo AG

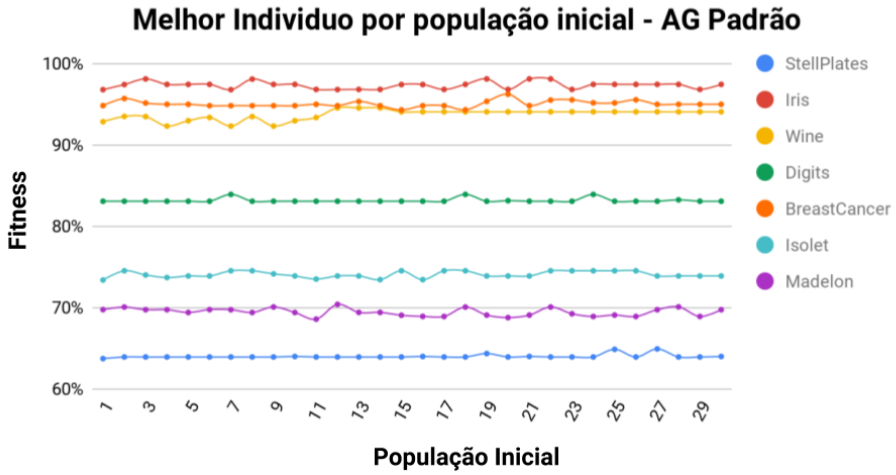
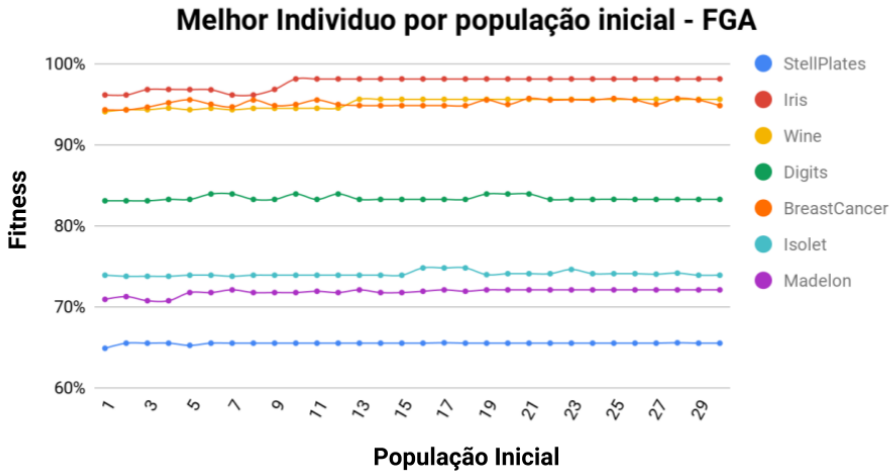


Figura 21 – Melhores indivíduos em 30 populações diferente para o algoritmo FGA



aumento do tamanho, e com a base *Iris*, onde o AG e o GATC obtiveram um tamanho final maior que algoritmo executado com os valores padrões. É possível ressaltar que, a aplicação nas bases *Stell plates*, *Isolet* e *Madelon* foi bem significativa. Respectivamente para FGA, AG e GATC, na base *Steel plates*, foi reduzido o tamanho de 607 para 27|27|35, na base de dados *Isolet*, foi reduzido de 331 para 91|61|93 e na base de dados *Madelon*, foi reduzido de 109 para 29|37|37.

Como se trata de um problema multiobjetivo, os algoritmos evolutivos obtiveram resultados bastantes satisfatórios dentro do conjunto de dados estudados e aplicados a uma algoritmo árvore de decisão. Conseguindo obter melhor taxa de acerto e reduzindo o tamanho da árvore final gerada.

Figura 22 – Melhores indivíduos em 30 populações diferente para o algoritmo GATC

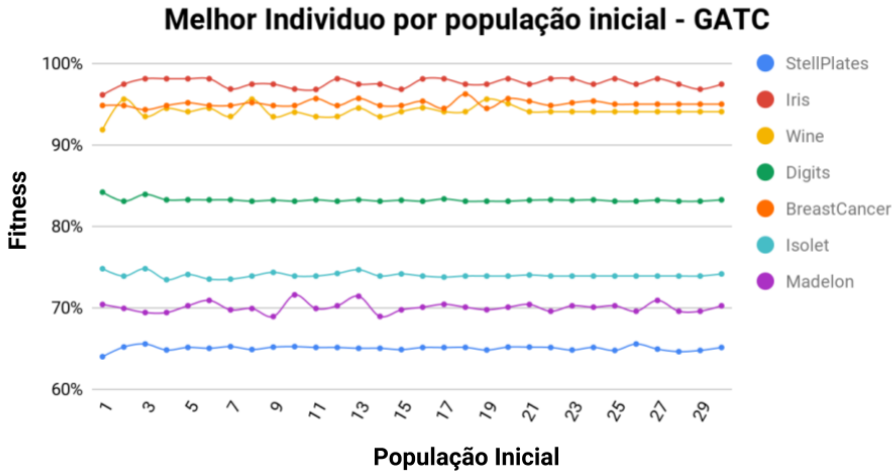


Figura 23 – Comparação do Fitness obtido entre os algoritmos AG, FGA e GATC, e a utilização do algoritmo CART com os valores default

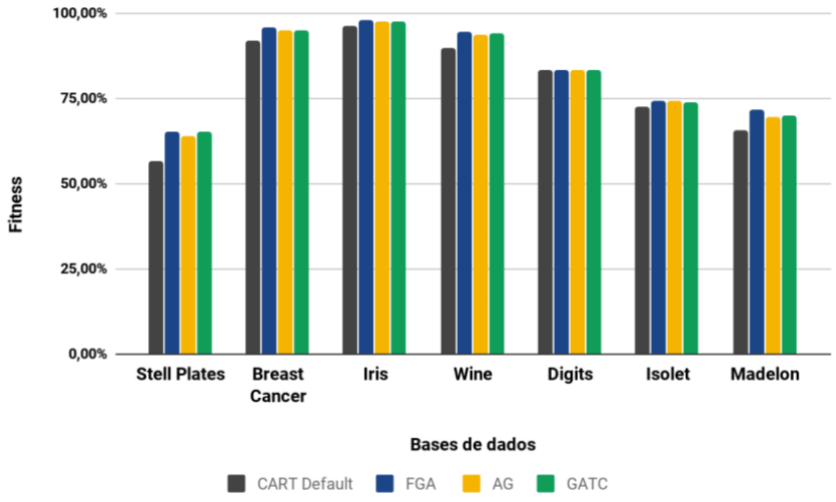
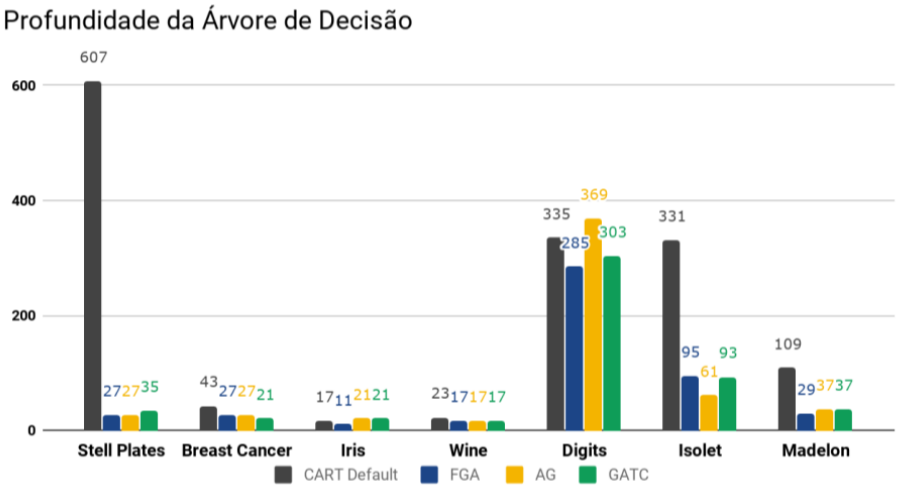


Figura 24 – Comparação da Profundidade da árvore final gerada, obtida entre os algoritmos AG, FGA e GATC, e a utilização do algoritmo CART com os valores default



6 CONSIDERAÇÕES FINAIS

As pesquisas sobre configuração automática de hiperparâmetros são constantes, e a aplicação de meta-heurísticas vem ganhando espaço como estrategias de busca. Após uma aplicação sistemática dos algoritmos estudados nesse trabalho, obteve-se resultados bastantes satisfatórios.

Os algoritmos escolhidos para manipulação foram, o algoritmo genérico padrão (AG), e duas variações do AG, o *Fluid Genetic Algorithm* (FGA) e o *Genetic Algorithm using Theory of Chaos* (GATC). Foi necessário manipular e adaptar o problema para cada algoritmo, para assim, realizar a etapa de testes. Para o modelo de *machine learning*, foi escolhido o algoritmo árvore de decisão *Classification and Regression Trees* (CART). A maior dificuldade da adaptação dos algoritmos FGA e GATC para usar junto do CART foi a manipulação dos, indivíduos e cromossomos, onde foi necessário retratar os hiperparâmetros de forma binária e aplicar a suas funções características, *born an individual* e função caótica.

6.1 Trabalhos Futuros

Para continuidade desse trabalho, pretende-se implementar as variações dos algoritmos genéticos presentes no trabalho do Borges (2017), e aplicar a algum(s) algoritmo(s) de *machine learning* além das árvores de decisão. Também é pertinente, propor uma nova variação de algoritmo genético híbrido, aplicando junções das varições apresentadas.

REFERÊNCIAS

ARONSON, J. E.; LIANG, T.-P.; TURBAN, E. **Decision support systems and intelligent systems**. [S.l.]: Pearson Prentice-Hall, 2005.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. **Journal of Machine Learning Research**, v. 13, n. Feb, p. 281–305, 2012.

BORGES, J. P. V. **REVISAO SISTEMÁTICA SOBRE ALGORITMOS GENÉTICOS**. [S.l.], 2017.

BREIMAN JEROME FRIEDMAN, C. J. S. L.; OLSHEN, R. **Classification and Regression Trees**. [S.l.]: Wadsworth Press, 1984.

FOUNDATION, P. P. S. **Python 3.6.5 documentation**. 2018. <https://docs.python.org/3/>. Acessado: 2018-04-30.

JAFARI-MARANDI, R.; SMITH, B. K. Fluid genetic algorithm (fga). **Journal of Computational Design and Engineering**, v. 4, n. 2, p. 158 – 167, 2017. ISSN 2288-4300. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2288430016300458>.

KOBLAR, V. **Optimizing parameters for machine learning algorithm**. [S.l.], 2012.

LINDEN, R. **Algoritmos genéticos: uma importante ferramenta da inteligência computacional**. [S.l.]: Brasport, 2006.

LORENZO, P. R. et al. Particle swarm optimization for hyper-parameter selection in deep neural networks. In: ACM. **Proceedings of the Genetic and Evolutionary Computation Conference**. [S.l.], 2017. p. 481–488.

LUO, G. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. **Network Modeling Analysis in Health Informatics and Bioinformatics**, v. 5, p. 1–16, 2016.

MANTOVANI, R. G. et al. Hyper-parameter tuning of a decision tree induction algorithm. In: IEEE. **Intelligent Systems (BRACIS), 2016 5th Brazilian Conference on**. [S.l.], 2016. p. 37–42.

MITCHELL, M. **An introduction to genetic algorithms**. [S.l.]: MIT press, 1998.

MITCHELL, T. M. et al. **Machine learning. WCB**. [S.l.]: McGraw-Hill Boston, MA:, 1997.

PROBST, P.; BISCHL, B.; BOULESTEIX, A.-L. Tunability: Importance of hyperparameters of machine learning algorithms. **arXiv preprint arXiv:1802.09596**, 2018.

QOLOMANY, B. et al. Parameters optimization of deep learning models using particle swarm optimization. In: IEEE. **Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International**. [S.l.], 2017. p. 1285–1290.

ROKACH, L.; MAIMON, O. Clustering methods. In: **Data mining and knowledge discovery handbook**. [S.l.]: Springer, 2005. p. 321–352.

ROSSI, A. L. D. **Ajuste de parâmetros de técnicas de classificação por algoritmos bioinspirados**. Tese (Doutorado) — Universidade de São Paulo, 2009.

SKLEARN. **Scikit Learning documentation**. 2018. <http://scikit-learn.org/stable/>. Acessado: 2018-04-30.

SNASELOVA, P.; ZBORIL, F. Genetic algorithm using theory of chaos. **Procedia Computer Science**, Elsevier, v. 51, p. 316–325, 2015.

TICONA, W. G. C. **Algoritmos evolutivos multi-objetivo para a reconstrução de árvores filogenéticas**. Tese (Doutorado) — Universidade de São Paulo, 2003.

YUAN, F.-C. Parameters optimization using genetic algorithms in support vector regression for sales volume forecasting. **Applied Mathematics**, Scientific Research Publishing, v. 3, n. 10, p. 1480, 2012.