UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ENGENHARIA DE COMPUTAÇÃO

JOSÉ MARCOS MIRANDA NEVES

OTIMIZAÇÃO DE HIPERPARÂMETROS EM MACHINE LEARNING UTILIZANDO UMA SURROGATE E ALGORITMOS EVOLUTIVOS

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO

2020

JOSÉ MARCOS MIRANDA NEVES

OTIMIZAÇÃO DE HIPERPARÂMETROS EM MACHINE LEARNING UTILIZANDO UMA SURROGATE E ALGORITMOS EVOLUTIVOS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso II, do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para obtenção do título de Bacharel.

Orientador: Danilo Sipoli Sanches

CORNÉLIO PROCÓPIO

2020

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Cornélio Procópio
Diretoria de Graduação e Educação Profissional
Departamento de Computação
Engenharia de Computação

TERMO DE APROVAÇÃO

OTIMIZAÇÃO DE HIPERPARÂMETROS EM MACHINE LEARNING UTILIZANDO UMA SURROGATE E ALGORITMOS EVOLUTIVOS

por

José Marcos Miranda Neves

Este Trabalho de Conclusão de Curso de graduação foi julgado adequado para obtenção do Título de Bacharel em Engenharia de Computação e aprovado em sua forma final pelo Programa de Graduação em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Cornélio Procópio, 16/07/2020

_____

Prof. Dr. Danilo Sipoli Sanches

_____

Profa. Dra. Natássya Barlate Floro da Silva

_____

Prof. Dr. Adriano Rivolli da Silva

"A Folha de Aprovação assinada encontra-se na Coordenação do Curso"

RESUMO

MIRANDA NEVES, J. M.. OTIMIZAÇÃO DE HIPERPARÂMETROS EM MACHINE LEARNING UTILIZANDO UMA SURROGATE E ALGORITMOS EVOLUTIVOS. 36 f. Trabalho de Conclusão de Curso – Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2020.

Este trabalho apresenta uma nova abordagem para otimização de hiperparâmetros em algoritmos de *Machine Learning*. A ideia é construir uma *surrogate* através de pontos semi-aleatórios gerados com o método de Sobol e então utilizar um algoritmo evolutivo (DE ou PSO neste caso) para realizar a otimização sobre ela. Também é implementado uma forma de realizar mais de uma otimização com o algoritmo evolutivo em uma mesma execução do método sem aumentar o custo computacional em relação a outros métodos de otimização. O objetivo é então verificar se a utilização dessas duas estratégias tonaria o método proposto menos propenso a ficar preso em mínimos locais e também mais consistente se comparado a outros. O foco do trabalho foi a otimização do LightGBM aplicado a classificação binária, porém é possível expandir para outras áreas fazendo as devidas adaptações. Além do método proposto, são utilizados *Random Search* e *Bayesian Optimization* para realizar a otimização do LightGBM com 3 conjuntos de dados. Os resultados obtidos mostram que a abordagem proposta com a utilização do PSO consegue ser a mais consistente dos 3 métodos, porém *Bayesian Optimization* ainda se sai melhor no geral.

Palavras-chave: Otimização de hiperparâmetros, Machine Learning, Evolução diferencial, LightGBM, Surrogate, PSO

ABSTRACT

MIRANDA NEVES, J. M.. OPTIMIZATION OF MACHINE LEARNING HYPERPARAMETERS BY USING A SURROGATE AND EVOLUTIONARY ALGORITHMS. 36 f. Trabalho de Conclusão de Curso – Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2020.

This work presents a new approach for hyperparameter optimization in Machine Learning algorithms. The idea is to build a surrogate with quasirandom numbers generated by Sobol's algorithm and then use an evolutionary algorithm (DE or PSO in this case) to perform the optimization on it. It is also implemented a way of performing more than one optimization with the evolutionary algorithm in the same execution of the method without increasing the computational cost in relation to other optimization methods. The objective is then to verify if the use of these two strategies would make the proposed method less likely to be trapped in a local minimum and also make it more consistent when compared to others. The focus of this work was the optimization of LightGBM applied to binary classification, however it is possible to expand it to other areas by making the necessary adaptations. In addition to the proposed method, Random Search and Bayesian Optimization are also used to optimize LightGBM models trained on 3 datasets. The results obtained show that the proposed approach with PSO is the most consistent of the 3 methods, however Bayesian Optimization still performs better in general.

Keywords: Hyperparameter optimization, Machine Learning, Differential evolution, LightGBM, Surrogate, Particle Swarm Optimization

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

GBMs	Gradient Boosting Machines
MARS	Multivariate Adaptive Regression Splines
DE	Differential Evolution
PSO	Particle Swarm Optimization
RBF	Radial Basis Function
APIs	Application Programming Interfaces
GPU	Graphics Processing Unit
ROC	Receiver Operating Characteristics
AUC	Area under the ROC Curve
GAM	Generalized Additive Model

SUMÁRIO

1 INTRODUÇÃO

Machine Learning, que é um ramo da Inteligência Artificial, baseia-se na ideia de que sistemas computacionais podem aprender com dados e serem capazes de identificar padrões, com uma mínima intervenção humana (SAS, 2019). Fatores como os crescentes volume e variedade de dados, processamento computacional mais poderoso e acessível e maior capacidade de armazenamento de informações (SAS, 2019) facilitaram e ampliaram significativamente a aplicação de *Machine Learning* nos últimos anos. Sua utilização inclui uma ampla gama de exemplos, tais como prevenção de fraudes, identificação de *insights* e tendências, sistemas de recomendação, análise de risco, entre outros, proporcionando a diversos setores da sociedade melhoria contínua de seus processos e serviços.

Algoritmos de *Machine Learning* possuem dois tipos de parâmetros, aqueles que são ajustados automaticamente durante o seu treinamento, como os pesos de uma rede neural, e aqueles que precisam ser especificados manualmente antes do processo de treinamento, os quais são chamados de hiperparâmetros. A escolha certa desses hiperparâmetros pode determinar se o modelo (resultado do algoritmo após o treinamento) obterá uma performance satisfatória ou não. Tal escolha porém pode ser uma tarefa simples, como especificar o número de vizinhos K do *K-nearest neighbors* (JAMES et al., 2013), ou complexa, como no caso do LightGBM, um *framework* para *Gradient Boosting Machines* (GBMs), que possui mais de 10 hiperparâmetros (LIGHTGBM, 2019). Os algoritmos com maior potencial para lidar com problemas reais, como redes neurais e GBMs, são também os que mais dependem da otimização de seus hiperparâmetros, por isso essa área tem ganhado bastante atenção nos últimos anos e, embora já possua abordagens bastante aceitas, ainda é um campo com potencial de desenvolvimento.

Para realizar essa otimização, é necessário avaliar o desempenho do modelo obtido após o treinamento utilizando várias combinações de hiperparâmetros. Tal processo, considerando a complexidade de treinamento do algoritmo de *Machine Learning* e a quantidade de dados utilizada, a qual tem se tornado cada vez maior como mencionado antes, pode ter um custo computacional bastante alto. Mesmo utilizando computadores atuais com alto poder de processamento ou até *clusters*, em muitos problemas reais haverá um limite do número de

avaliações de desempenho do modelo que podem ser realizadas para que o processo termine em um período de tempo viável, principalmente em ambientes em que muitos modelos precisam ser treinados em um determinado intervalo de tempo.

Os métodos de otimização mais comumente utilizados são o manual, *Grid Search*, *Random Search* e *Bayesian Optimization*, os quais serão melhor detalhados posteriormente neste trabalho. Os dois primeiros têm sua viabilidade limitada a casos em que o número de hiperparâmetros é pequeno, além de não garantirem que a solução encontrada seja a melhor (JOHNSON, 2017). *Random Search*, por se tratar de um método completamente aleatório, necessita de sorte para conseguir um bom resultado, sendo que quanto mais hiperparâmetros estiverem envolvidos maior será o número de avaliações necessário para se obter um resultado satisfatório (JOHNSON, 2017). *Bayesian Optimization* tem ganhado cada vez mais atenção por obter melhores resultados que os outros métodos em diversas situações (SNOEK et al., 2012; EGGENSPERGER et al., 2015) e ainda necessitando de um número menor de avaliações. Seu diferencial consiste em que, ao contrário dos demais, ele utiliza informações de avaliações passadas para escolher um novo conjunto de hiperparâmetros a ser testado (SHAHRIARI et al., 2016).

Embora *Bayesian Optimization* possa ser considerado o estado da arte em otimização de hiperparâmetros, ele ainda possui algumas limitações, como necessidade de um maior número de avaliações à medida que a dimensionalidade do problema aumenta e a possibilidade de ocorrer uma otimização local (BROCHU et al., 2010). Este trabalho, então, propõe uma nova abordagem para otimização de hiperparâmetros. A qual consiste na utilização de uma *surrogate* (explicada posteriormente) construída com pontos gerados por uma *quasirandom sequence*, como uma forma de diminuir a quantidade de avaliações de desempenho do modelo, e um algoritmo evolutivo para realizar a otimização sobre a *surrogate*.

O foco do trabalho será a otimização de hiperparâmetros do LightGBM aplicado a problemas de classificação binária, considerando que GBMs são bastante populares atualmente e altamente dependentes de otimização. Serão utilizados o método de Sobol como *quasirandom sequence*, o algoritmo de regressão MARS (Multivariate Adaptive Regression Splines) para gerar a *surrogate* e evolução diferencial, ou DE (Differential Evolution), e PSO (Particle Swarm Optimization) como algoritmos evolutivos.

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

Verificar se a abordagem de otimização apresentada consegue resultados melhores ou mais consistentes do que os obtidos com *Random Search* e *Bayesian Optimization*. O que também permitiria verificar a hipótese de que um método de otimização que utiliza uma *surrogate* construída sem um processo iterativo estaria menos sujeito a ficar preso em mínimos locais do que métodos com um processo iterativo, que é o caso de *Bayesian Optimization*.

1.1.2 OBJETIVOS ESPECÍFICOS

- Entender os métodos de otimização de hiperparâmetros já existentes.
- Definir 3 conjuntos de dados que envolvem classificação binária para realização dos experimentos.
- Definir o conjunto de hiperparâmetros do LightGBM que serão otimizados.
- Gerar conjuntos de hiperparâmetros utilizando Sobol sequence para cada um dos 3 conjuntos de dados.
- Avaliar diferentes algoritmos de regressão que serão utilizados para construir a *surrogate* e utilizar o que obtiver melhor desempenho.
- Investigar quais configurações utilizar para os algoritmos DE e PSO, levando em conta que se trata de um problema de otimização de números reais e inteiros em conjunto.
- Definir uma metodologia para comparar o desempenho dos métodos de otimização de forma a obter uma estimativa de qual se sair melhor ou é mais consistente.
- Utilizar a abordagem apresentada para realizar a otimização dos hiperparâmetros do LightGBM utilizando os 3 conjuntos de dados.
- Utilizar *Random Search* e *Bayesian Optimization* para realizar a otimização nos 3 conjuntos de dados.
- Fazer um comparativo entre os 3 métodos de otimização utilizados.

1.1.3 ORGANIZAÇÃO DO TEXTO

O trabalho está dividido em cinco capítulos, no primeiro foi dado uma introdução geral sobre o trabalho. O segundo capítulo será a apresentação de toda a teoria para o desenvolvimento do trabalho, apresentando explicações mais detalhadas sobre conceitos e algoritmos mencionados na introdução, e também de trabalhos relacionados. O terceiro capítulo apresenta como a pesquisa foi realizada, descrevendo tecnologias e métodos utilizados e configurações de algoritmos. O quarto capítulo é a apresentação dos resultados obtidos no trabalho e a análise dos mesmos. E por fim, a conclusão no quinto capítulo, contextualizando o que foi desenvolvido e apresentando as limitações e o que pode ser feito em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são discutidos os principais conceitos necessários para o entendimento do trabalho e também apresentados trabalhos relacionados.

2.1 CLASSIFICAÇÃO E REGRESSÃO

Considerando um conjunto de dados (X, Y), em que X = $\{x_1, x_2, ..., x_n\}$ corresponde a n observações com m atributos cada e Y = $\{y_1, y_2, ..., y_n\}$ corresponde às saídas de cada observação. O objetivo de um modelo de *Machine Learning*, considerando a área de aprendizado supervisionado, é estimar uma função $\hat{F}$ tal que $\hat{y} = \hat{F}(x)$, ou seja, utilizar o conjunto de dados para aprender a relação entre x e y e com isso ser capaz de estimar a saída de observações ainda não conhecidas (MURPHY, 2012).

Quando a saída assume um valor categórico, $y \in \{1, 2, ...C\}$ sendo C o número de classes, trata-se de um problema de classificação. Já quando a saída assume um valor contínuo, $y \in \mathbb{R}$, fala-se em regressão (MURPHY, 2012). Quando há apenas duas classes o problema denomina-se classificação binária, sendo geralmente consideradas as classes 0 (classe negativa) e 1 (classe positiva). Nesse caso, em vez do modelo gerar uma classe como saída para uma nova observação, ele pode gerar a probabilidade da observação pertencer à classe 1 e então pode-se utilizar um *threshold* para decidir a qual classe atribuí-la.

Algumas observações importantes sobre termos utilizados neste trabalho precisam ficar claras. Primeiro, um algoritmo de *Machine Learning* é entendido como um procedimento para se encontrar $\hat{F}$ a partir do treinamento com um conjunto de dados, já um modelo se trata da própria $\hat{F}$. É importante também não confundir uma observação x com um conjunto de hiperparâmetros $\mathbf{x}$, que será discutido nas próximas seções. Contudo, na explicação dos algoritmos evolutivos x é tratado como um conjunto de hiperparâmetros para o texto não ficar muito sobrecarregado de notações.

2.2 OTIMIZAÇÃO E SURROGATE

Otimização, neste contexto, consiste em encontrar $\mathbf{x}^* \in \chi$, em que $\chi \subset \mathbb{R}^d$ (d é o número de hiperparâmetros) corresponde a todo o espaço de conjuntos de hiperparâmetros $\mathbf{x}$ considerado, que maximiza uma função objetivo (chamada de função original neste trabalho) desconhecida f, a qual pode ser definida como o desempenho de todos os modelos obtidos utilizando χ (SHAHRIARI et al., 2016). O desempenho do modelo para cada conjunto de hiperparâmetros é, idealmente, obtido utilizando-se *k-fold cross-validation*, o que implica em treinar k modelos diferentes e utilizar a média de suas performances como resultado final.

Como já mencionado, esse processo de avaliação de desempenho pode se tornar computacionalmente caro a depender da quantidade de dados e algoritmo utilizado. Uma solução para contornar isso é a utilização de *surrogates*, que utilizam um conjunto de pares $(\mathbf{x}, f(\mathbf{x}))$ obtido nesse processo lento para construir um modelo de regressão que se aproxime da função original f (EGGENSPERGER et al., 2015). A *surrogate* obtida $\hat{f}$, que tem um custo computacional bastante baixo para gerar uma saída, pode então ser utilizada para realizar futuras avaliações de conjuntos de hiperparâmetros (EGGENSPERGER et al., 2015).

2.3 QUASIRANDOM SEQUENCES

Quasirandom sequences são sequências geradas de maneira determinística mas que possuem algumas propriedades de sequências aleatórias e podem ser mais vantajosas em determinadas aplicações (COOK, 2009). Seu objetivo é ocupar da melhor maneira possível o espaço de busca com a sequência gerada. No contexto de criação de *surrogates* deste trabalho, a sequência gerada seria um conjunto com todas as combinações de hiperparâmetros que seriam utilizadas, junto de suas avaliações de desempenho, para gerar o modelo. Uma *surrogate* gerada dessa maneira teria uma maior capacidade de generalização do que se fosse gerada com sequências aleatórias. Existem vários algoritmos para gerar essas sequências, neste trabalho será utilizado o de Sobol, exemplificado na Figura 1.

2.4 GRID SEARCH E RANDOM SEARCH

Grid Search é o método automático de otimização mais simples e foi por muito tempo o principal também, até ser demonstrado que *Random Search* é em geral mais eficiente (BERGSTRA; BENGIO, 2012). Seu funcionamento consiste em escolher um conjunto com diferentes valores para cada hiperparâmetro, realizando depois a avaliação dos modelos obtidos

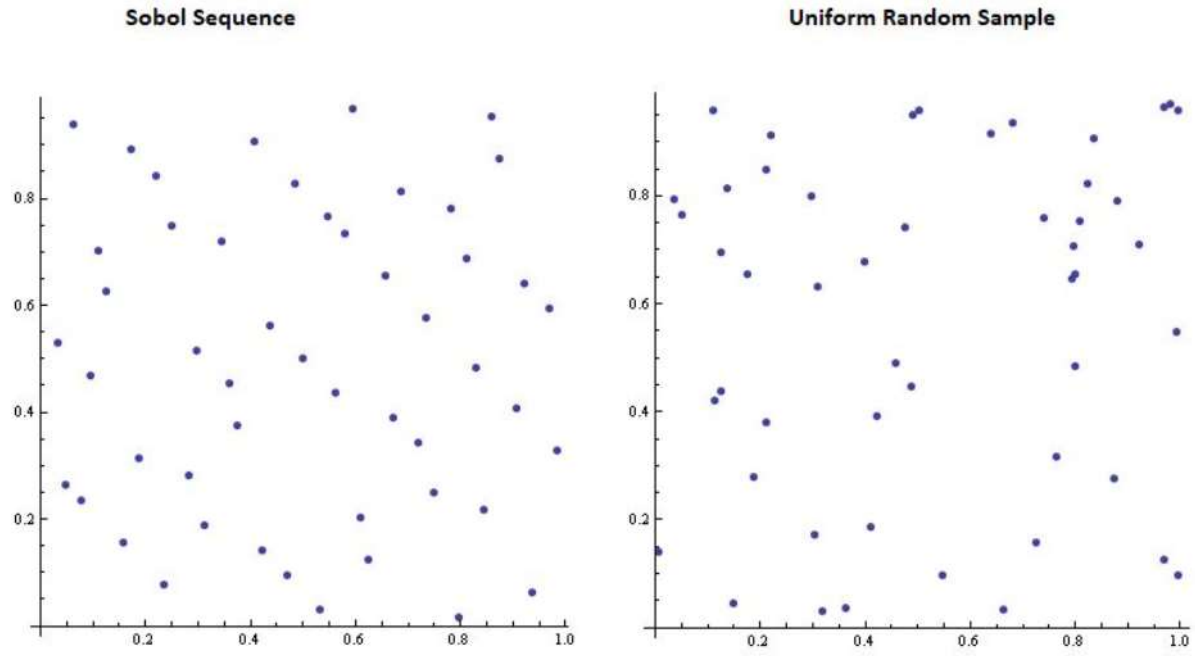

Figura 1: Sequências de 50 pontos bidimensionais gerados através do algoritmo de Sobol (esquerda) e de uma distribuição uniforme (direita). É possível notar que há espaços pouco ocupados e outros com pontos bastante próximos na sequência aleatória uniforme.

Fonte: (COOK, 2009)

a partir de todas combinações entre eles e selecionando o melhor resultado (SHEVCHUK, 2016). À medida que o número de hiperparâmetros aumenta, esse método pode ser tornar completamente inviável. Por exemplo, em um caso com 10 hiperparâmetros e 5 valores para cada o número de avaliações necessárias seria 9765625 (5^{10}). Considerando essa limitação, *Grid Search* não será utilizado neste trabalho como comparativo.

Random Search é também conceitualmente simples, consistindo em gerar aleatoriamente conjuntos independentes de hiperparâmetros atráves de uma distribuição de probabilidade uniforme (BROWNLEE, 2015). Dessa forma, é possível controlar quantos conjuntos de hiperparâmetros serão avaliados, sendo que quanto maior o número maior a chance de se obter um bom resultado. Bergstra e Bengio (2012) mostra que *Random Search* é o algoritmo base com o qual outros que utilizam informações de avaliações passadas devem ser comparados, e assim será feito neste trabalho.

2.5 BAYESIAN OPTIMIZATION

Bayesian Optimization possui dois principais componentes: um modelo probabilístico de regressão utilizado para construir uma *surrogate* e uma função de aquisição utilizada para

decidir o próximo **x** a ser avaliado por f (FRAZIER, 2018).

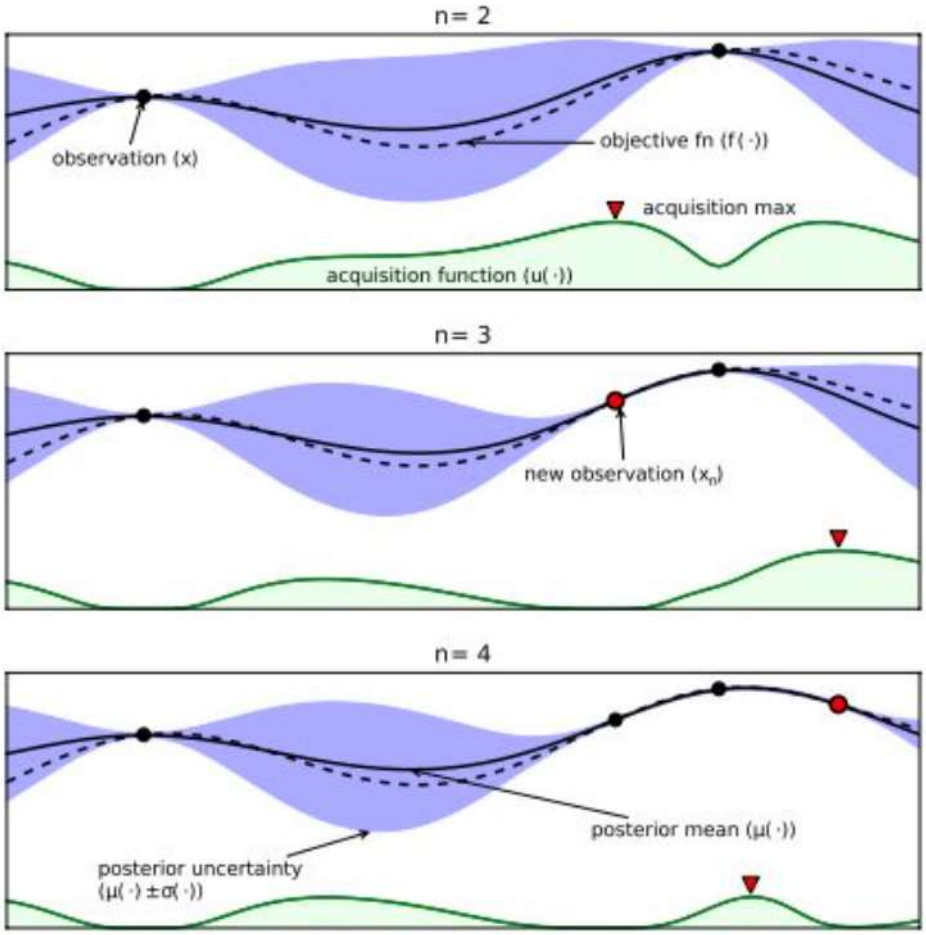

Figura 2: Ilustração de 3 iterações do algoritmo *Bayesian Optimization*. Nos gráficos, *objective fn* representa a função objetivo f, que na verdade é desconhecida, e *posterior* representa o modelo probabilístico de regressão $\hat{f}$.

Fonte: (SHAHRIARI et al., 2016)

Primeiro, a *surrogate* é inicializada utilizando-se um pequeno subconjunto de χ e seus valores de f. Seguindo essa inicialização, novos valores de **x** são selecionados através da otimização da função de aquisição, que tem por objetivo, utilizando as informações de $\hat{f}$, realizar um equilíbrio entre explorar novas regiões e escolher regiões que sabidamente são promissoras (DEWANCKER et al., 2015). A cada iteração, o novo par $(\mathbf{x}, f(\mathbf{x}))$ é utilizado para atualizar a *surrogate* e esse processo se repete até que um valor limite **N** de número de avaliações em f seja alcançado (DEWANCKER et al., 2015). No final é retornado o ponto que possui o maior valor de f ou o maior valor médio de $\hat{f}$ (FRAZIER, 2018). Esse processo é exemplificado na Figura 2.

Os 3 métodos de regressão mais comumente utilizados para construir a *surrogate*

são Gaussian Process, Random Forest e Tree Parzen Estimator, porém qualquer um pode ser utilizado desde que produza uma distribuição de propabilidade sobre sua saída (DEWANCKER et al., 2015). A função de aquisição mais comum é *expected improvement*.

2.6 EVOLUÇÃO DIFERENCIAL

Algoritmos evolutivos possuem uma inspiração em mecanismos de evolução biológica e têm como foco resolver problemas de otimização. São de especial interesse em problemas de otimização global de funções desconhecidas, nos quais não é possível aplicar métodos analíticos (VESTERSTROM; THOMSEN, 2004). Um exemplo dessa classe de problemas é justamente a otimização de hiperparâmetros em *Machine Learning*.

Evolução Diferencial foi um dos algoritmos evolutivos escolhidos por demonstrar, de acordo com Vesterstrom e Thomsen (2004), uma maior robustez e performance se comparado a algoritmos genéticos e *Particle Swarm Optimization* na maioria dos problemas, e também por possuir menos parâmetros.

As principais etapas do algoritmo DE são (SU, 2008):

1. Inicialização

Uma população de P indivíduos $\mathbf{X}_G = \{x_{1,G}, x_{2,G}, ..., x_{P,G}\}$ é criada aleatoriamente considerando o espaço χ, pois no contexto deste trabalho um indivíduo corresponde a um conjunto de d hiperparâmetros.

2. Mutação

O vetor modificado é obtido por

$$v_{i,G+1} = x_{r1,G} + F.(x_{r2,G} - x_{r3,G}) \tag{1}$$

em que $r_1, r_2, r_3 \in \{1, 2, ..., P\}$ são escolhidos aleatoriamente e o fator de mutação $F \in [0, 2]$.

3. Cruzamento

O vetor modificado é combinado com um indivíduo selecionado aleatoriamente da população (o vetor alvo), resultando em um vetor experimental

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{se } rand_j \leq CR \text{ or } j = I_{rand} \\ x_{ji,G} & \text{se } rand_j > CR \text{ and } j \neq I_{rand} \end{cases} \tag{2}$$

em que $j = 1, 2, ..., d$; $rand_j$ é um número gerado aleatoriamente entre 0 e 1; CR é a

constante de cruzamento $\in$ [0, 1] e I_{rand} é um inteiro aleatório em $\{1,2,...,d\}$.

4. Seleção

O vetor alvo é comparado com o vetor experimental e o com melhor avaliação é selecionado para próxima geração

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{se } f(u_{i,G+1}) \geq f(x_{i,G}) \\ x_{i,G} & \textit{caso contrário} \end{cases} \tag{3}$$

Esse processo (etapas 2, 3 e 4) é repetido até que uma condição de parada seja satisfeita, a qual pode ser um número máximo de iterações por exemplo.

2.7 PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization se baseia no comportamento de um bando de pássaros à procura de um alvo, em que são relevantes tanto a experiência individual quanto a dos outros membros do grupo. O PSO foi utilizado no trabalho pela necessidade de se obter uma análise confiável do desempenho da abordagem proposta, tendo em vista que resultados ruins poderiam ser causados por uma inadequação do DE ao problema.

O algoritmo funciona da seguinte maneira (WANG; LIU, 2016):

Uma população de M partículas de d dimensões (número de hiperparâmetros neste contexto) é gerada aleatoriamente, sendo que cada partícula possui uma posição $x_i = \{x_{i,1}, x_{i,2}, ..., x_{i,d}\}$ e uma velocidade $v_i = \{v_{i,1}, v_{i,2}, ..., v_{i,d}\}$ para indicar seu estado atual, em que a posição representa uma possível solução para o problema de otimização. A posição e a velocidade da partícula são atualizadas de acordo com a melhor posição encontrada pela partícula até o momento $p_i = \{p_{i,1}, p_{i,2}, ..., p_{i,d}\}$ e a melhor encontrada pela população inteira $p_g = \{p_{g,1}, p_{g,2}, ..., p_{g,d}\}$ através das seguintes equações:

$$v_{i,d}(t+1) = \omega v_{i,d}(t) + c_1 r_1 (p_{i,d}(t) - x_{i,d}(t)) + c_2 r_2 (p_{g,d}(t) - x_{i,d}(t)) \tag{4}$$

$$x_{i,d}(t+1) = x_{i,d}(t) + v_{i,d}(t+1) \tag{5}$$

em que c_1 e c_2 são os parâmetros cognitivo e social, respectivamente, r_1 e r_2 são números aleatórios no intervalo [0, 1] e ω é ponderação de inércia. As partículas têm seus

estados atualizados até que uma condição de parada do algoritmo seja satisfeita, como um número máximo de iterações por exemplo.

2.8 GRADIENT BOOSTING MACHINES

Gradient Boosting é uma abordagem geral que pode ser aplicada a diversos algoritmos de *Machine Learning*, mas em geral são utilizadas árvores de decisão. Sua estratégia, assim como no Random Forest, consiste em combinar o resultado de diversas árvores de decisão para gerar uma saída mais robusta. Porém se diferencia pelo fato de as árvores serem construídas sequencialmente, cada uma utilizando informações da anterior (JAMES et al., 2013). De acordo com Bruce e Bruce (2017), enquanto Random Forest não é muito dependente de otimização de hiperparâmetros, Gradient Boosting toma o caminho contrário, sendo o primeiro comparado a um Honda (confiável e estável) e o segundo a um Porsche (poderoso mas exige mais cuidado).

Em sua execução, dado o modelo atual, que corresponde ao conjunto de árvores até o momento, a próxima árvore é treinada utilizando-se os resíduos atuais, que são as diferenças entre as saídas reais e as estimadas pelo modelo. Depois, essa nova árvore é adicionada ao modelo e os resíduos atualizados, fazendo com que qualidade de estimação do modelo melhore lentamente durante o processo (JAMES et al., 2013). O Algoritmo 1 resume esse procedimento no caso de uma regressão, sendo que a única diferença para classificação seria a forma de calcular os resíduos.

Algoritmo 1: Gradient Boosting Trees para regressão

Entrada: Conjunto de dados para treino (X, Y)
início
  Inicializa $\hat{F}(x) = 0$ e $r_i = y_i$ para todo i nos dados de treino
  para $b = 1, 2, ..., B$ *(número total de árvores)* **faça**
    Treine a árvore $\hat{F}^b$ com os dados (X, R)
    Atualiza $\hat{F}$ adicionando uma versão encolhida da nova árvore

$$\hat{F}(x) \leftarrow \hat{F}(x) + \lambda \hat{F}^b(x)$$

    Atualiza os resíduos

$$r_i \leftarrow r_i - \lambda \hat{F}^b(x_i)$$

  fim
fim
Saída: Modelo final

$$\hat{F}(x) = \sum_{b=1}^{B} \lambda \hat{F}^b(x)$$

2.9 MULTIVARIATE ADAPTIVE REGRESSION SPLINES

MARS é um algoritmo de regressão adaptativo, capaz de modelar não linearidades e interações de maneira automática. Sua implementação é consideravelmente complexa e seu detalhamento foge do escopo deste trabalho, portanto será feito apenas um resumo com base em Hastie et al. (2009).

MARS utiliza funções base no formato $(x-t)_+$ e $(t-x)_+$, em que

$$(x-t)_+ = \begin{cases} x-t, & \text{se } x > t \\ 0, & \textit{caso contrário} \end{cases} \quad e \quad (t-x)_+ = \begin{cases} t-x, & \text{se } x < t \\ 0, & \textit{caso contrário} \end{cases} \tag{6}$$

Cada função é linear em trecho com um nó em t. A ideia é formar uma coleção de funções base com nós em cada elemento x_{ij} do conjunto de dados de treinamento:

$$C = \{(X_j - t)_+, (t - X_j)_+\}, \quad t \in \{x_{1j}, x_{2j}, ..., x_{Nj}\} \quad e \quad j = 1, 2, ..., d \tag{7}$$

Na construção do modelo há duas etapas: uma direta e uma inversa. Na direta inicia-se um conjunto M do modelo contendo a função $h_0(X) = 1$. Em cada estágio seguinte uma nova função é adicionado ao modelo, sendo ela o produto de alguma função h_m em M com algum par em C que causa a maior diminuição do erro de treinamento. Esse processo se repete até que um número máximo de termos em M seja alcançado.

Na etapa inversa, sucessivamente o termo cuja remoção causa o menor aumento do erro de treinamento é retirado, restando apenas λ termos no conjunto M. O valor ótimo de λ é estimado através de *generalized cross-validation*. O modelo obtido no final do processo tem a seguinte forma

$$f(X) = \beta_0 + \sum_{m=1}^{\lambda} \beta_m h_m(X), \tag{8}$$

sendo que os parâmetros podem ser estimados da mesma maneira que em uma regressão linear.

2.10 TRABALHOS RELACIONADOS

Como este trabalho propõe uma nova abordagem, não foram encontrados outros que tratam especificamente da mesma ideia, apenas alguns que propõem utilizar DE em conjunto com uma *surrogate* auxiliar e outros que utilizam DE ou PSO para otimização de hiperparâmetros.

Su (2008) propôs um novo procedimento de otimização utilizando evolução diferencial e uma *surrogate* construída com Gaussian Process, assemelhando-se a *Bayesian Optimization*. A *surrogate* é utilizada para diminuir a quantidade de avaliações na função original, sendo iterativamente melhorada com a aquisição de novos pontos obtidos pelo algoritmo DE. No trabalho ele compara o método a *Particle Swarm Optimization* e a um DE tradicional, utilizando funções de *benchmark* e uma aplicação real de identificação de parâmetros de rochas. No final constata que o método proposto é bem mais eficiente, necessitando de um número menor de avaliações da função original.

Zhang et al. (2011) mostra a utilização de um DE tradicional na otimização dos hiperparâmetros de Support Vector Machines com RBF *kernel*. O método é comparado a *Particle Swarm Optimization* e *Grid Search*, utilizando 4 conjuntos de dados. Os resultados indicam que a abordagem consegue um melhor resultado que as demais, porém a execução foi mais lenta comparada à do PSO.

Krempser et al. (2012) propõe a utilização do DE com uma *surrogate*, gerada através de uma variação do algoritmo *K-nearest neighbors*, para otimização do tamanho de estruturas. A *surrogate* é construída a partir de uma população inicial avaliada na função original, os novos candidatos a solução são avaliados pela *surrogate* e o melhor deles pela função original, o qual substitui seu progenitor caso possua uma avaliação melhor. Cada ponto avaliado pela função original durante o processo é adicionado ao modelo da *surrogate*, sendo ela então melhorada iterativamente. O trabalho compara o método proposto com algumas variações do DE usado de forma isolada e chega ao resultado de que ele se sai melhor na maioria dos problemas testados.

3 DESENVOLVIMENTO

Neste capítulo serão apresentados as tecnologias, métodos e configurações de algoritmos utilizados no desenvolvimento do projeto.

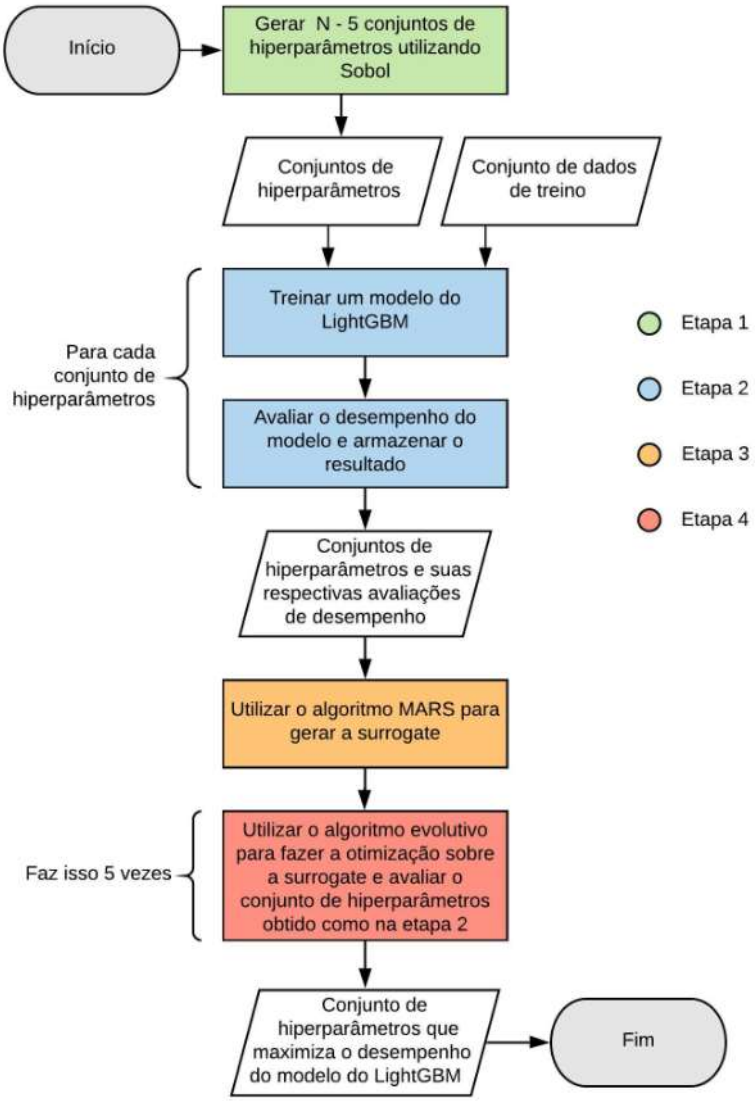

Figura 3: Fluxograma da abordagem proposta.

Fonte: Autoria Própria

3.1 A ABORDAGEM PROPOSTA

Para realizar a comparação entre os métodos de otimização de forma justa é necessário fixar um número máximo de avaliações de desempenho do modelo para todos. Dessa forma é possível assumir que todos eles tenham um tempo de execução muito próximo, considerando que a maior carga computacional está no processo de avaliação de desempenho. E de fato o tempo de execução de outros processos da abordagem proposta, como a geração da *surrogate* e a otimização com algoritmo evolutivo, é irrelevante quando comparado ao tempo total. Com base nisso, surgiu a ideia de utilizar N - Z, em que N é o número máximo de avaliações de desempenho, conjuntos de hiperparâmetros para construir a *surrogate*. Assim, poderiam ser executadas Z otimizações com o algoritmo evolutivo em uma única execução da abordagem proposta, utilizando as Z avaliações de desempenho restantes para validar o desempenho de cada otimização e retornar o conjunto de hiperparâmetros que se sair melhor. Com isso seria possível obter uma maior consistência no resultado da otimização, o qual varia de uma execução para outra por fatores aleatórios.

O fluxograma da Figura 3 resume todas as etapas da otimização com a abordagem proposta. Na primeira etapa são gerados os conjuntos de hiperparâmetros utilizando o método de Sobol, sendo que o valor de Z escolhido foi 5 e o de N será discutido posteriormente neste capítulo. Na segunda é realizada a avaliação de desempenho para cada conjunto de hiperparâmetros, gerando um conjunto de pares (hiperparâmetros, desempenho). Tal conjunto é utilizado na terceira etapa, em que é gerada a *surrogate* através de regressão com o algoritmo MARS. Na última etapa acontece a otimização com o algoritmo evolutivo sobre a *surrogate*, processo que é repetido 5 vezes, sendo armazenados os conjuntos de hiperparâmetros obtidos junto de suas avaliações na função original. O conjunto que tiver o melhor desempenho é o resultado da otimização.

3.2 TECNOLOGIAS E FERRAMENTAS

3.2.1 PYTHON E JUPYTER NOTEBOOK

Python é uma linguagem de programação de código aberto que suporta diversos paradigmas como orientação a objetos, programação estruturada, funcional, entre outros. Devido à sua simplicidade e grande quantidade de bibliotecas científicas disponíveis, ela se tornou uma das mais utilizadas nas áreas de *Data Science* e *Machine Learning* (PROGRAMS, 2018).

Alguns dos módulos utilizados na implementação foram: NumPy, para operações com *arrays*; Matplotlib e Seaborn, para visualização de gráficos; Pandas, para manipulação e análise de dados e scikit-learn, que implementa utilitários e algoritmos de *Machine Learning*. Para gerar a sequência de Sobol foi utilizado o módulo Chaospy, como implementação do algoritmo MARS o módulo Py-Earth, como implementação do DE o módulo SciPy, como implementação do PSO o módulo PySwarms e para realizar a otimização com Random Search e Bayesian Optimization foi utilizado o módulo Hyperopt.

Jupyter Notebook é uma aplicação web de código aberto que suporta linguagens como Python, R e Julia. O ambiente possibilita a criação de documentos que permitem a escrita de código, bem como sua execução e saída, texto, equações, imagens, vídeos e muito mais (JUPYTER, 2019).

3.2.2 LIGHTGBM

LightGBM é um *framework* para Gradient Boosting Machines baseado em árvores de decisão, o qual possui APIs para Python, R e outras linguagens. Foi proposto com o objetivo de ser eficiente e distribuído, permitindo utilizar uma GPU para realizar o processamento (LIGHTGBM, 2019). É considerado uma das implementações com maior velocidade de execução para GBMs e amplamente utilizado em competições de *Machine Learning*.

Ele possui diversos hiperparâmetros que precisam ser otimizados, porém neste trabalho somente 9 serão focados, por possuírem uma maior influência no controle de *overfitting*. A tabela abaixo os apresenta, juntamente com os limites inferior e superior do intervalo considerado de cada um. Os 5 primeiros estão no domínio dos números reais e os 4 últimos inteiros.

Tabela 1: Hiperparâmetros do LightGBM escolhidos e os intervalos considerados.

Hiperparâmetro	Limite inferior	Limite superior
learning rate	0.0005	0.5
feature fraction	0.1	1
bagging fraction	0.3	1
L1 regularization	0.001	1
L2 regularization	0.001	1
max number of bins	15	255
max number of leaves	2	50
bagging frequence	1	10
minimal number of data in one leaf	1	200

Fonte: Autoria própria

3.3 LIMITAÇÃO DO HARDWARE UTILIZADO E SUAS CONSEQUÊNCIAS

Todas as etapas do projeto foram executadas em um notebook com processador Intel Core i5-7200U, que possui 2 núcleos e 4 threads, e 10GB de memória RAM. Sendo assim, algumas escolhas tiveram que ser feitas de modo que a execução das etapas terminassem em um intervalo de tempo razoável, considerando que o treinamento, e consequentemente a otimização de hiperparâmetros, de GBMs possui um alto custo computacional se comparado ao de outros algoritmos.

A primeira consequência foi a escolha dos conjuntos de dados: eles deveriam ter um número de instâncias não superior a 2000, um número bem pequeno para os padrões atuais, sendo que o recomendado para utilização de GBMs é acima de 10000, abaixo disso há uma forte tendência à ocorrência de *overfitting*. Outra consequência foi a escolha do número máximo de avaliações de desempenho do modelo para o processo de otimização: apenas 105, o que pode ter prejudicado a qualidade das *surrogates* geradas na abordagem proposta.

3.4 CONJUNTOS DE DADOS UTILIZADOS

Os 3 conjuntos de dados utilizados envolvem problemas de classificação binária e possuem apenas atributos numéricos. O primeiro foi retirado de uma competição de *Machine Learning* do Kaggle que consistia em identificar se determinado cliente do Santander iria fazer uma transação em um futuro próximo, sendo que os atributos são anônimos e foi utilizada uma amostra de 2000 instâncias do conjunto original de 200000. Os outros 2 foram retirados sem nenhuma modificação do repositório da Universidade da Califórnia de Irvine (UCI), o primeiro contendo atributos extraídos de imagens de fundo de olho para identificação de retinopatia diabética e o segundo contendo descrições moleculares para identificar se determinada substância química é biodegradável.

A tabela abaixo apresenta algumas características dos conjuntos de dados, sendo que eles são nomeados como Santander, Retinopatia e Biodegradável de acordo com a ordem no parágrafo anterior.

Tabela 2: Características dos conjuntos de dados utilizados.

Conjunto	Número de atributos	Tamanho	Tamanho da classe 1	Tamanho da classe 0
Santander	200	2000	1000	1000
Retinopatia	19	1151	611	540
Biodegradável	41	1055	356	699

Fonte: Autoria própria

3.5 METODOLOGIA DE AVALIAÇÃO DE DESEMPENHO DO MODELO

Avaliar o desempenho de um modelo de *Machine Learning* corresponde a dizer quão bem ele se sai em estimar a saída de observações não vistas por ele, ou seja, dados que não foram utilizados durante seu treinamento (JAMES et al., 2013). Existem várias técnicas para isso, sendo uma delas o *hold-out*, que consiste em dividir o conjunto de dados em 2, um para realizar o treinamento e outro para avaliação de desempenho.

Uma técnica mais robusta e utilizada, embora com maior custo computacional, é *k-fold cross validation*, que consegue diminuir a influência da maneira de como os dados são divididos sobre a avaliação. Nessa técnica, o conjunto de dados é dividido em k grupos de mesmo tamanho, sendo $k-1$ utilizados para realizar o treinamento e o último para avaliação. Esse processo é repetido k vezes até que todos os grupos sejam utilizados para avaliação uma única vez. O resultado do desempenho do modelo é então obtido calculando-se a média dos desempenhos individuais (JAMES et al., 2013). Neste trabalho foi utilizado *5-fold* como uma forma de se ter robustez sem aumentar muito o custo computacional.

A avaliação de desempenho também exige que uma métrica seja utilizada. Existem diversas alternativas para problemas de classificação binária, que é o caso tratado neste trabalho, sendo acurácia a mais comum. Essa métrica simplesmente calcula a porcentagem de observações classificadas corretamente. Contudo, ela nem sempre produz resultados confiáveis, como no caso de classes não balanceadas por exemplo. Suponha que um conjunto de dados possua 90% de suas observações pertencendo à classe 0 e o restante à classe 1. Um modelo que sempre classifica uma nova instância como classe 0 teria uma acurácia de 90%, mesmo sendo completamente ineficiente.

Em problemas de classificação binária em que a saída do modelo corresponde à probabilidade da observação pertencer à classe 1 pode-se utilizar a curva ROC (*Receiver Operating Characteristics*). Essa curva mostra o equilíbrio entre *true positive rate* (capacidade do modelo de classificar corretamente as observações da classe positiva) e *false positive rate* (proporção de observações da classe negativa erroneamente classificadas como positiva) para diferentes *thresholds* (JAMES et al., 2013), sendo ilustrada na Figura 4 . A partir dela pode-se obter a métrica AUC (Area under the ROC Curve), que tende a ser mais confiável que acurácia e portanto foi utilizada neste trabalho.

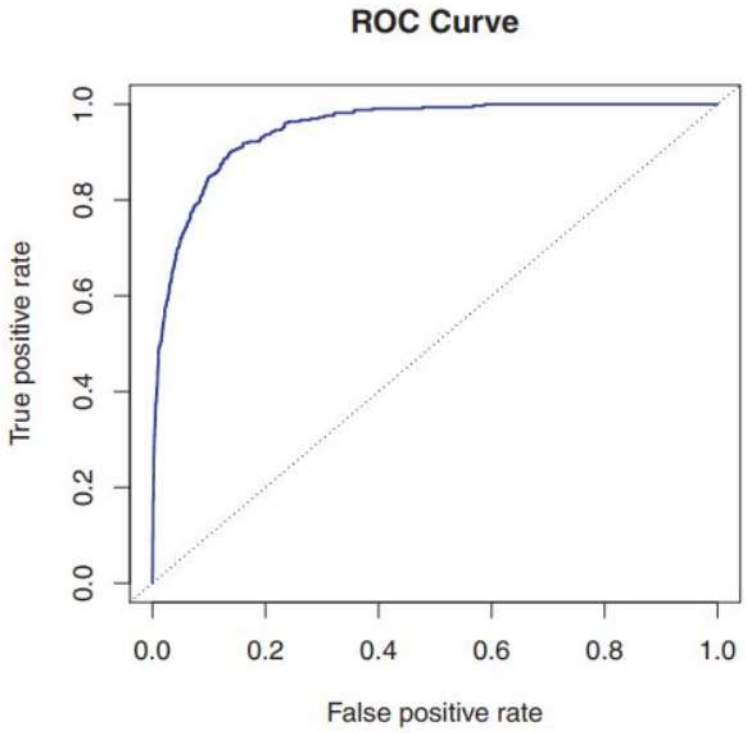

Figura 4: Ilustração da curva ROC.
Fonte: (JAMES et al., 2013)

3.6 CONFIGURAÇÕES DOS ALGORITMOS UTILIZADOS

3.6.1 MARS

MARS foi o único algoritmo de regressão utilizado neste trabalho por dois motivos: a intenção do projeto é criar uma nova abordagem para otimização de hiperparâmetros, se para isso fosse também necessário otimizar os hiperparâmetros do algoritmo de regressão utilizado para gerar a *surrogate* criar-se-ia um ciclo indesejado. No entanto, escolhendo manualmente apenas um hiperparâmetro do MARS já é possível obter resultados satisfatórios. E também, foram testados outros algoritmos, como Redes Neurais, Gaussian Process e GAM (Generalized Additive Model), porém nenhum deles conseguiu bons resultados, seja por falta de dados para o treinamento ou pela dificuldade de otimização dos hiperparâmetros de forma manual.

O hiperparâmetro do MARS que foi escolhido manualmente controla o grau máximo de interação que pode ocorrer no modelo, ou seja, o número máximo de variáveis que podem participar de um produto. Fixando esse número em 2 e mantendo todos os outros hiperparâmetros com os valores padrão do módulo Py-Earth, foi possível obter *surrogates* de qualidade satisfatória (RMSE menor que 0.01), independentemente do conjunto de dados do

problema.

3.6.2 PSO E DE

Apesar de haver uma mistura de números reais e inteiros no conjunto de hiperparâmetros, nenhuma estratégia específica foi utilizada para lidar com isso durante a otimização. Como a *surrogate* trata todos os atributos como números reais, durante a otimização sobre ela com o PSO e DE os conjuntos de hiperparâmetros a serem avaliados por ela foram os mesmos gerados através das operações dos 2 algoritmos, sem nenhuma modificação. Ou seja, durante a otimização todos os hiperparâmetros foram tratados como reais. Já na fase de validação, quando o conjunto de hiperparâmetros resultante da otimização sobre a surrogate é avaliado na função original, foi preciso arredondar para o número inteiro mais próximo os hiperparâmetros que são inteiros.

No PSO 5 parâmetros precisaram ser definidos: parâmetros cognitivo e social, ponderação de inércia (Equação 4), quantidade de partículas da população e número máximo de iterações. Os valores utilizados foram, respectivamente, 0.5, 0.3, 0.9, 180 e 500. Os 3 primeiros constavam em um exemplo da documentação do módulo PySwarms e resultaram em melhores resultados do que outras combinações, considerando todos os conjuntos de dados. Já os outros 2 foram escolhidos de forma manual após a realização de diversos testes.

A estratégia utilizada para o DE foi a clássica ou DE/rand/1/bin, cujo funcionamento é o mesmo do algoritmo apresentado no capítulo anterior. Sobre os parâmetros, 4 precisaram ser escolhidos: o tamanho da população, fator de mutação (Equação 1), constante de cruzamento (Equação 2) e um número máximo de gerações. O primeiro e os 2 últimos foram otimizados manualmente, considerando o desempenho nos 3 conjuntos de dados, obtendo-se os respectivos valores: 90, 0.9 e 4. Já o fator de mutação foi mantido como o padrão da implementação do módulo Scipy, o qual escolhe um número aleatório entre 0.5 e 1 para ser usado em cada geração.

4 RESULTADOS

Neste capítulo serão apresentados e discutidos os resultados obtidos utilizando os 3 métodos de otimização (o proposto, Random Search e Bayesian Optimization) nos 3 conjuntos de dados, sendo que o método proposto possui duas variantes, uma com PSO e outra com DE. Cada tabela apresentará os resultados de um conjunto de dados.

Para cada método de otimização foram realizadas 20 execuções para cada um dos conjuntos de dados, sendo o resultado de cada execução a métrica AUC do conjunto de hiperparâmetros resultante da otimização. A partir dessas 20 execuções foram calculadas as seguintes medidas descritivas: média, mediana, valor máximo, valor mínimo e desvio padrão. Dessa forma, é possível definir não apenas o método que obteve o melhor resultado mas também o que foi mais consistente. Um resultado melhor significa maiores média, mediana e valor máximo, e um resultado mais consistente significa maior valor mínimo e menor desvio padrão.

4.1 CONJUNTO DE DADOS SANTANDER

Neste conjunto de dados, *Bayesian Optimization* conseguiu os melhores resultados, considerando média, mediana e valor máximo, seguido pelo método proposto com PSO. Este, porém, foi o mais consistente, apresentando o menor desvio padrão e maior valor mínimo. Já a variante com DE foi o pior método em todos os quesitos.

Tabela 3: Resultados no conjunto de dados Santander.

Método	Média	Mediana	Máximo	Mínimo	Desvio Padrão
Bayesian Optimization	0.8571	0.8567	0.8609	0.8513	0.0023
Random Search	0.8543	0.8540	0.8582	0.8511	0.0024
Método proposto (DE)	0.8524	0.8532	0.8562	0.8451	0.0029
Método proposto (PSO)	0.8553	0.8555	0.8592	0.8524	0.0016

Fonte: Autoria própria

4.2 CONJUNTO DE DADOS RETINOPATIA

Neste conjunto de dados, *Bayesian Optimization* conseguiu os melhores resultados, considerando média, mediana e valor máximo, seguido pelo método proposto com DE, embora este tenha sido o pior em termos de consistência. Já a variante com PSO, apesar de ter a pior média e valor máximo, conseguiu novamente ser o mais consistente, considerando o valor mínimo e desvio padrão.

Tabela 4: Resultados no conjunto de dados Retinopatia.

Método	Média	Mediana	Máximo	Mínimo	Desvio Padrão
Bayesian Optimization	0.7912	0.7885	0.8038	0.7793	0.0063
Random Search	0.7841	0.7840	0.7968	0.7789	0.0041
Método proposto (DE)	0.7861	0.7851	0.7997	0.7774	0.0065
Método proposto (PSO)	0.7839	0.7836	0.7879	0.7820	0.0014

Fonte: Autoria própria

4.3 CONJUNTO DE DADOS BIODEGRADÁVEL

Neste conjunto de dados, os resultados foram mais próximos. *Bayesian Optimization* foi novamente o melhor, apresentando maiores média, mediana e valor máximo. O método proposto com PSO foi o segundo melhor, embora tenha obtido um valor máximo ligeiramente menor do que o com DE, e foi pela terceira vez o mais consistente. O método com DE teve um resultado bem semelhante ao do *Random Search*.

Tabela 5: Resultados no conjunto de dados Biodegradável.

Método	Média	Mediana	Máximo	Mínimo	Desvio Padrão
Bayesian Optimization	0.9327	0.9331	0.9352	0.9297	0.0012
Random Search	0.9317	0.9318	0.9329	0.9301	0.0008
Método proposto (DE)	0.9313	0.9312	0.9333	0.9301	0.0008
Método proposto (PSO)	0.9321	0.9320	0.9331	0.9315	0.0004

Fonte: Autoria própria

4.4 SÍNTESE

Bayesian Optimization conseguiu os melhores resultados em todos os conjuntos de dados, porém foi o menos consistente em um deles. A variante com PSO do método proposto foi o segundo melhor em dois conjuntos e foi o mais consistente em todos. Já a variante com

DE foi o menos consistente em dois conjuntos e o segundo melhor em um. E por fim, o *Random Search* curiosamente não foi o menos consistente em nenhum dos conjuntos, embora também não tenha sido o melhor ou segundo melhor.

5 CONCLUSÃO

Este trabalho apresentou uma nova proposta para otimização de hiperparâmetros em algoritmos de *Machine Learning*. Nela, a ideia é utilizar uma *surrogate* para substituir a função original de avaliação de desempenho do modelo, a qual é gerada com um algoritmo de regressão, o MARS neste trabalho, e pontos obtidos através do método de Sobol. Em seguida, é possível utilizar um algoritmo evolutivo (PSO ou DE neste caso) para realizar a otimização sobre a *surrogate*.

O método proposto utilizando o PSO, embora não tenha obtido os melhores resultados em nenhum dos conjuntos de dados, foi o segundo melhor em dois deles e foi o mais consistente em todos. Isso foi na verdade uma surpresa, pois a ideia era utilizá-lo apenas como base de comparação com o DE, tendo em vista que este era foco do trabalho inicialmente. Os resultados com o DE foram bem abaixo do esperado, considerando que, mesmo o método proposto tendo sido pensado de forma a aumentar a consistência dos resultados, ele foi o menos consistente em dois conjuntos.

De qualquer forma, esperava-se uma consistência ainda maior, mesmo do PSO, já que são realizadas 5 otimizações com o algoritmo evolutivo em uma mesma execução do método proposto. Isso sugere que esse número deve ser maior, mesmo que para isso menos conjuntos de hiperparâmetros sejam utilizados na construção da *surrogate*.

Na maioria dos casos, *Bayesian Optimization* ainda é a melhor opção para otimização de hiperparâmetros, principalmente em situações em que poucos modelos precisam ser otimizados ou o objetivo é o melhor resultado possível, como por exemplo em projetos de pesquisa. Porém pode haver situações em que muitos modelos precisem ser otimizados em um intervalo de tempo limitado, então talvez seja preferível um método mais consistente, que garante que o pior resultado obtido seja ainda satisfatório. Nesta situação o método proposto com a utilização do PSO pode ser uma alternativa a ser considerada.

5.1 LIMITAÇÕES

As principais limitações do trabalho estão relacionadas à própria limitação do hardware utilizado, o que fez com que algumas escolhas fossem feitas de modo que as execuções terminassem em um intervalo de tempo razoável. Primeiro, os conjuntos de dados tinham um número muito pequeno de instâncias para o que é recomendado na utilização de GBMs, o que pode ter diminuído a capacidade de otimização devido à alta tendência a *overfitting* dos modelos. Segundo, o número máximo de avaliações de desempenho do modelo sendo definido como apenas 105 provavelmente prejudicou a qualidade das *surrogates* obtidas, e consequentemente o resultado das otimizações. E por fim, 20 execuções de cada método por conjunto de dados está abaixo do ideal para obtenção de resultados mais confiáveis do ponto de vista estatístico, então algumas diferenças encontradas podem estar na verdade dentro da margem de erro.

5.2 TRABALHOS FUTUROS

Há várias possibilidades de melhoria da abordagem proposta. Podem ser utilizados outros algoritmos de regressão para a construção da *surrogate*, tendo em vista que apenas uma minúscula parcela de todos os existentes foi testada neste trabalho. Outros algoritmos de otimização podem ser testados, não se prendendo apenas aos evolutivos, ou ainda o PSO ou DE podem ser configurados de outra forma. Também pode ser utilizada uma estratégia melhor para lidar com a mistura de números e reais e inteiros, e não apenas tratar todos como reais e arredondar quando necessário.

Outra questão é que, embora o foco do trabalho tenha sido especificamente em um *framework* de GBMs aplicado a classificação binária, o método proposto, desde que com as devidas adaptações, serve para qualquer algoritmo, mesmo que seja classificação multinomial, regressão ou até mesmo aprendizado não supervisionado. O único requisito é que seja possível avaliar o desempenho do modelo obtido com um determinado conjunto de hiperparâmetros através de uma métrica numérica.

E também, é possível explorar o método proposto, com algumas adaptações, em um tipo de problema específico em que se tem um conjunto de combinações de variáveis junto de suas métricas, porém não é possível realizar novas avaliações na função original. Dessa forma, é possível gerar uma *surrogate* com esse conjunto e realizar a otimização sobre ela. *Bayesian Optimization* e *Random Search* não podem ser utilizados neste contexto por dependerem de novas avaliações na função original.

REFERÊNCIAS

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

BROCHU, E.; CORA, V. M.; FREITAS, N. de. **A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning**. 2010.

BROWNLEE, J. **Random Search**. 2015. Disponível em: <http://www.cleveralgorithms.com/nature-inspired/stochastic/random_search.html>. Acesso em: 14 de março de 2020.

BRUCE, P.; BRUCE, A. **Practical Statistics for Data Scientists**. Sebastopol, CA: O'Reilly, 2017.

COOK, J. D. **Quasi-random sequences in art and integration**. 2009. Disponível em: <https://www.johndcook.com/blog/2009/03/16/quasi-random-sequences-in-art-and-integration/>. Acesso em: 15 de março de 2020.

DEWANCKER, I.; MCCOURT, M.; CLARK, S. **Bayesian Optimization Primer**. 2015.

EGGENSPERGER, K. et al. Efficient benchmarking of hyperparameter optimizers via surrogates. **AAAI Press**, p. 1114–1120, 2015.

FRAZIER, P. I. **A Tutorial on Bayesian Optimization**. 2018.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. 2. ed. New York: Springer, 2009.

JAMES, G. et al. **An Introduction to Statistical Learning**. 1. ed. New York: Springer, 2013.

JOHNSON, A. **Common Problems in Hyperparameter Optimization**. 2017. Disponível em: <https://sigopt.com/blog/common-problems-in-hyperparameter-optimization/>. Acesso em: 12 de maio de 2019.

JUPYTER. **Jupyter home page**. 2019. Disponível em: <https://jupyter.org/index.html>. Acesso em: 16 de maio de 2019.

KREMPSER, E. et al. Differential evolution assisted by surrogate models for structural optimization problems. **Proceedings of the Eighth International Conference on Engineering Computational Technology**, v. 49, 2012.

LIGHTGBM. **LightGBM's documentation**. 2019. Disponível em: <https://lightgbm.readthedocs.io/en/latest/>. Acesso em: 12 de maio de 2019.

MURPHY, K. P. **Machine Learning: A Probabilistic Perspective**. 1. ed. Cambridge, Massachusetts: The MIT Press, 2012.

PROGRAMS, D. S. G. **Understanding How Python is Used in Data Science**. 2018. Disponível em: <https://www.datasciencegraduateprograms.com/about-us/>. Acesso em: 16 de maio de 2019.

SAS. **Machine Learning: o que é e qual sua importância.** 2019. Disponível em: <https://www.sas.com/pt_br/insights/analytics/machine-learning.html>. Acesso em: 11 de maio de 2019.

SHAHRIARI, B. et al. Taking the human out of the loop: A review of bayesian optimization. **Proceedings of the IEEE**, v. 104, n. 1, p. 148–175, 2016.

SHEVCHUK, Y. **Hyperparameter optimization for Neural Networks**. 2016. Disponível em: <http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.htmlgrid-search>. Acesso em: 13 de maio de 2019.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. **Practical Bayesian Optimization of Machine Learning Algorithms**. 2012.

SU, G. Gaussian process assisted differential evolution algorithm for computationally expensive optimization problems. **IEEE PACIIA**, p. 272–276, 2008.

VESTERSTROM, J.; THOMSEN, R. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. **Proceedings of the 2004 Congress on Evolutionary Computation**, v. 2, p. 1980–1987, 2004.

WANG, C. F.; LIU, K. A novel particle swarm optimization algorithm for global optimization. **Computational Intelligence and Neuroscience**, 2016.

ZHANG, J. et al. Model selection in svms using differential evolution. **IFAC Proceedings Volumes**, v. 44, p. 14717–14722, 2011.