UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS SOBRAL

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

ANDRÉ LUÍS MARQUES RODRIGUES

UM ESTUDO COMPARATIVO ENTRE ALGORITMOS PARA OTIMIZAÇÃO DE HIPERPARÂMETROS

SOBRAL

2023

ANDRÉ LUÍS MARQUES RODRIGUES

UM ESTUDO COMPARATIVO ENTRE ALGORITMOS PARA OTIMIZAÇÃO DE HIPERPARÂMETROS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da Computação Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia da Computação.

Orientador: Prof. Me. David Nascimento Coelho.
Coorientador: Flávio Vasconcelos dos Santos.

SOBRAL

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

R611e Rodrigues, André.
Um estudo comparativo entre algoritmos para otimização de hiperparâmetros / André Rodrigues. – 2023.
46 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2023.
Orientação: Prof. Me. David Nascimento Coelho.
Coorientação: Prof. Flávio Vasconcelos dos Santos.

1. Hiperparâmetros. 2. Otimização. 3. Busca em Grade. 4. Otimização Bayesiana. 5. otimização por enxame de partículas. I. Título.

CDD 621.39

ANDRÉ LUÍS MARQUES RODRIGUES

UM ESTUDO COMPARATIVO ENTRE ALGORITMOS PARA OTIMIZAÇÃO DE HIPERPARÂMETROS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da Computação Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia da Computação.

Aprovada em: 13/07/2023.

BANCA EXAMINADORA

Prof. Me. David Nascimento Coelho (Orientador)
Universidade Federal do Ceará (UFC)

Flávio Vasconcelos dos Santos (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Iális Cavalcante de Paula Júnior
Universidade Federal do Ceará (UFC)

Prof. Dr. Carlos Alexandre Rolim Fernandes
Universidade Federal do Ceará (UFC)

RESUMO

A inteligência artificial e os modelos de aprendizado de máquina têm se tornado cada vez mais relevantes com o passar dos anos, por conta da grande quantidade de aplicações que eles possuem em grande parte dos campos da sociedade. Torna-se, assim, de grande relevância o estudo de maneiras de melhorar a qualidade desses sistemas, e uma das formas de melhorar seu desempenho é através da otimização de hiperparâmetros. Assim, este trabalho tem como objetivo analisar de forma comparativa a aplicação de diferentes algoritmos na otimização de hiperparâmetros de modelos de aprendizado de máquina. São abordadas 5 técnicas de otimização, sendo estas a busca em grade, busca randômica, busca bayesiana, algoritmos genéticos e otimização por enxame de partículas, onde é realizada uma pesquisa a respeito de soluções para integração dessas técnicas visando facilitar a sua aplicação para a otimização tanto dos modelos de aprendizado abordados no trabalho quanto de outros modelos. A análise comparativa é feita explorando tarefas de aprendizado supervisionado, utilizando quatro bancos de dados de classificação e um banco de dados de regressão, com o objetivo de comparar a eficiência de cada técnica de otimização em problemas variados e comparar também influências de características próprias de cada modelo, tais como número de hiperparâmetros em cada um.

Palavras-chave: hiperparâmetros; otimização; busca em grade; busca randômica; otimização bayesiana; algoritmos genéticos; otimização por enxame de partículas.

ABSTRACT

Artificial intelligence and machine learning models have become increasingly relevant over the years, because of the large amount of applications that they have in most areas of society. Therefore, the study of ways to improve the quality of these systems becomes very relevant, and one of the ways to improve their performance is through the hyperparameters optimization. Thus, this work aims to analyze comparatively the application of different algorithms in the hyperparameter optimization of machine learning models. Five optimization techniques are discussed, these being, grid search, random search, bayesian search, genetic algorithm and particle swarm optimization, where a research is accomplished about solutions for integration of these techniques aiming to facilitate their application for the optimization of either the addressed models in this work or other learning models. The comparative analysis is done by exploring supervised learning tasks, using for classification databases and one regression database, with the objective of comparing the efficiency of each optimization technique in different problems and also comparing the influences of specific features of each model, such as the number of hyperpameters on each one.

Keywords: hyperparameters; optimization; grid search; random search; bayesian optimization; genetic algorithms; particle swarm optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A inteligência artificial é um dos campos de estudo mais promissores da atualidade, com aplicações em praticamente todas as áreas conhecidas, como engenharia, medicina, entretenimento, direito e política (SILVA *et al.*, 2021). Dentre as subáreas que mais têm se destacado no cenário atual da tecnologia podem ser citados o processamento de imagens e vídeos e processamento de linguagem natural (VALERI, 2020). A partir do surgimento de novas técnicas cada vez mais complexas e custosas computacionalmente para resolução de diferentes problemas deste campo, tem surgido um grande interesse na pesquisa de técnicas de otimização. A aplicação destas técnicas em modelos estatísticos pode trazer diversos benefícios no desempenho destes, como aumento significativo na taxa de acerto e melhoria em geral das métricas utilizadas para avaliar da qualidade do modelo (ALVES, 2021).

Sistemas de aprendizado de máquina utilizam modelos probabilísticos para resolver problemas. Esses modelos possuem um conjunto de características responsáveis por encontrar a melhor solução possível de um problema em uma determinada condição. As características do sistema que podem ser otimizadas com o treinamento do modelo para obtenção do resultado esperado são os chamados parâmetros. Porém, geralmente, o ajuste dos parâmetros pelo treinamento não são suficientes para obter bons resultados. Isso ocorre porque existem outros fatores relacionados à configuração do modelo, os quais também influenciam no desempenho e que não são otimizados por meio do ajuste do modelo, mas devem ser definidos antes desse ajuste. Estes fatores são os chamados hiperparâmetros. Os hiperparâmetros são configurações que impactam diretamente na eficiência e no aprendizado, e também na forma como os parâmetros são utilizados (DUTTA, 2022).

Uma boa parte dos problemas reais, onde o aprendizado de máquina é aplicado, é de alta complexidade, tais como processamento de imagens, áudio, vídeos e conjuntos de dados muito grandes (ARIWALA, 2022). Para esses problemas ocorre uma crescente demanda de capacidade e recursos de processamento. Apesar de recursos físicos serem pontos importantes a se levar em conta, pois podem reduzir muito o tempo de processamento, outro ponto igualmente importante é a otimização do modelo (SUN *et al.*, 2019). A otimização de hiperparâmetros procura analisar, de forma manual ou automática, diferentes variações de hiperparâmetros, visando chegar em uma variação que traz o melhor desempenho nos resultados dentre as testadas para maior acurácia, menor taxa de erro, entre outras métricas que podem ser utilizadas. Dentre os benefícios que esta análise traz estão: obter um maior entendimento sobre como as configurações

dos hiperparâmetros impactam no problema e sobre a relevância de alguns hiperparâmetros (BERGSTRA; BENGIOV, 2012).

A otimização dos hiperparâmetros é realizada por meio de técnicas de busca, como busca em grade (*grid search*), busca randômica (*random search*) e algoritmos genéticos. Essa otimização é feita aplicando um conjunto de variações de hiperparâmetros à configuração do modelo e analisando seus resultados após o processo de treino/teste. As métricas de validação são utilizadas para comparar cada variação e decidir qual, dentre o conjunto escolhido, melhor apresenta resultados para o problema específico.

Alguns algoritmos de otimização, como a Busca em Grade, buscam encontrar a configuração ótima testando todas as possíveis variações. Isso pode ser extremamente custoso, pois, considerando que cada classificador possui um conjunto de hiperparâmetros, testar todas as possibilidades aumentaria exponencialmente a complexidade a cada variação (DUFOUR; NEVES, 2019). Portanto, em alguns casos, principalmente se há recursos limitados ou se o espaço de busca que as variações dos hiperparâmetros geram é muito grande, a busca em grade não é uma boa escolha (BERGSTRA; BENGIOV, 2012).

Em vista disso, o presente trabalho se propõe a realizar uma comparação, na tarefa de otimização de hiperparâmetros, dos seguintes algoritmos: busca em grade, busca randômica, otimização bayesiana, algoritmos genéticos e otimização por enxame de partículas (sendo esses dois últimos algoritmos de otimização estocástica). Esses algoritmos são aplicados em problemas de aprendizado de máquina com o propósito de fazer uma comparação entre seus impactos no desempenho dos resultados, e consequentemente levar em conta sua utilização para obtenção de uma maior qualidade para a solução destes problemas.

1.1 Objetivos

- Realizar uma revisão bibliográfica sobre algoritmos aplicados a otimização de hiperparâmetros em problemas variados de classificação e regressão;
- Analisar a efetividade de algoritmos de otimização na melhoria do desempenho dos modelos por meio de medidas tais como F_1-Score e Acurácia;

2 TRABALHOS RELACIONADOS

A seguir são mostrados alguns dos trabalhos já existentes na literatura relacionados ao tema proposto, que serviram como inspiração para a pesquisa e desenvolvimento do presente trabalho.

Alibrahim e Ludwig (2021) realizaram uma análise de algumas técnicas de otimização de hiperparâmetros e fizeram uma comparação entre a abordagem de algoritmos genéticos e o uso de algoritmos de busca em grade e de otimização bayesiana. O conjunto de dados utilizado, disponível no site kaggle.com: Santander Customer Transaction Prediction Dataset, busca predizer se um cliente irá ou não fazer uma transação. Com os resultados obtidos, é possível chegar à conclusão de que a abordagem de algoritmos genéticos tem um melhor desempenho para o problema em questão.

Shekhar *et al.* (2022) realizaram um estudo comparativo entre diferentes ferramentas aplicadas a otimização de hiperparâmetros que usam técnicas baseadas em otimização bayesiana e algoritmos evolucionários. Doze modelos de classificadores foram analisados em problemas de larga escala, e um total de 58 hiperparâmetros foram otimizados para a escolha do melhor modelo. Concluiu-se, a partir dos experimentos realizados que: para o primeiro problema analisado (*CASH problem benchmark*) a ferramenta *Optuna* obteve melhor desempenho, já para segundo (*MLP problem*) a ferramenta *HyperOpt* a foi a melhor.

Bergstra e Bengiov (2012) fazem um estudo comparando a eficiência do algoritmo de busca randômica, com a de outros algoritmos, principalmente o de busca em grade, em vários problemas diferentes. Os autores mostram que para a maioria dos problemas, a busca em grade não é a melhor escolha de otimizador a ser utilizado, por ser um algoritmo que sofre de um problema de dimensionalidade, por causa do aumento exponencial das combinações conforme é aumentada a quantidade de hiperparâmetros utilizados. Como opção para a busca em grade, a busca randômica é uma opção razoavelmente melhor para a maioria dos casos analisados, principalmente com problemas de alta dimensionalidade e onde há baixa dimensionalidade efetiva (Alguns hiperparâmetros sendo mais relevantes para a performance do que outros).

KimHo-Chan e KangMin-Jae (2020) utilizam o banco de dados MNIST para comparar três métodos de otimização: busca em grade, busca randômica e a otimização bayesiana. Os resultados obtidos com esta última técnica são melhores do que com os outros dois métodos. Concluem, portanto, que deve ser considerado o uso da otimização bayesiana como alternativa para os algoritmos de busca em grade e busca randômica, os quais são computacionalmente

intensivos, e necessitam de mais tempo e recursos computacionais, em especial a busca em grade, que aumenta exponencialmente conforme aumentam os hiperparâmetros.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 Hiperparâmetros

Hiperparâmetros, em um sistema de aprendizado de máquina, são parâmetros da arquitetura do modelo de aprendizado. São configurações que devem ser determinadas antes do processo de treinamento e teste e que têm influência no desempenho do modelo (YANG; SHAMI, 2020). Um algoritmo de aprendizado A possui um conjunto de parametros de modelo (θ) que são ajustados no processo iterativo de treinamento do modelo. Além desses, temos os hiperparâmetros (λ), que influenciam diretamente os parâmetros do estimador. Por exemplo, em uma Máquina de Vetor de Suporte, temos o hiperparâmetro C (parâmetro de regularização)(BERGSTRA; BENGIOV, 2012). Em uma rede neural tem-se, como um segundo exemplo, o número de camadas e o número de neurônios por camada. Outros hiperparâmetros podem ser: o *batch size*, que diz respeito ao tamanho da amostra usada; taxa de aprendizado; taxa de *dropouts*; percentual usado para treino/teste; número de épocas (iterações) entre outros (NYUYTIYMBIY, 2022).

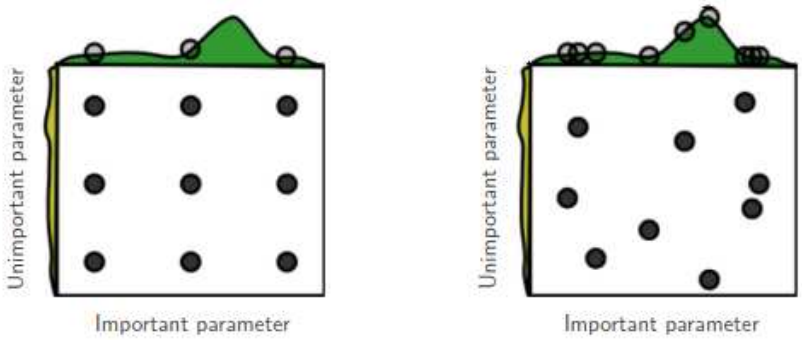
A otimização de hiperparâmetros consiste em encontrar o conjunto de hiperparâmetros que permite minimizar uma função de erro L aplicada a seu algoritmo A. Esse A é aplicado ao conjunto de dados de treino. Aplicando uma validação cruzada, o conjunto de hiperparâmetros ideais minimiza a função de erro médio do algoritmo aplicado ao conjunto de dados de validação (BERGSTRA; BENGIOV, 2012).

3.2 Busca em Grade

A busca em grade é um algoritmo que testa todas as possibilidades de combinação para os hiperparâmetros, dado um conjunto (uma grade) de combinações. Esse algoritmo realiza a avaliação do produto cartesiano desse conjunto, tornando essa técnica extremamente custosa, que cresce exponencialmente à medida que cresce a dimensionalidade, ou seja, conforme o espaço de configuração aumenta (HUTTER *et al.*, 2019).

Seja Λ um conjunto com K variáveis de configuração de um modelo. A busca em grade exige que seja escolhido um conjunto de valores para cada variável ($\mathbf{L_1}$, $\mathbf{L_2}$ ..., $\mathbf{L_K}$), em que $\mathbf{L_n}$ é um vetor de valores de cada variável. Temos, portanto, um conjunto de possibilidades formado pela combinação de todos os valores para cada variável. Esse fato faz com que a busca

Figura 1 – Ilustração representando a busca em grade (esquerda) e busca randômica (direita) aplicadas a uma função $f(x,y) = g(x) + h(y) \approx g(x)$.



Fonte: Bergstra e Bengiov (2012).

em grade sofra com a chamada "maldição da dimensionalidade", pois o espaço de possibilidades cresce exponencialmente com o número de hiperparâmetros (BERGSTRA; BENGIOV, 2012).

Pelo fato de a busca em grade ser um método exaustivo, que percorre todas as possibilidades de combinações, certamente o algoritmo encontrará a melhor configuração para o problema, dado um número suficiente de combinações. O que torna esse algoritmo problemático é a lentidão, o tempo excessivo levado para percorrer todas as possibilidades que muitas vezes o torna inviável (MALATO, 2022).

Na Figura 1, a ilustração a esquerda mostra a busca em grade em um problema de otimização com dois parâmetros, sendo um deles (x) mais importante do que o outro, ou seja, a função objetivo $f(x,y) = g(x) + h(y) \approx g(x)$ depende praticamente de um parâmetro. Um dos problemas da busca em grade é que, das 9 combinações para os dois parâmetros mostradas no gráfico, apenas 3 pontos do parâmetro x são explorados, dificultando encontrar o ponto ótimo, já que por mais que seja explorado 3 pontos distintos de y, esse parâmetro é pouco relevante para a otimização de f.

3.3 Busca Randômica

A busca randômica é um algoritmo que busca otimizar uma função a partir de uma faixa de números aleatórios dentro do domínio, realizando combinações dentro desse espaço de busca até que um critério de parada seja alcançado. Na otimização de hiperparâmetros, esse algoritmo é uma alternativa à busca em grade, pois ele substitui a seleção exaustiva de todas as combinações de hiperparâmetros por um intervalo de possibilidades, combinando aleatoriamente em cada iteração. Essa busca supera a busca em grade, especialmente quando

alguns hiperparâmetros afetam mais o desempenho do algoritmo de aprendizado do que outros (LIASHCHYNSKYI; LIASHCHYNSKYI, 2019).

Na Figura 1, a ilustração a direita mostra uma busca randômica aplicada à mesma função $f(x,y) = g(x) + h(y) \approx g(x)$, em que x é mais importante do que y. Diferentemente da busca em grade, na qual há uma limitação na exploração de pontos dos parâmetros, a busca randômica possui as mesmas 9 combinações, mas consegue explorar 9 pontos distintos de x. Com esse espaço de busca, é muito mais fácil encontrar o ponto ótimo da função f, se comparado com a busca em grade.

3.4 Algoritmos Genéticos

Algoritmo genético (AG) é uma técnica que utiliza conceitos trazidos das áreas biológicas de evolução natural, utilizado dentre outras áreas, para diversas aplicações de engenharia, na otimização de problemas computacionais complexos como otimização de redes neurais e solução de problemas de controle e fluxo (SRINIVAS; PATNAIK, 1994).

Em um algoritmo genético, o problema proposto é analisado, sendo extraídos, então, algumas das características e parâmetros relevantes para a resolução do problema. Essas características são traduzidas para o algoritmo como genes, e esse conjunto de genes compõe um indivíduo. Esse indivíduo representa, para o AG, um possível arranjo de atributos que leva à solução do problema, e o conjunto desses indivíduos constitui a população. O AG pode ser definido como um processo evolutivo onde a população evolui durante uma sequência iterações chamadas de gerações. Os indivíduos são avaliados para cada geração, ocorrendo um processo semelhante ao da evolução natural, em que os indivíduos com maior desempenho conforme as métricas de avaliação possuirão maior probabilidade de serem escolhidos para definir a próxima geração de indivíduos, enquanto os que tem desempenho ruim, vão sendo eliminados (SRINIVAS; PATNAIK, 1994).

Durante o processo de evolução, também é inserido um importante processo: a mutação. Este processo é inserida durante a etapa de recombinação genética, como uma forma de aumentar a robustez do cálculo, inserindo uma aleatoriedade a mais ao sistema e impedindo que haja uma rápida convergência, dentre os fatores, devido ao cruzamento entre indivíduos semelhantes, podendo levar o algoritmo a convergir em um mínimo local, a uma solução distante da ótima. (SRINIVAS; PATNAIK, 1994).

Os passos mais comumente realizados pelo algoritmo genético são descritos abaixo:

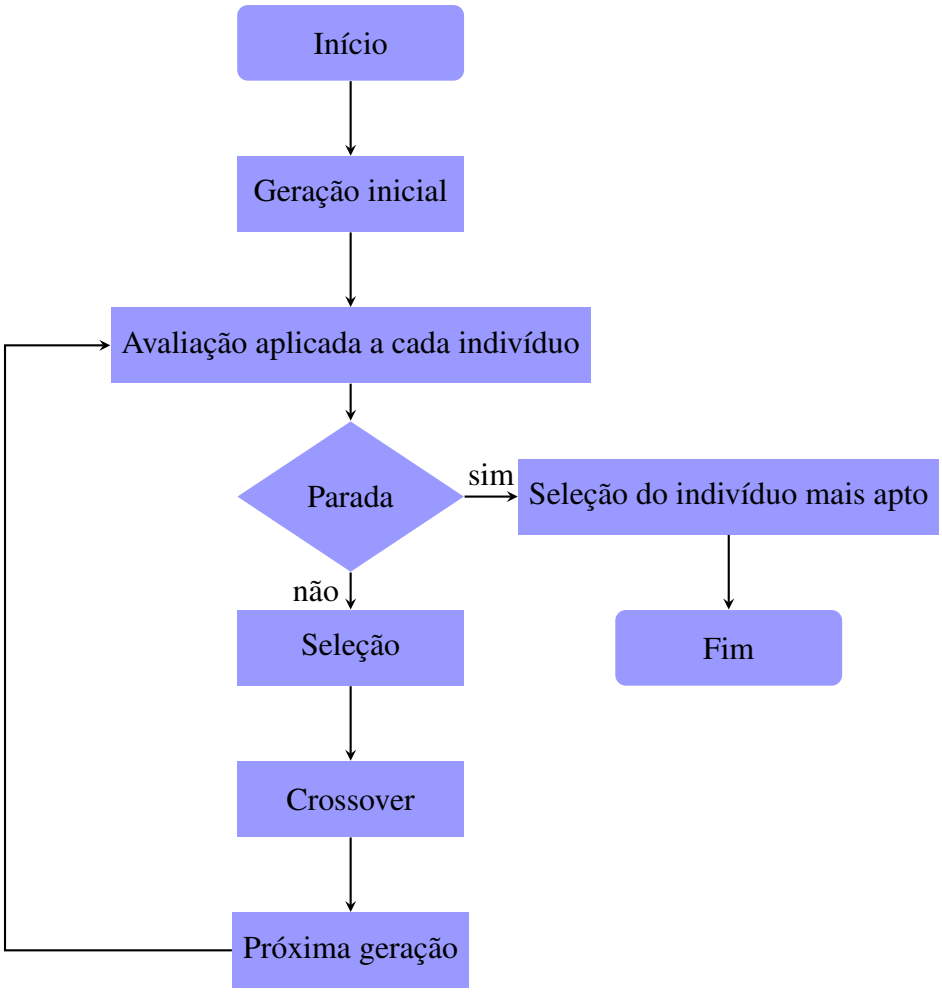
1. Inicialização: A população é iniciada, comumente de forma aleatória;
2. Avaliação: A função objetivo é aplicada a cada indivíduo, resultando em um valor de desempenho;
3. Seleção: Os indivíduos são selecionados com probabilidade proporcional ao seu valor de avaliação;
4. Reprodução (Crossover): Os indivíduos selecionados são agrupados em pares e seus atributos são recombinados entre si;
5. Mutação: Em alguns casos, após o crossover, ocorre a mudança do valor de um gene dentre os recombinados, causando assim, uma mutação aleatória.
6. Estabelecimento da nova geração: A nova geração de indivíduos recombinados substitui a geração anterior e o ciclo recomeça para ela;
7. Condição de parada: O algoritmo termina quando a condição de parada é alcançada. Exemplos de condição de parada podem ser: A solução mínima que satisfaz os critérios do problema é encontrada; um número máximo de gerações é encontrado; O sistema chega em um platô, não apresentando melhoras com as iterações.

O fluxograma que apresenta o ciclo de execução do algoritmo genético é ilustrado na Figura 2, mostrando os passos desde o início do algoritmo até o cumprimento da condição de parada.

3.5 Otimização Bayesiana

A Otimização Bayesiana (OB) utiliza dois conceitos chave: função probabilística substituta e função de aquisição. A função substituta tem o objetivo de se moldar ao comportamento da função objetivo, utilizando as amostras coletadas dela. Uma escolha comum à função substituta é o processo gaussiano, que consiste em uma função média e uma função covariância. A partir dos dados coletados, o processo gaussiano dá uma predição média para a função objetivo com um certo intervalo de confiança, criando uma curva estimada da função real com regiões de incerteza para cada ponto da curva. A função de aquisição, por sua vez, avalia o próximo ponto a ser escolhido como ótimo. A Melhora Esperada (*Expected Improvement*) é uma função comumente usada como função de avaliação. Essa função, de forma resumida, analisa o grau de melhora entre a próxima amostra e a melhor amostra já encontrada e, a partir de uma série de cálculos probabilísticos, determina o melhor ponto a ser escolhido como próximo ponto ótimo (KIMHO-CHAN; KANGMIN-JAE, 2020).

Figura 2 – Fluxograma representando o ciclo de um algoritmo genético.



Fonte: Elaborada pelo Autor.

Este algoritmo é iterativo e a curva é moldada a partir dessas duas funções em cada iteração, se aproximando cada vez mais da função objetivo, conseguindo, com isso, encontrar regiões mais pertinentes para explorar e encontrar melhores pontos ótimos (HUTTER *et al.*, 2019).

3.6 Otimização por Enxame de Partículas

Este subtópico baseia-se no livro de Engelbrecht (2007), o qual pode ser consultado para uma explicação mais aprofundada a respeito do algoritmo em questão.

A Otimização por Enxame de Partículas (*Particle Swarm Optimization*, PSO), é um algoritmo estocástico de otimização que foi inspirado no comportamento de pássaros voando em grupos e cardumes de peixes, que possuem características aleatórias mas também se comportam de forma coordenada, formando certas sincronias. No PSO temos a ideia de enxame e

partícula, que de uma forma análoga ao algoritmo genético, representaria população e indivíduo, respectivamente. Cada partícula do enxame representa um vetor de variáveis de dimensão n_x, e "sobrevoa"o espaço de busca n_x-dimensional.

O movimento de cada partícula do enxame leva em consideração a melhor posição encontrada até o momento por ela mesma e a melhor posição encontrada por sua vizinhança, e, a cada iteração, suas posições são atualizadas e avaliadas até que um critério de parada seja atingido. A posição da partícula x é dada pela equação (3.1):

$$x_i(t+1) = x_i(t) + v_i(t+1). \tag{3.1}$$

em que $x_i(t)$ é a posição da partícula i no instante de tempo t, e $v_i(t)$ é a velocidade dessa partícula.

O PSO possui duas variações básicas de algoritmo, o *lbest* (descrito acima) e o *gbest*. O *lbest* traz o conceito de vizinhança $\mathcal{N}_i$, e utiliza para o cálculo da velocidade a melhor posição encontrada pela vizinhança de cada partícula i. O *gbest*, o qual é utilizado pelo presente trabalho, é um caso especial do *lbest* que considera $n_{\mathcal{N}_i} = n_s$, com $n_{\mathcal{N}_i}$ sendo o tamanho da vizinhança $\mathcal{N}_i$ e n_s, o tamanho de todo o grupo de partículas. A equação (3.2) descreve o cálculo da velocidade da partícula i para o algoritmo *gbest*:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_2(t)\left[\hat{y}_j(t) - x_{ij}(t)\right] \tag{3.2}$$

onde $v_{ij}(t)$ é a velocidade da partícula i na dimensão $j = 1, 2, \ldots, n_x$; w é o **peso de inércia**, que controla a relevância de $v_{ij}(t)$ para o cálculo de $v_{ij}(t+1)$; c_1 e c_2 são os coeficientes de aceleração dos componentes e $r_1(t)$ e $r_2(t)$ são valores aleatórios obtidos no intervalo $[0, 1]$. A melhor posição $\hat{y}$ é dada por:

$$\hat{y}(t) = min\{f(x_0(t)), f(x_1(t)), \ldots, f(x_{n_s}(t)\} \tag{3.3}$$

O algoritmo *gbest* PSO é definido no Algoritmo 1.

3.7 Medidas de Avaliação

3.7.1 Matriz de Confusão

A Matriz de Confusão é uma tabela que tem como objetivo mostrar com mais detalhes, o comportamento das predições de um classificador. Essa tabela é composta por duas

Algoritmo 1: Algoritmo *gbest* PSO

```
p = partícula
Criar o enxame de dimensão n_x;
while condição de parada ≠ true do
    for p_i, i = 1,...,n_s do
        // Obter a melhor posição da partícula;
        if f(x_i) < f(y_i) then
            y_i = x_i;
        end
        // Obter a melhor posição global
        if f(y_i) < f(ŷ) then
            ŷ = y_i;
        end
    end
    for p_i, i = 1,...,n_s do
        Aplicar equação (3.2);
        Aplicar equação (3.1);
    end
end
```

ou mais linhas e colunas, onde as linhas representam as classes reais, e as colunas representam as classes estimadas. Em uma classificação binária, há duas linhas e duas colunas, onde os seguintes valores são evidenciados (KELLEHER *et al.*, 2015):

- Verdadeiro Positivo (VP): Valores que possuem predição e valor real positivos;
- Verdadeiro Negativo (VN): Valores que possuem predição e valor real negativos;
- Falso Positivo (FP): Valores que obtiveram predição positiva, mas que têm valor real negativo.
- Falso Negativo (FN): Valores que obtiveram predição negativa, mas que têm valor real positivo.

Em problemas de multiclasse, a matriz de confusão separa nas diagonais os valores verdadeiramente preditos para cada classe, e fora das diagonais, os falsamente preditos (SCIKIT-LEARN, 2022).

3.7.2 Acurácia

A acurácia avalia o percentual de acertos das predições. É a razão entre os valores corretamente preditos, com o total de predições do conjunto de dados. Por exemplo, se 9 de 10 predições foram corretas, a acurácia é de 0.9, ou 90%. A fórmula da acurácia para classificadores

binários pode ser dada da seguinte forma (THARWAT, 2021):

$$\text{Acurácia} = \frac{VP+VN}{VP+VN+FP+FN} \tag{3.4}$$

ou pode ser generalizada para classificadores binários ou multiclasses simplesmente como (SCIKIT-LEARN, 2022):

$$\text{Acurácia} = \frac{\text{Predições corretas}}{\text{Total de predições}} \tag{3.5}$$

Em geral, a acurácia não avalia bem a capacidade de o modelo classificar corretamente cada classe, sendo importante utilizar também na avaliação outras métricas como o F_1-Score.

3.7.3 F_1-Score

O F_1-*Score* é a média harmônica entre a precisão (p) e a sensibilidade (s), duas outras métricas dadas respectivamente pelas equações (3.6) e (3.7). É um caso específico do F_β-*Score* com $\beta = 1$, em que a precisão e a sensibilidade têm igual importância (TAHA; HANBURY, 2015).

$$p = \frac{VP}{VP+FP} \tag{3.6}$$

$$s = \frac{VP}{VP+FN} \tag{3.7}$$

portanto, o F_1-*Score* é definido por:

$$F_1\text{-}Score = \frac{2 \cdot p \cdot s}{p+s} \tag{3.8}$$

A equação (3.8) se aplica aos problemas de classificação binários. Para *datasets* multiclasse, o F_1-Score Macro pode ser utilizado. Esta medida pode ser definida como a média aritmética dos F_1-Scores de cada classe.

$$F_1\text{-}Score\ Macro = \frac{1}{N}\sum_{i=1}^{N} F_1\text{-}Score_i \tag{3.9}$$

em que N é o total de classes da base de dados (SCIKIT-LEARN, 2022).

3.7.4 Erro Médio Quadrático

O erro médio quadrático (*Mean Squared Error*, MSE) é uma métrica de risco, que corresponde à média dos quadrados dos erros de cada predição. Ou seja, o erro é a diferença entre predição e valor real, e o MSE é a média deste erro elevado ao quadrado (JAMES *et al.*, 2021).

Se temos um vetor $\hat{\mathbf{y}}$ de N predições e com $\mathbf{y}$ sendo o vetor de valores alvo reais, temos que o MSE é dado por:

$$MSE = \frac{1}{N}\sum_{1}^{N}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2. \tag{3.10}$$

3.7.5 Coeficiente de Determinação (R^2)

O Coeficiente de Determinação (R^2) é dado pela equação:

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \overline{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2}. \tag{3.11}$$

e pode ser definido como a razão entre a variância explicada pelas variáveis independentes do modelo e a variância total. Esta métrica também pode ser apresentada da seguinte forma:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2}. \tag{3.12}$$

em que $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\varepsilon_i^2$ é a soma residual da predição (GLANTZ *et al.*, 2016).

3.7.6 Ganho (percentual de aumento)

O ganho percentual é dado pela razão entre a métrica (por exemplo: de F_1-Score) obtida após executar dado algoritmo de otimização (m_o) e a mesma métrica obtida sem otimização (m_s), ou seja, com parâmetros padrão do modelo.

$$Ganho\ (\%) = \left(\frac{m_o}{m_s} - 1\right) \cdot 100. \tag{3.13}$$

4 MATERIAIS E MÉTODOS

Para a realização dos experimentos da pesquisa foram utilizados alguns bancos de dados conhecidos. A biblioteca desenvolvida do projeto possibilitou aplicar vários tipos de modelos de aprendizado nesses bancos de dados, bem como analisar a performance do uso de algoritmos de otimização para cada um desses modelos. Os materiais utilizados serão descritos a seguir.

4.1 Conjuntos de dados

As bases de dados utilizadas nos experimentos são: *MNIST*, *Wine Quality*, *Winscosin Breast Cancer*, o conjunto de dados de texto de *tweets* de linhas aéreas dos EUA e o *California Housing*, sendo este último utilizado para aplicação do modelo de regressão.

4.1.1 MNIST Dataset (dígitos escritos à mão)

O MNIST é um subconjunto reduzido de uma base de dados maior chamada NIST, que consiste em amostras de imagens de dígitos feitos à mão. Os dígitos dessa base de dados já passaram por uma normalização de tamanho, foram centralizados em imagens de tamanho fixo e transformados em vetores. O conjunto utilizado possui 5620 amostras, com 64 atributos e 10 classes (0 a 9). Essa base de dados está disponível em: http://yann.lecun.com/exdb/mnist/.

4.1.2 Wine Quality Dataset

Essa base de dados se refere a dados de variantes tinto e branco de um vinho português, chamado Vinho Verde. Esse banco de dados é conhecido e usado em problemas de classificação e regressão. As variáveis disponíveis dizem respeito a aspectos psico-químicos e sensoriais, e são 12: acidez fixa, acidez volátil, ácido cítrico, cloretos, açúcar residual, dióxido de enxofre livre, dióxido de enxofre total, densidade, pH, sulfatos, álcool e qualidade. Sendo a variável qualidade o rótulo, recebendo pontuação de 0 a 10. Esta abse possui 4898 amostras com 12 atributos e está disponível em: https://archive.ics.uci.edu/ml/datasets/Wine+Quality.

4.1.3 Winscosin Breast Cancer Dataset

Esse conjunto de dados é tirado de imagens digitalizadas de amostras de tecido mamário. Os atributos dos dados descrevem as características das células presente nas imagens. Os atributos, que possuem valores reais, são: raio, perímetro, áreas, suavidade, compacidade, concavidade, pontos côncavos, simetria, dimensão fractal. A variável usada como rótulo é o diagnóstico (M sendo maligno, B sendo benigno). Esta base possui 30 atributos e 569 amostras. O banco de dados e mais informações estão disponíveis em: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

4.1.4 California Housing Dataset

O *California Housing Dataset* é uma base de dados obtida a partir do censo dos EUA de 1990, para o qual cada linha do banco de dados representa dados de um grupo de quarteirão, que geralmente contém uma população de 600 a 3000 pessoas. Esse conjunto de dados possui 8 atributos: renda e idade medianas no grupo de quarteirões, número médio de cômodos por residência, número médio de quartos por família, população do grupo de blocos, número médio dos membros da família, longitude e latitude do grupo de blocos. O conjunto contém 20640 amostras, e a variável dependente é o preço médio da casa. Esse conjunto de dados encontra-se em: https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html.

4.1.5 Twitter US Airline Sentiment Dataset

Essa base de dados trata de uma varredura feita na rede social *Twitter*, onde os *tweets* foram extraídos e classificados como positivos, negativos ou neutros, e pela categorização dos motivos negativos (como "voo atrasado"ou "serviço grosseiro"). A avaliação dos *tweets* foi feita para 6 companhias aéreas. Este conjunto de dados possui 14640 amostras com 15 atributos e pode ser acessado em: https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment.

4.2 Algoritmos de Aprendizado

Os algoritmos utilizados nos experimentos são aplicados às bases de dados descritas anteriormente, e suas aplicações são usadas para a comparação dos métodos de otimização estudados nesta pesquisa. Três destes são modelos de classificação e um destes é um modelo de

regressão. Os hiperparâmetros destes algoritmos são listados a seguir:

4.2.1 Máquina de Vetor de Suporte (SVM)

- *Kernel*: Função de kernel que mapeia os dados originais em um espaço de dimensão superior. Útil quando não é possível encontrar um separador linear que se adequa aos dados (MORETTIN; SINGER, 2022). As funções de kernel utilizadas neste trabalho são as seguintes (SCIKIT-LEARN, 2023):
  1. Polinomial: $\left(\gamma\langle x, x'\rangle + r\right)^d$;
  2. RBF: $\exp\left(-\gamma\left\|x - x'\right\|^2\right)$;
  3. Sigmoidal: $\tanh\left(\gamma\langle x, x'\rangle + r\right)$.
- C: Nível de penalidade aplicado ao modelo para classificações erradas (MORETTIN; SINGER, 2022);
- γ: O parâmetro γ está relacionado ao nível de influência que as amostras selecionadas pelo modelo como vetores de suporte têm (SCIKIT-LEARN, 2023). Este hiperparâmetro é utilizado pelos kernels: polinomial, RBF e sigmoidal.
- r: Termo independente de algumas funções de kernel não lineares (SCIKIT-LEARN, 2023).

4.2.2 K vizinhos mais próximos (K Nearest Neighbors, KNN)

- K: Número K de vizinhos. Quanto maior for esse valor, menos o classificador será suscetível a ruídos (SCIKIT-LEARN, 2023).
- *Pesos*: Pesos atribuídos a cada vizinho. Se definido como *uniforme*, seu valor será distribuído uniformemente para todos os vizinhos; também é possível atribuir os pesos pela *distância*, ou seja, cada vizinho terá um peso proporcional ao inverso da sua distância ao ponto de consulta (SCIKIT-LEARN, 2023).
- *Métrica de Distância*: Métrica de distância a ser definida que melhor se adequa ao conjunto de dados. As usadas neste trabalho são as distâncias de Minkowski, Euclidiana, Manhattan, Braycurtis, Chebyshev e cosseno.

4.2.3 Naive Bayes Gaussiano

– Suavização da variância: O Naive Bayes Gaussiano é um classificador que se baseia no teorema de Bayes, e assume que a base de dados segue uma distribuição Gaussiana. O hiperparâmetro *suavização da variância* é um valor adicionado às variâncias, com o propósito de aumentar a estabilidade do modelo (SCIKIT-LEARN, 2023).

4.2.4 Regressão por Vetores de Suporte (SVR)

O algoritmo SVR possui as mesmas configurações de hiperparâmetros que o SVM, além do parâmetro ε.

– ε: É o valor máximo de erro tolerável, sem que nenhuma penalidade seja aplicada (MORETTIN; SINGER, 2022);
– *Tolerância*: Valor de tolerância para o critério de parada (SCIKIT-LEARN, 2023).

4.3 Faixa de Busca dos Hiperparâmetros

Os seguintes modelos foram escolhidos por serem bastante conhecidos e amplamente utilizados pela comunidade, além de terem quantidades diferentes de hiperparâmetros, que permite uma variação dos cenários onde serão aplicados os algoritmos de otimização. O conjunto de hiperparâmetros e suas respectivas faixas de busca foram escolhidos com base em pesquisas do uso da ferramenta, também pela comunidade. Esses hiperparâmetros são definidos, para cada modelo, nas seguintes subseções:

4.3.1 SVM

– *Kernel* (`kernel`): Gaussiano ou RBF (`rbf`), polinomial (`poly`) e sigmoidal (`sigmoid`). O valor padrão é `rbf`;
– C (`C`): Valores espaçados em escala logarítmica no intervalo $[10^{-6}, 10^{7}]$. O valor padrão é 1,0;
– γ (`gamma`) (relevante apenas se o *kernel* for RBF, polinomial ou sigmoidal): Valores espaçados em escala logarítmica no intervalo $[5^{-7}, 5^{1}]$. O valor padrão é $1/(n_a \cdot var(X))$, em que n_a é o número de atributos e $var(x)$ é a variância do vetor colapsado em uma dimensão;

– *r* (`coef_0`) (apenas para os *kernels* polinomial e sigmoidal): Variação logarítmica no intervalo $[10^{-2}, 10^{2}]$ O valor padrão é 0,0.

4.3.2 KNN

– *K* (`n_neighbors`): Espaço de busca com números inteiros, com intervalo $[1, 50]$. O valor padrão é 5;
– *Pesos* (`weights`): `uniform` ou `distance`. O valor padrão é `uniform`;
– *Métrica* (`metric`): `minkowski`, `euclidean`, `manhattan`, `braycurtis`, `chebyshev` e `cosine`. O valor padrão é `minkowski`.

4.3.3 Naive Bayes Gaussiano

– *Suavização de variância* (`var_smoothing`): Valores em escala logarítmica variando no intervalo $[2^{-9}, 2^{1}]$. O valor padrão é 1^{-9}.

4.3.4 SVR (Support Vector Regression)

– *Kernel* (`kernel`): Gaussiano ou RBF (`rbf`), polinomial (`poly`) e sigmoidal (`sigmoid`). O valor padrão é `rbf`;
– *C* (`C`): Valores espaçados em escala logarítmica no intervalo $[10^{-6}, 10^{7}]$. O valor padrão é 1,0;
– γ (`gamma`) (relevante apenas se o *kernel* for RBF, polinomial ou sigmoidal): Valores espaçados em escala logarítmica no intervalo $[5^{-7}, 5^{1}]$. O valor padrão é $1/(n_a \cdot var(X))$, em que n_a é o número de atributos e $var(x)$ é a variância do vetor colapsado em uma dimensão;
– *r* (`coef0`) (apenas para os *kernels* polinomial e sigmoidal): Variação logarítmica no intervalo $[10^{-2}, 10^{2}]$. O valor padrão é 0,0;
– ε (`epsilon`): Valores espaçados em escala logarítmica no intervalo $[2^{-6}, 2^{0}]$. O valor padrão é 0,1;
– *Tolerância* (`tol`): Variação logarítmica no intervalo $[10^{-6}, 10^{-2}]$. O valor padrão é 1^{-3}.

4.3.5 Configurações do espaço de busca e número de amostras

Para padronizar a quantidade de combinações de valores de hiperparâmetros testadas por cada algoritmo de otimização, as seguintes definições foram feitas:

Para o busca em grade, como este algoritmo testa todas as combinações possíveis dentro de um espaço de busca, foi definido que haveriam 200 combinações possíveis entre diferentes valores de hiperparâmetros. Isso foi possível para os modelos SVM, KNN e Naive Bayes. Como não foi possível obter este valor exato de combinações para o modelo SVR, definiu-se um valor aproximado de combinações possíveis (216).

Já para a busca randômica, em um espaço de busca bem superior à busca em grade, são selecionadas, aleatoriamente, 200 combinações de valores de hiperparâmetros (ou seja, são realizadas 200 iterações do algoritmo).

No caso do Algoritmo Genético, a quantidade de combinações de hiperparâmetros testada é estabelecida pelo tamanho da população e pelo número de gerações. Assim, de modo que este algoritmo testasse aproximadamente 200 combinações de valores, para o tamanho da população e o número de gerações foram definidos, respectivamente, os valores 12 e 35 para as bases *Breast Cancer*, *MNIST Digits*, *Wine Quality* e *California Housing*, e os valores 10 e 33 para a base de dados *Twitter Airline*.

Para os algoritmos PSO e otimização bayesiana, foram definidos 200 iterações para o algoritmo, assim como foi definido para a busca randômica.

A quantidade de valores possíveis para cada hiperparâmetro dos modelos de aprendizado está disponibilizado na Tabela 1.

4.4 Ferramentas utilizadas

O projeto foi desenvolvido utilizando a linguagem Python (versão 3.10), a qual possui uma ampla comunidade sempre pesquisando e trazendo novas soluções e ferramentas que facilitam aplicações na área de IA, e também em muitas outras áreas. Foram utilizadas também as seguintes ferramentas *open-source*:

1. **Numpy**: Um dos pacotes de base para computação científica em Python. Dentre vários outros recursos, traz a manipulação de vetores multidimensionais e operações de álgebra linear e estatística (NUMPY, 2023).
2. **Pandas**: Biblioteca que oferece vários recursos de analise de dados e manipulação de

Tabela 1 – Número de amostras utilizado na configuração dos espaços de busca dos hiperparâmetros para cada algoritmo de otimização

Algoritmo	C	kernel	gamma	coef_0
Busca em Grade	5	2	5	4
Outros	50	3	50	50

(a) Espaços de busca dos hiperparâmetros do SVM

Algoritmo	n_neighbors	weights	metric
Busca em Grade	25	2	4
Outros	50	2	6

(b) Espaços de busca dos hiperparâmetros do KNN

Algoritmo	var_smoothing
Busca em Grade	200
Outros	500

(c) Espaços de busca dos hiperparâmetros do *Naive Bayes*

Algoritmo	C	kernel	gamma	coef_0	epsilon	tol
Busca em Grade	3	2	3	3	2	2
Outros	50	2	50	50	50	50

(d) Espaços de busca dos hiperparâmetros do SVR

estruturas de dados em *dataframes*[1] (PANDAS, 2023).

3. **Matplotlib**: Biblioteca amplamente utilizada para criação de gráficos e visualizações de dados em Python (MAPLOTLIB, 2023).
4. **Scikit-learn**: Biblioteca de aprendizado de máquina que possui recursos para aprendizado supervisionado e não supervisionado, além de oferecer outras ferramentas para preprocessamento de dados, treinamento, seleção e avaliação de modelos, entre outros recursos (SCIKIT-LEARN, 2023).
5. **Jupyterlab**: É um ambiente de desenvolvimento interativo, que facilita a execução de experimentos computacionais e visualização dos resultados em gráficos (JUPYTER, 2023).
6. **Sklearn-deap**: Biblioteca simples desenvolvida para criar uma integração entre a scikit-learn e a biblioteca de algoritmos genéticos *DEAP*[2]. De modo a utilizar esta biblioteca com as versões mais novas do *scikit-learn*, correções de erros precisaram ser feitas. A biblioteca corrigida está disponível em: https://github.com/andrerodrig/sklearn-deap.
7. **Sklearn-pso**: Biblioteca simples que implementa uma integração entre a scikit-learn e a biblioteca de PSO chamada *PySwarms*[3]. De modo semelhante à biblioteca *sklearn-deap*,

[1] Estrutura de dados tabular e indexada.
[2] DEAP é uma biblioteca que implementa algoritmos evolucionários. https://deap.readthedocs.io/en/master/
[3] PySwarms é uma biblioteca que implementa o algoritmo de PSO. https://pyswarms.readthedocs.io/en/latest/

algumas correções precisaram ser feitas para que esta biblioteca fosse utilizada com as versões mais novas do *scikit-learn* e *numpy* A biblioteca corrigida está disponível em: https://github.com/andrerodrig/sklearn-pso.

8. **Scikit-optimize**: Biblioteca criada a partir do scikit-learn, a qual implementa diversos métodos de otimização, como a otimização bayesiana (SCIKIT-OPTIMIZE, 2023).

4.5 Uso de Pipelines

Para manter a construção do modelo de aprendizado organizada, é usado o conceito de pipelines. Pipelines são objetos que dividem as etapas de preprocessamento (seleção de atributos, normalização das entradas, redução de dimensionalidade, etc.) e de aplicação dos modelos de aprendizagem em passos. A biblioteca scikit-learn implementa esses pipelines como objetos que possuem os métodos `fit`, que treina o classificador, e `predict`, que prediz o resultado para o dado de teste. A seguir é mostrado um trecho de código em que um objeto `Pipeline` é definido.

```
pipeline = Pipeline(
    [
        ('preprocess', StandardScaler()),
        ('clf', SVC()),
    ]
)
pipeline.fit(X_train, y_train)
pipeline.predict(X_test, y_test)
```

No exemplo acima, o objeto `Pipeline` abstrai todos os passos de preprocessamento (descrito como `preprocess`) e aplicação do classificador SVM (descrito como `clf`). Essa abstração é bastante útil quando se tem um grande número de passos desde o preprocessamento até a aplicação do modelo de aprendizado (RASCHKA; MIRJALILI, 2017).

4.6 Uso do módulo BaseEstimator da biblioteca scikit-learn

Neste trabalho, utiliza-se da classe base do scikit-learn chamada `BaseEstimator` para criar um módulo de `Transformer` customizado, que nesse caso, tem como objetivo abstrair

a etapa de preprocessamento da base de dados *Twitter Airline*. `Transformers`, no scikit-learn, são objetos que são responsáveis pelo preprocessamento e transformação dos dados, fornecendo como saída a base de dados preprocessada pronta para aplicação do modelo de aprendizado. Um `Transformer` herda tanto a classe base `BaseEstimator` quanto da classe `TransformerMixin`, as quais provem uma série de métodos que integram o módulo criado com a API do scikit-learn. Dessa forma, caso algum usuário utilize a classe `BaseEstimator` para implementar um novo modelo, este modelo poderá utilizar, além dos métodos padrão `fit`, `predict`, `transform` entre outros disponibilizados na API do scikit-learn, todos os algoritmos de otimização mostrados neste trabalho.

4.7 Definição dos Parâmetros dos Algoritmos de Otimização

Cada algoritmo de otimização possui seu próprio conjunto de parâmetros, que podem ser chamados de **metaparâmetros**. Os metaparâmetros foram definidos, assim como as faixas de buscas dos hiperparâmetros, também com base em pesquisas de uso das ferramentas utilizadas. A seguir, são definidos os metaparâmetros utilizados durante este trabalho.

1. `cv` (*cross-validation*): Define a estratégia de divisão para a validação cruzada. Nos experimentos do presente trabalho, estratégia K-Fold com 5 *folds* foi usada para as bases de dados *Breast Cancer*, *MNIST Digits* e *Wine Quality*, e essa mesma estratégia, com 3 *folds* foi utilizada para as bases *Twitter Airline* e *California Housing*.
2. `scoring`: Estratégia para avaliar a desempenho da validação cruzada durante a otimização. Nos experimentos são usados o F_1-Score nos modelos de classificação, e o R^2 no modelo de regressão.
3. `n_iter` (para os algoritmos de busca randômica e otimização bayesiana): Número de configurações de hiperparâmetros amostradas (número de iterações). Esse parâmetro afeta diretamente o tempo de execução da busca, pois quanto mais iterações houver, mais tempo a otimização levará para ser finalizada. Nos experimentos, como já dito antes, foi escolhido 200 iterações por padrão, exceto para os casos onde não é possível obter exatamente este número.
4. `population_size` (Para o algoritmo genético): Tamanho da população. O padrão usado nos experimentos é de 12 para os três bancos de dados de classificação simples e para o de regressão e 10 para a base do Twitter.
5. `gene_mutation_prob` (para o algoritmo genético): A probabilidade de mutação de um

gene. O padrão usado é 0,05.

6. `gene_crossover_prob` (para o algoritmo genético): Probabilidade de ocorrer cruzamento entre os genes. O padrão dos experimentos é de 0,5.
7. `generations_number` (Para o algoritmo genético): Número de gerações, onde o AG é aplicado iterativamente a cada nova população criada. Neste trabalho será definido um número de 35 gerações para os três primeiros bancos de dados de classificação e para a base de regressão, e 33 para o *dataset* do Twitter.
8. `n_particles` (para o PSO): Número de partículas do enxame. Foi definido o número de 10 partículas para este trabalho.
9. `iterations` (para o PSO): Número de iterações em que o algoritmo será executado nas partículas. Para o PSO, o número total de iterações é o produto deste parâmetro com o `n_particles`, sendo assim, para manter o padrão de 200 iterações foi determinado o valor de 20 para este parâmetro.
10. `n_jobs`: Controla o número de CPUs que serão usadas em paralelo (*jobs*) para a execução da busca. Para este projeto, considerando os recursos da máquina utilizada para execução dos experimentos, foram definidos, exceto da otimização bayesiana, 2 *jobs* para a execução das buscas para os três primeiros bancos de dados de classificação, 7 *jobs* para a base do Twitter, 7 *jobs* para a regressão na base *California Housing* e 7 *jobs* para a execução da otimização bayesiana para todos os casos.

5 RESULTADOS E DISCUSSÕES

Neste capítulo são mostrados os resultados das execuções dos experimentos, onde será comparada a eficiência das técnicas de otimização apresentadas anteriormente. Essas técnicas foram aplicadas aos 4 modelos de aprendizado apresentados, e estes modelos foram testados nas 5 bases de dados descritas. Para as tarefas de classificação, as técnicas são comparadas em função do tempo de execução, ganho de F_1-Score e da acurácia. Já no caso dá regressão, as técnicas são comparadas em função do tempo de execução e do R^2. As buscas foram realizadas da seguinte forma:

- Para as bases *Breast Cancer*, *MNIST Digits* e *Wine Quality*, foram obtidos valores mais altos de Acurácia e F_1-Score utilizando como métrica de avaliação da busca a Acurácia;
- Para o banco *Twitter Airline*, foram obtidos melhores ganhos tendo o F1-Score dos modelos como métrica de busca;
- Para o modelo de regressão, foi utilizado como métrica de avaliação o R^2;

5.1 Comparação dos Resultados

Primeiramente, os modelos de aprendizado de máquina foram aplicados, aos bancos de dados, sem que seus hiperparâmetros fossem otimizados. Os resultados de F_1-Score e Acurácia, para os dados de classificação, são mostrados na Tabela 2. Já o resultado de R^2, para a base de dados de regressão, é mostrado na Tabela 3.

Tabela 2 – Resultados dos modelos não otimizados aplicados aos bancos de dados de classificação.

Banco de dados	SVM	KNN	NB
Breast Cancer	0,982	0,965	0,918
MNIST Digits	0,974	0,967	0,850
Wine Quality	0,981	0,944	0,963
Twitter Airline S.	0,861	0,713	0,671

(a) Acurácia

Banco de dados	SVM	KNN	NB
Breast Cancer	0,986	0,972	0,936
MNIST Digits	0,974	0,967	0,851
Wine Quality	0,978	0,944	0,961
Twitter Airline S.	0,562	0,452	0,425

(b) F_1-Score

Tabela 3 – Resultados do modelo de regressão não otimizado aplicado ao banco de dados California Housing.

Métrica	SVR
R^2	0,724

A partir da Tabela 2, pode-se perceber que os algoritmos, mesmo sem otimização de hiperparâmetros, já possuem altas taxa de Acurácia e F_1-Score para os bancos de dados *Breast Cancer*, *MNIST Digits* e *Wine Quality*. Neste caso, apenas o banco de dados *Twitter Airline* se mostrou uma tarefa um pouco mais complexa.

5.1.1 Breast Cancer Dataset

A Figura 3 e a Tabela 4 evidenciam os resultados obtidos a partir da aplicação dos modelos, após a otimização dos hiperparâmetros, no banco de dados *Breast Cancer*. Analisando essa Tabela, não foi notado grandes mudanças com relação à Acurácia de teste e o F_1-Score para os 5 métodos de otimização. Principalmente para o modelo de melhor desempenho, o SVM, que permaneceu com 0,991, como melhor valor encontrado na busca. Todas as técnicas obtiveram um ganho de 0,51%, que, a depender do problema, não é um ganho de desempenho relevante. Para o KNN, o PSO conseguiu um maior ganho percentual com relação aos outros algoritmos. A otimização bayesiana não foi eficiente na convergência, obtendo um F_1-Score menor que o obtido sem recorrer a otimizações. Como mostra a Figura 3, o algoritmo bayesiano obteve um valor de ganho percentual negativo, de -1,75%.

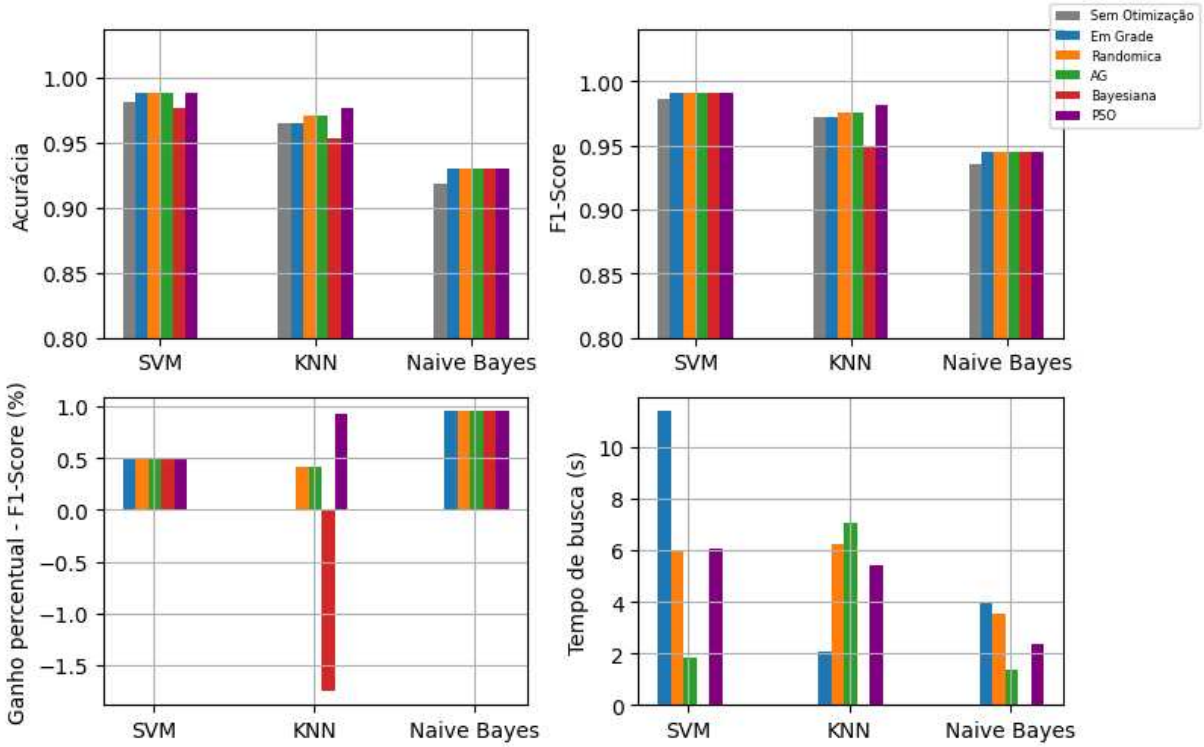
Tabela 4 – Comparação das métricas obtidas pelo melhor estimador encontrado para cada técnica de otimização - *Breast Cancer Dataset*

Busca	Modelo	Acurácia	F_1-Score	Ganho (F_1)	Tempo de Busca
Sem Otimização	SVM	0,982	0,986		
	KNN	0,965	0,972		
	Naive Bayes	0,918	0,936		
em Grade	SVM	0,988	0,991	0,51%	11,40s
	KNN	0,965	0,972	0,00%	2,10s
	Naive Bayes	0,930	0,945	0,96%	4,02s
Randômica	SVM	0,988	0,991	0,51%	5,93s
	KNN	0,971	0,976	0,41%	6,25s
	Naive Bayes	0,930	0,945	0,96%	3,52s
AG	SVM	0,988	0,991	0,51%	1,82s
	KNN	0,971	0,976	0,41%	7,06s
	Naive Bayes	0,930	0,945	0,96%	1,38s
Bayesiana	SVM	0,988	0,991	0,51%	31,89min
	KNN	0,953	0,950	-2.26%	28,67min
	Naive Bayes	0,930	0,945	0,96%	23,28min
PSO	SVM	0,988	0,991	0,51%	6,09s
	KNN	0,977	0,981	0,92%	5,44s
	Naive Bayes	0,930	0,945	0,96%	2,38s

Outro ponto a destacar-se da tabela é a discrepância no tempo de busca do algoritmo bayesiano, com relação às outras técnicas. Este leva um intervalo de 31,89 minutos para executar as 200 iterações. Ao finalizar as iterações, o resultado F_1-Score foi semelhante ao encontrado na busca em grade, a qual demorou 11,40 segundos para executar o mesmo número de iterações.

O valor de tempo da otimização bayesiana foi ocultada, na Figura 3, do gráfico de tempo de busca dos modelos, com finalidade de facilitar a visualização.

Figura 3 – Comparação do desempenho das buscas para otimização dos 3 modelos de classificação - Breast Cancer Dataset



Fonte: Elaborada pelo Autor.

5.1.2 MNIST Digits Dataset

Diferentemente do caso abordado na subseção anterior, para o banco de dados MNIST Digits, a busca randômica gerou o maior ganho de F_1-Score e acurácia para o classificador SVM, seguida pelos algoritmos PSO e AG. Isto pode ser observado na Tabela 5.

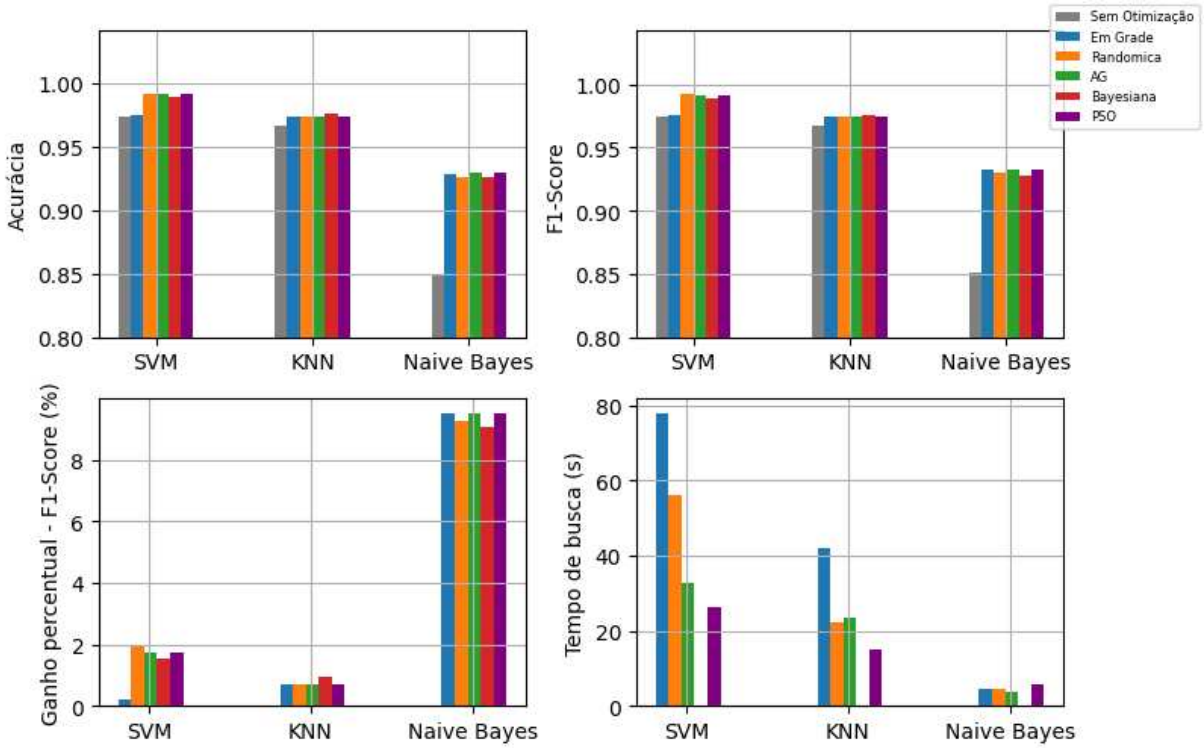
Novamente a busca bayesiana não se mostrou ser uma solução eficiente, obtendo ganhos percentuais menores do que as outras técnicas e com um tempo de processamento muito grande para a natureza simples do banco de dados em questão.

Nos gráficos da Figura 4, é mostrada a comparação dos algoritmos no caso do

Tabela 5 – Comparação das métricas obtidas pelo melhor estimador encontrado para cada técnica de otimização - *MNIST Dataset*

Busca	Modelo	Acurácia	F_1 Macro	Ganho (F_1)	Tempo de Busca
Sem Otimização	SVM	0,974	0,974		
	KNN	0,967	0,967		
	Naive Bayes	0,850	0,851		
em Grade	SVM	0,975	0,976	0,20%	1.30min
	KNN	0,974	0,974	0,72%	41.80s
	Naive Bayes	0,929	0,932	9.51%	4.51s
Randômica	SVM	0,992	0,993	1.95%	56.00s
	KNN	0,974	0,974	0,72%	22.21s
	Naive Bayes	0,926	0,930	9.28%	4.58s
AG	SVM	0,991	0,991	1.74%	32.86s
	KNN	0,974	0,974	0,72%	23.45s
	Naive Bayes	0,930	0,932	9.51%	3.86s
Bayesiana	SVM	0,989	0,989	1.54%	39.27min
	KNN	0,976	0,976	0,93%	33.96min
	Naive Bayes	0,926	0,928	9.05%	25.54min
PSO	SVM	0,991	0,991	1.74%	26.19s
	KNN	0,974	0,974	0,72%	14.84s
	Naive Bayes	0,930	0,932	9.51%	5.78s

Figura 4 – Comparação do desempenho das buscas para otimização dos 3 modelos de classificação - MNIST Dataset



Fonte: Elaborada pelo Autor.

MNIST Dataset. Contrastando-se com o gráfico da Figura 3, nota-se que, ao aplicar ao modelo SVM outros algoritmos ao invés da busca em grade, são descobertos melhores valores tanto de acurácia, quanto de F_1-Score. É possível verificar, além disso, que as otimizações oferecem um ganho relativo maior e em um intervalo menor para o modelo Naive Bayes. O ganho de F_1-Score das outras técnicas de otimização não superaram o valor máximo encontrado da busca em grade, de 0,932. A maior eficiência das buscas para o Naive Bayes pode estar relacionada ao fato de o modelo somente possuir um hiperparâmetro a ser otimizado, a suavização da variância (`var_smoothing`).

5.1.3 Wine Quality Dataset

Analisando a tabela 6 e os gráficos da Figura 5, nota-se que, para o SVM, foi mais eficiente a execução sem otimização, se comparado com o uso dos algoritmos de busca em grade (a qual não obteve ganho algum), busca randômica, AG e PSO (os quais obtiveram valores piores do que as configurações padrão). No entanto, ao contrário dos experimentos anteriores, a busca bayesiana foi, para este *dataset*, a única que conseguiu otimizar o modelo, além disso, chegando a um F_1-Score de 1,0, os tempos de execução da busca bayesiana também foram ocultados, na Figura 5, dos gráficos do tempo de busca para esse banco de dados, como forma de melhorar a visualização.

Para o modelo KNN, o uso do algoritmo bayesiano não foi eficiente, pois além do grande tempo de processamento de 28,19 minutos, a busca chegou a 0,928 para F_1-Score, valor inferior ao obtido utilizando as configurações padrão. Já para o Naive Bayes, a otimização bayesiana e o PSO obtiveram valores acima dos encontrados pelas outras técnicas.

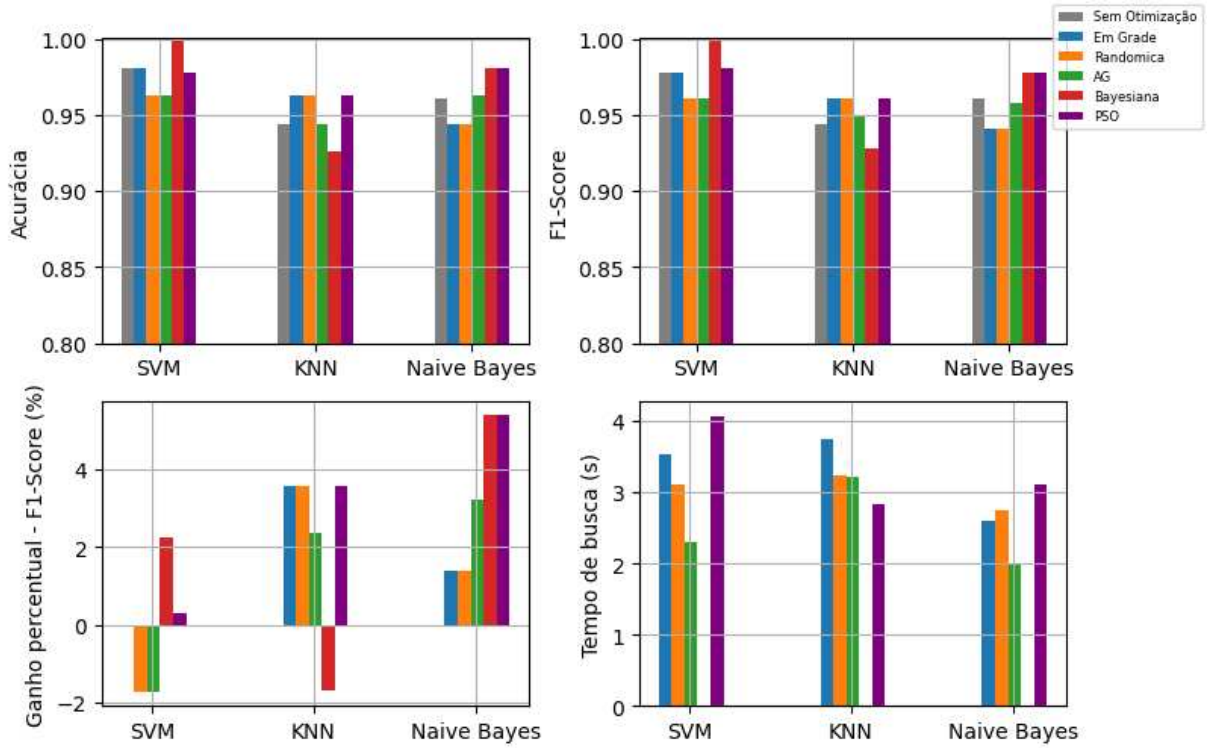
5.1.4 Twitter Airline Sentiment Dataset

A aplicação dos modelos de classificação nesse banco de dados trouxe alguns desafios. Por ser uma base de dados composta de 3 classes (comentários neutros, positivos e negativos) e esta ser muito desbalanceada, mesmo com a otimização de hiperparâmetros, os resultados de acurácia e F_1-Score foram considerados ruins. Sendo maioria dos dados, de *tweets* negativos, impactou de forma considerável a qualidade dos modelos, conseguindo separar bem a classe negativa, mas não sendo capaz de classificar de forma razoável as outras duas classes. Assim, foram removidos os dados da classe neutra, transformando esta base de dados em um problema de classificação binário (comentários positivos e negativos). O banco de dados

Tabela 6 – Comparação das métricas obtidas pelo melhor estimador encontrado para cada técnica de otimização - *Wine Quality Dataset*

Busca	Modelo	Acurácia	F_1 Macro	Ganho (F_1)	Tempo de Busca
Sem Otimização	SVM	0,981	0,978		
	KNN	0,944	0,944		
	Naive Bayes	0,963	0,961		
em Grade	SVM	0,981	0,978	0,00%	3,53s
	KNN	0,963	0,961	3,56%	3,74s
	Naive Bayes	0,944	0,941	1.40%	2,59s
Randômica	SVM	0,963	0,961	-1.74%	3,10s
	KNN	0,963	0,961	3,56%	3,23s
	Naive Bayes	0,944	0,941	1.40%	2,74s
AG	SVM	0,963	0,961	-1.74%	2,31s
	KNN	0,944	0,950	2,37%	3,22s
	Naive Bayes	0,963	0,958	3,23%	1,98s
Bayesiana	SVM	1,000	1,000	2,25%	37,74min
	KNN	0,926	0,928	-1,69%	28,19min
	Naive Bayes	0,981	0,978	5,39%	20,48min
PSO	SVM	0,978	0,981	0,31%	4,07s
	KNN	0,963	0,961	3,56%	2,83s
	Naive Bayes	0,981	0,978	5,39%	3,12s

Figura 5 – Comparação do desempenho das buscas para otimização dos 3 modelos de classificação - Wine Quality Dataset



Fonte: Elaborada pelo Autor.

resultante continuou sendo desbalanceado (79.5% de dados representando comentários negativos e 20,5% representando comentários positivos), porém, a avaliação das possíveis melhorias provocadas pelos algoritmos de otimização ficaram mais simples de ser avaliados. Os resultados obtidos para esta configuração estão dispostos na Figura 6 e na Tabela 7.

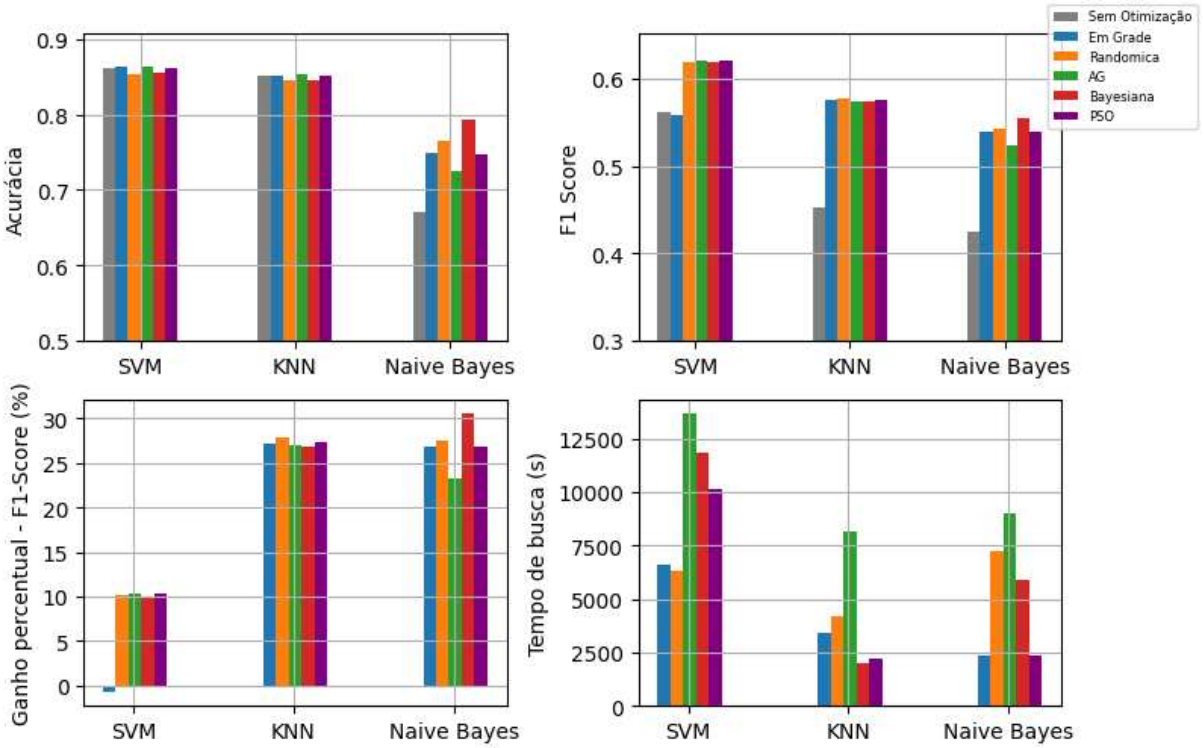
Tabela 7 – Comparação das métricas obtidas pelo melhor estimador encontrado para cada técnica de otimização - *Twitter Airline Dataset*

Busca	Modelo	Acurácia	F_1-Score	Ganho (F_1)	Tempo de Busca
Sem Otimização	SVM	0,861	0,562		
	KNN	0,713	0,452		
	Naive Bayes	0,671	0,425		
em Grade	SVM	0,864	0,558	-0,71%	1,83h
	KNN	0,854	0,575	27,21%	56,78min
	Naive Bayes	0,750	0,539	26,82%	39,77min
Randômica	SVM	0,854	0,619	10,14%	1,76h
	KNN	0,846	0,578	27,88%	1,17h
	Naive Bayes	0,766	0,542	27,53%	2,01h
AG	SVM	0,864	0,620	10,32%	3,80h
	KNN	0,854	0,574	26,99%	2,26h
	Naive Bayes	0,726	0,524	23,29%	2,51h
Bayesiana	SVM	0,855	0,618	9,96%	3,29h
	KNN	0,845	0,573	26.77%	33,65min
	Naive Bayes	0,794	0,555	30,59%	1,64h
PSO	SVM	0,861	0,620	10,32%	2,81h
	KNN	0,852	0,576	27.43%	36,92min
	Naive Bayes	0,748	0,539	26.82%	39,86min

Sem otimização, o modelo SVM atingiu um F_1-Score de 0,562. Já com a utilização dos algoritmos AG e PSO, atingiu-se um valor de 0,620 para essa métrica, melhorando-a em 10,32% (ou seja, obtendo uma classificação mais balanceada). Vale pontuar o baixo desempenho da busca em grade nesse modelo, que ao invés de otimizar a métrica em questão, chegou em um valor inferior ao encontrado sem otimização, enquanto as outras técnicas conseguiram, em geral, melhorar as métricas de F_1-Score.

Em relação aos outros dois modelos (KNN e Naive Bayes), os quais sem otimização obtiveram baixo desempenho (abaixo de 0,5) levando em conta a métrica F_1-Score, o uso dos algoritmos de otimização trouxe um ganho percentual bem maior comparado aos ganhos do SVM, chegando a um ganho de 27,88% para a busca randômica aplicada ao KNN, e a 30,59% com da otimização bayesiana, para o Naive Bayes. Isso mostra que, mesmo utilizando modelos mais simples, estes podem ter desempenho satisfatório caso seus hiperparâmetros sejam otimizados.

Figura 6 – Comparação do desempenho das buscas para otimização dos 3 modelos de classificação - Twitter Airline Dataset



Fonte: Elaborada pelo Autor.

Analisando os tempos de busca, observa-se que o AG apresentou uma maior demora, em comparação aos outros. Contrastando-se às aplicações das técnicas aos bancos de dados anteriores, em que a busca bayesiana se apresentou extremamente demorada, levando intervalos de aproximadamente 30 minutos, enquanto as outras técnicas por padrão demoraram poucos segundos.

5.1.5 California Housing Dataset

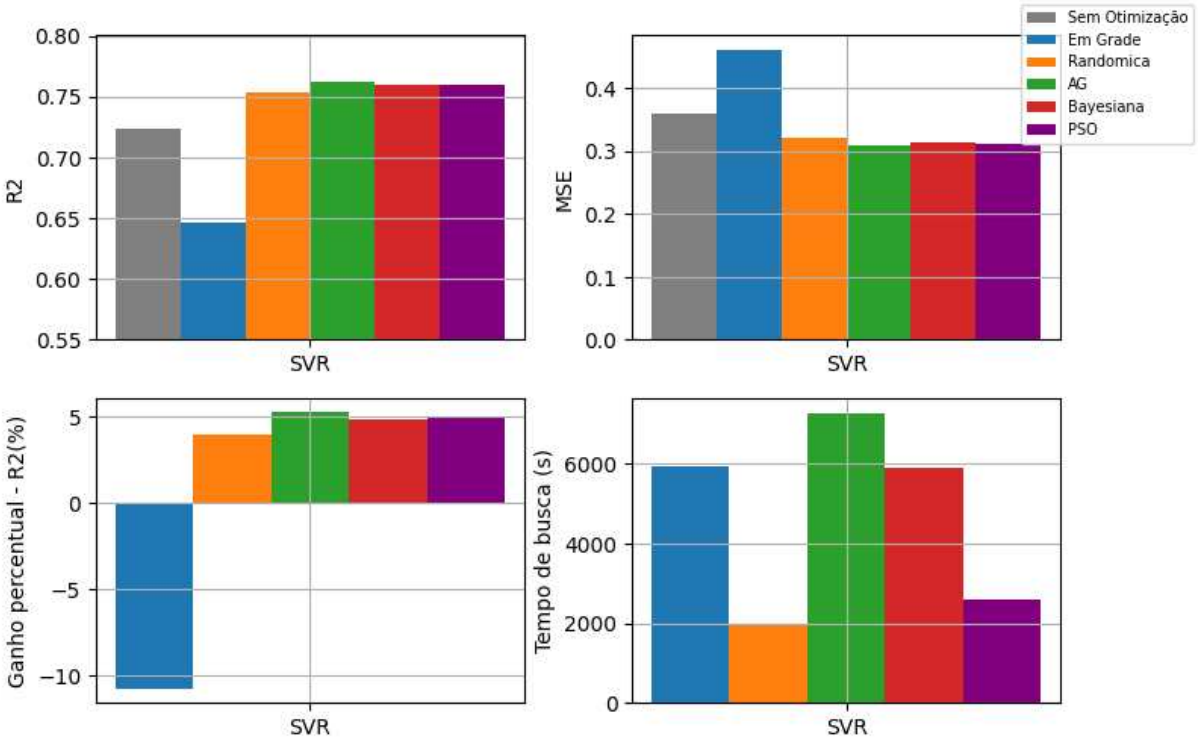
Tabela 8 – Comparação das métricas obtidas pelo melhor estimador encontrado para cada técnica de otimização aplicada ao modelo SVR - *California Housing Dataset*

Busca	R^2	Ganho (R^2)	Tempo de Busca
Sem Otimização	0,724		
em Grade	0,646	-10,77%	1,64h
Randômica	0,753	4,00%	32,80min
AG	0,762	5,25%	2,02h
Bayesiana	0,759	4,83%	1,63h
PSO	0,760	4,97%	43,37min

Este banco de dados pode ser caracterizado como um problema de regressão. Assim,

o SVR foi o modelo escolhido para comparar os métodos de otimização, visto que, como este modelo possui 6 hiperparâmetros, pode-se analisar as implicações de um número maior de hiperparâmetros nas limitações do espaço de busca para o algoritmo de busca em grade, e observar como isto difere dos outros algoritmos que não são afetados por essa característica do modelo. Pela Tabela 8 e pelos gráficos da Figura 7, pode-se inferir que a busca em grade não foi uma boa escolha de otimizador, pois obteve um valor bem inferior ao obtido com valores padrão do SVR. Todos os outros algoritmos conseguiram otimizar as medidas em questão, sendo o AG o mais eficiente, obtendo maior ganho de R^2. Porém, dentre os 5 métodos de otimização, este foi o que demandou mais tempo de execução.

Figura 7 – Comparação do desempenho das buscas para otimização do modelo de regressão - California Housing Dataset



Fonte: Elaborada pelo Autor.

5.1.6 Discussões Gerais

Apesar de a busca em grade apresentar uma capacidade mais limitada de busca do que os outros algoritmos de otimização, esta se mostra eficiente para problemas que envolvem bancos de dados e modelos de aprendizado simples, que não possuam uma grande variedade de hiperparâmetros. Ainda assim, mesmo com o SVM, em que a faixa de busca estava bem

limitada, esta técnica conseguiu obter bons resultados, alguns melhores do que as outras técnicas utilizadas. A busca randômica obteve resultados levemente superiores a busca em grade, para esses bancos de dados mais simples.

Faz-se necessário avaliar pela simplicidade do banco de dados/modelo de aprendizado, a necessidade de utilizar outras técnicas além da busca em grade ou se é realmente necessário fazer o uso da otimização de hiperparâmetros, pois conforme os testes constatam, modelos que já apresentam boas métricas, não apresentam ganhos significativos após a busca. Em alguns casos, como na otimização bayesiana, pode ser acrescentado um custo de tempo desnecessário, que a depender do problema, pode não convergir em bons resultados.

Para o caso do banco de dados de análise de sentimentos, entretanto, onde envolve uma dimensionalidade consideravelmente maior do que os anteriores, a busca em grade não teve uma eficiência considerável, obtendo percentuais de otimização bem inferiores aos das outras técnicas, como a busca randômica e o algoritmo genético. Evidencia-se, para casos como este de bancos de dados mais complexos, a necessidade de buscar outras opções para otimização, como AG, e PSO. Utilizar o PSO ou um algoritmo genético seria uma boa escolha, pois eles conseguiram convergir e obter resultados melhores do que a busca randômica.

Para o modelo de regressão, a limitação do espaço de busca para obter-se 200 iterações teve um evidente impacto na capacidade de otimização da busca em grade, pois não foi possível chegar a valores de hiperparâmetros próximos ao padrão utilizado na biblioteca pelo amplo espaço de busca, obtendo assim um resultado de R^2 inferior ao obtido sem otimização. Enquanto isso, as outras buscas conseguiram encontrar valores superiores bem próximos entre si. Portanto, para a otimização de bancos de dados mais complexos e modelos que possuem um número maior de hiperparâmetros, é mais vantajoso o uso de algoritmos em que o número de iterações não seja definido pelas combinações de valores de cada hiperparâmetro.

Sobre as configurações de hiperparâmetros encontradas pelas buscas, nota-se que para os modelos com menos hiperparâmetros, as buscas encontram valores máximos com configurações de hiperparâmetros semelhantes, já em modelos com um número maior de hiperparâmetros, as buscas costumam chegar em valores bem diferentes de alguns atributos, que em alguns casos apresentam métricas semelhantes. As Tabelas 9 e 10 apresentam o conjunto de hiperparâmetros encontrados para as bases de dados *Twitter Airline* e *California Housing*, respectivamente.

Os valores de suavização da variância encontrados para o modelo Naive Bayes

transitaram entre 0,19 e 0,21, obtendo valores de F_1-Score e acurácia aproximados. Para o modelo KNN, todos os algoritmos convergiram para os mesmos valores, exceto o hiperparâmetro do número de vizinhos, que variou entre os valores de 10 e 18. Já para os modelos SVM e SVR, a maior quantidade de hiperparâmetros fizeram com que, para os parâmetros que possuem maior espaço de busca, as buscas chegassem a valores bem diferentes, enquanto a maioria dos algoritmos de otimização encontraram a função RBF para o hiperparâmetro `kernel`. O AG e PSO obtiveram o mesmo valor de F_1-Score para o SVM, mesmo com valores distintos de hiperparâmetros. Já para o SVR, a configuração encontrada pelo AG, que obteve o melhor resultado em termos de R^2, foi próxima à encontrada pelo PSO, o qual também obteve métricas bem próximas às obtidas pelo AG.

Tabela 9 – Tabela das melhores configurações de hiperparâmetros encontradas em cada busca - *Twitter Airline Dataset*

Modelo	Hiperparâmetros	Em Grade	Randômica	AG	OB	PSO
SVM	`C`	2	14,087	0,296	44,334	8,657
	`kernel`	`rbf`	`rbf`	`poly`	`rbf`	`rbf`
	`gamma`	0,2	1,0334	0,0441	0,3853	1,0183
	`coef0`	4,64	1,60	15,26	0,01	6,76
KNN	`n_neighbors`	16	10	18	10	14
	`weights`	`distance`	`distance`	`distance`	`distance`	`distance`
	`metric`	`cosine`	`cosine`	`cosine`	`cosine`	`cosine`
NB	`var_smoothing`	0,2000	0,2079	0,1918	0,2179	0,2001

Tabela 10 – Tabela das melhores configurações de hiperparâmetros encontradas em cada busca - *California Housing Dataset*

Modelo	Hiperparâmetros	Em Grade	Randômica	AG	OB	PSO
SVR	`C`	128,0	35,331	6,751	3,235	7,997
	`kernel`	`rbf`	`rbf`	`rbf`	`rbf`	`rbf`
	`gamma`	0,0080	0,1642	0,3612	0,3612	0,2734
	`coef0`	0,01	0,08	26.83	2.81	0,07
	`epsilon`	0,0156	0,0185	0,1994	0,3317	0,1898
	`tol`	1.000e-6	7.906e-6	1.326e-4	2.947e-5	2.299e-4

Por fim, considerando quantidades semelhantes de combinações de hiperparâmetros testadas, a ordem de grandeza dos tempos de busca dos algoritmos de otimização foram equivalentes.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho aplicou 5 técnicas de otimização de hiperparâmetros em uma base de dados de regressão, três bases simples e uma relativamente mais complexa de classificação, visando analisar de forma comparativa sua eficiência em diferentes modelos de aprendizado. Durante os experimentos, dentre os principais pontos de discussão, destacam-se os ganhos relativos para as bases de dados mais simples, os quais foram ou idênticos, ou muito próximos, mesmo utilizando-se de técnicas diferentes, e convergindo para configurações de hiperparâmetros diferentes, como no SVM. Conclui-se a partir disso que os algoritmos de otimização tendem a encontrar valores próximos a um ponto máximo. Os algoritmos AG e PSO na maioria das vezes conseguem convergir consideravelmente bem para valores próximos a esse máximo.

Deve-se, portanto, ponderar a escolha do tipo de técnica de otimização levando em conta variáveis como simplicidade do banco de dados, eficiência do modelo não otimizado, quantidade de hiperparâmetros que o modelo permite variar e eficiência da técnica, que a depender da simplicidade do modelo e do banco de dados é suficiente utilizar a busca em grade. Podemos citar o modelo Naive Bayes utilizado nos experimentos do presente trabalho, que foi otimizado utilizando-se um único hiperparâmetro. Para este caso, o uso da busca em grade não foi fortemente afetado pelo aumento do espaço de busca. Já o espaço de busca do algoritmo busca em grade aplicado a modelos com mais hiperparâmetros, como o SVM e SVR, é bem afetado e não se consegue um grande aumento do espaço de busca sem aumentos drásticos do número de iterações, e consequentemente, do tempo de processamento.

Para trabalhos futuros, considera-se ampliar a pesquisa com a utilização de bases de dados maiores, semelhantes a *Twitter Airline*, utilizada nesta pesquisa, e de tipos variados, como imagens, áudio, texto entre outros. Aplicação de outros modelos tais como MLP, redes neurais recorrentes e convolucionais, as quais possuem uma maior variedade de hiperparâmetros. Utilizar, além disso, uma variedade maior de algoritmos de busca para ampliar a comparação, como o **Algoritmo de Colônia de Formigas** (ENGELBRECHT, 2007), **Algoritmo de Tempera Simulada** (EREN *et al.*, 2017), **Simplex** (OZAKI *et al.*, 2017) e **Otimização Baseada em Gradiente Descendente** (MACLAURIN *et al.*, 2015). Pretende-se ainda ampliar, melhorar e tornar pública a biblioteca construída para organizar e facilitar a execução dos experimentos, permitindo um grande reuso de código.

REFERÊNCIAS

ALIBRAHIM, H.; LUDWIG, S. A. Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. 2021 IEEE Congress on Evolutionary Computation (CEC), 2021.

ALVES, V. D. Otimização de hiperparâmetros arquiteturais em redes adversárias generativas. Repositório Institucional da UFSC, 2021.

ARIWALA, P. **9 Real-World Problems that can be Solved by Machine Learning**. 2022. Disponível em: https://marutitech.com/problems-solved-machine-learning/. Acesso em: 21/07/2023.

BERGSTRA, J.; BENGIOV, Y. Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, 2012.

DUFOUR, J.-M.; NEVES, J. Chapter 1 - finite-sample inference and nonstandard asymptotics with monte carlo tests and r. In: VINOD, H. D.; RAO, C. (Ed.). **Conceptual Econometrics Using R**. Elsevier, 2019, (Handbook of Statistics, v. 41). p. 3–31. Disponível em: https://www.sciencedirect.com/science/article/pii/S0169716119300367.

DUTTA, M. **Impact of Hyperparameters on a Deep Learning Model**. 2022. Disponível em: https://www.analyticsvidhya.com/blog/2022/05/impact-of-hyperparameters-on-a-deep-learning-model/. Acesso em: 20/09/2022.

ENGELBRECHT, A. P. (Ed.). **Computational Intelligence: An Introduction**. [*S. l.*]: The MIT Press, 2007. ISBN 9780470035610.

EREN, Y.; KüçüKDEMIRAL İbrahim B.; ÜSTOğLU İlker. Chapter 2 - introduction to optimization. In: ERDINç, O. (Ed.). **Optimization in Renewable Energy Systems**. Boston: Butterworth-Heinemann, 2017. p. 27–74. ISBN 978-0-08-101041-9. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780081010419000028.

GLANTZ, S.; SLINKER, B.; NEILANDS, T. B. (Ed.). **Primer of Applied Regression and Analysis of Variance, Third Edition**. [*S. l.*]: MCGRAW-HILL EDUCATION / MEDICAL, 2016. ISBN 978-0-07-182411-8.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). **Automated Machine Learning - Methods, Systems, Challenges**. [*S. l.*]: Springer, 2019.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. (Ed.). **An Introduction to Statistical Learning**. [*S. l.*]: Springer, 2021.

JUPYTER. **Jupyter Documentation**. 2023. Disponível em: https://jupyter.org/. Acesso em: 02/04/2023.

KELLEHER, J. D.; NAMEE, B. M.; D'ARCY, A. (Ed.). **Machine Learning for Predicted Data Analysis: Algorithms, Worker Examples, and Case Studies**. [*S. l.*]: The MIT Press, 2015.

KIMHO-CHAN; KANGMIN-JAE. Comparison of hyper-parameter optimization methods for deep neural networks. **Journal of the Institute of Electrical and Electronics Engineers**, Korean Institute of Electrical and Electronics Engineers, v. 24, n. 4, p. 969–974, 12 2020.

LIASHCHYNSKYI, P.; LIASHCHYNSKYI, P. Grid search, random search, genetic algorithm: A big comparison for NAS. **CoRR**, abs/1912.06059, 2019. Disponível em: http://arxiv.org/abs/1912.06059.

MACLAURIN, D.; DUVENAUD, D.; ADAMS, R. P. **Gradient-based Hyperparameter Optimization through Reversible Learning**. 2015.

MALATO, G. **Hyperparameter tuning. grid search and Random Search**. 2022. Disponível em: https://www.yourdatateacher.com/2021/05/19/hyperparameter-tuning-grid-search-and-random-search/#:~:text=Grid\%20search\%20is\%20the\%20simplest,performance\%20metrics\%20using\%20cross\%2Dvalidation. Acesso em: 30/09/2022.

MAPLOTLIB. **Matplotlib Documentation**. 2023. Disponível em: https://matplotlib.org/. Acesso em: 02/04/2023.

MORETTIN, P. A.; SINGER, J. da M. (Ed.). **Estatística e Ciência de Dados**. [*S. l.*]: LTC, 2022. ISBN 8521638167.

NUMPY. **Numpy Documentation**. 2023. Disponível em: https://numpy.org/doc/stable/. Acesso em: 02/04/2023.

NYUYTIYMBIY, K. **Parameters and hyperparameters in machine learning and Deep Learning**. Towards Data Science, 2022. Disponível em: https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac. Acesso em: 20/09/2022.

OZAKI, Y.; YANO, M.; ONISHI, M. Effective hyperparameter optimization using nelder-mead method in deep learning. **IPSJ Transactions on Computer Vision and Applications volume**, Nov 2017.

PANDAS. **Pandas Documentation**. 2023. Disponível em: https://pandas.pydata.org/docs/index.html. Acesso em: 02/04/2023.

RASCHKA, S.; MIRJALILI, V. (Ed.). **Python Machine Learning - Second Edition: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow**. [*S. l.*]: Packt Publishing, 2017.

SCIKIT-LEARN. **3.3. Metrics and scoring: quantifying the quality of predictions**. 2022. Disponível em: https://scikit-learn.org/stable/modules/model_evaluation.html. Acesso em: 27/11/2022.

SCIKIT-LEARN. **Scikit-Learn Documentation**. 2023. Disponível em: https://scikit-learn.org/stable/getting_started.html. Acesso em: 02/04/2023.

SCIKIT-OPTIMIZE. **Scikit-Optimize Documentation**. 2023. Disponível em: https://scikit-optimize.github.io/stable/getting_started.html. Acesso em: 02/04/2023.

SHEKHAR, S.; BANSODE, A.; SALIM, A. A comparative study of hyper-parameter optimization tools. **CoRR**, abs/2201.06433, 2022. Disponível em: https://arxiv.org/abs/2201.06433.

SILVA, E. B. da; PAGANOTTI, G. A.; MELLO, L. B. de. **Inteligência Artificial: realidade, mitos e projeções para o futuro**. 2021. Disponível em: http://www.each.usp.br/petsi/jornal/?p=2834. Acesso em: 11/11/2022.

SRINIVAS, M.; PATNAIK, L. M. Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, 1994.

SUN, S.; CAO, Z.; ZHU, H.; ZHAO, J. **A Survey of Optimization Methods from a Machine Learning Perspective**. 2019.

TAHA, A. A.; HANBURY, A. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. **BMC Med Imaging**, Aug 2015. ISSN 2634-1964.

THARWAT, A. Classification assessment methods. **Applied Computing and Informatics**, Jan 2021. ISSN 2634-1964. Disponível em: https://doi.org/10.1016/j.aci.2018.08.003.

VALERI, V. **[Inteligência Artificial] Os avanços na deep learning estão aumentando a visão computacional**. 2020. Disponível em: https://www.oficinadanet.com.br/tecnologia/31409-inteligencia-artificial-os-avancos-na-deep-learning-estao-aumentando-a-visao-computacional. Acesso em: 11/11/2022.

YANG, L.; SHAMI, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. **Neurocomputing**, v. 415, p. 295–316, 2020. ISSN 0925-2312. Disponível em: https://www.sciencedirect.com/science/article/pii/S0925231220311693.