UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA MECÂNICA

**João Manoel Herrera Pinheiro**

# Um estudo sobre Algoritmos de Boosting e a Otimização de Hiperparâmetros Utilizando Optuna

**Palavras-chave:** Gradient Boosting. Aprendizado de Máquina Supervisionado. Classificação. Otimização.

Fevereiro de 2023

São Carlos - SP

João Manoel Herrera Pinheiro

# Um estudo sobre Algoritmos de Boosting e a Otimização de Hiperparâmetros Utilizando Optuna

Monografia apresentada ao Curso de Engenharia Mecatrônica da Escola de Engenharia de São Carlos, Universidade de São Paulo – EESC-USP, como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico .

Curso de Engenharia Mecatrônica da Escola de Engenharia de São Carlos, Universidade de São Paulo – EESC-USP

Orientador Dr. Marcelo Becker

São Carlos – SP

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Pinheiro, João Manoel Herrera
P654u Um estudo sobre Algoritmos de Boosting e a Otimização de Hiperparâmetros Utilizando Optuna / João Manoel Herrera Pinheiro; orientador Marcelo Becker. São Carlos, 2023.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2023.

1. Gradient Boosting. 2. Aprendizado de Máquina Supervisionado. 3. Classificação. 4. Otimização dos Hiperparâmetros. 5. Seleção de Modelos. 6. Estudo Empírico. 7. Shapley Values. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

## FOLHA DE AVALIAÇÃO

**Candidato:** João Manoel Herrera Pinheiro

**Título:** Um estudo sobre Algoritimos de Boosting e a Otimização de Hiperparâmetros Utilizando Optuna

**Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo Curso de Engenharia Mecatrônica.**

## BANCA EXAMINADORA

Professor Dr. Marcelo Becker
(Orientador)

Nota atribuída: 10,0 ( Dez ) _____
(assinatura)

Professor Dr. Daniel Varela Magalhães

Nota atribuída: 10,0 ( DEZ ) _____
(assinatura)

Professor Dr. Glauco Augusto de Paula Caurin

Nota atribuída: 10.0 ( Dez ) _____
(assinatura)

Média: 10.0 ( Dez )

Resultado: APROVADO

**Data: 14/02/2023.**

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☒ NÃO ☐ Visto do orientador _____

João Manoel Herrera Pinheiro

# A Study on Gradient Boosting Algorithms and Hyperparameter Optimization using Optuna

Undergraduate Thesis submitted to the Undergraduate Course in Mechatronics Engineering from São Carlos School of Engineering, University of São Paulo – EESC-USP, in partial fulfillment of the requirements for the bachelor degree in Mechatronics Engineer .

Undergraduate Course in Mechatronics Engineering from São Carlos School of Engineering,
University of São Paulo – EESC-USP

Advisor Dr. Marcelo Becker

São Carlos – SP

*Este trabalho é dedicado a minha família, meus pais, Danila e Luciano, e ao meu irmão, Felipe, que sempre me apoiaram nessa jornada e fizeram parte das minhas conquistas.*

# AGRADECIMENTOS

Agradeço principalmente a todos os servidores e docentes da Universidade de São Paulo que fizeram parte fundamental da minha formação. A todos os meus colegas da equipe de Formula SAE EESC-USP Tupã e da Campanha USP do Agasalho que me possibilitaram a gerencia de projetos diversos e o meu desenvolvimento em cargos de liderança.

E em especial os meus maiores agradecimentos vão para os meus grandes amigos que fizeram parte dessa minha jornada. Muito obrigado Letícia, Leonardo, Felippe, Roberto e Rayza.

Agradecimentos especiais direcionados à Maíra Martins da Silva, Leopoldo Pisanelli Rodrigues de Oliveira e Marcelo Becker, professores que me orientaram nos projetos e iniciações cientificas que eu realizei ao longo da graduação contribuindo para o meu desenvolvimento acadêmico.

E por último agradecimento ao Itaú Unibanco e a todos os meus colegas de trabalho que me deram a oportunidade de iniciar a carreira na área de Ciência de Dados.

*"Once you stop learning*
*you start dying."*
*(Albert Einstein)*

# RESUMO

PINHEIRO, J. M. H. **Um estudo sobre Algoritmos de Boosting e a Otimização de Hiperparâmetros Utilizando Optuna**. 2023. 147 p. Monografia (Engenharia Mecatrônica) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos – SP, 2023.

Este trabalho é um estudo sobre os algoritmos de *Gradient Boosting Machines* (GBMs), os quais são algoritmos de aprendizado de máquina supervisionado que vem obtendo excelente resultados em uma ampla gama de problemas e vencendo diversas competições de aprendizado de máquina. Neste trabalho vamos abordar três bibliotecas que implementam os GBMs, essas bibliotecas são: o XGBoost, CatBoost e LightGBM. Foi realizado um comparativo entre as técnicas e os resultados obtidos, e então partimos para a otimização dos hiperparâmetros utilizando o Optuna. Ao construir um modelo de aprendizado de máquina, a otimização de hiperparâmetros pode se tornar uma tarefa desafiadora e demorada, dependendo do número e do espaço de busca do procedimento de otimização. Definir os hiperparâmetros é uma tarefa fundamental. É necessário entender como eles afetam o resultado do modelo e a partir disso, implementamos o Optuna para os três algoritmos, um framework de otimização de hiperparâmetros. Os conjuntos de dados deste experimento são consumidos da plataforma Kaggle e OpenML. O primeiro experimento realizado foi um comparativo entre os algoritmos pelas métricas de validação e em seguida tentou-se alterar os hiperparâmetros para identificar a influência deles no modelo. Inicialmente, não foi possível obter uma conclusão. Na sequência, foi feita uma análise mais complexa pelo estudo do Optuna. Nesses resultados conseguimos identificar quais hiperparâmetros tiveram seu maior impacto na importância do estudo e se houve alterações desses hiperparâmetros em cada conjunto de dados. Além disso, foi possível melhorar todos os modelos preditivos com apenas a utilização do Optuna. E, ainda, para o modelo com a maior complexidade de previsão, o Optuna demonstrou ser eficaz e resultou em aumento de performance. Com a utilização dos *Shapley Additive exPlanations (SHAP Values)*, conseguimos interpretar os modelos e como cada variável impactou no conjunto de treino.

**Palavras-chave:** Gradient Boosting, Aprendizado de Máquina Supervisionado, Otimização dos Hiperparâmetros, Classificação, Seleção de Modelos, Estudo Empírico, Shapley Values.

# ABSTRACT

PINHEIRO, J. M. H. **A Study on Gradient Boosting Algorithms and Hyperparameter Optimization using Optuna**. 2023. 147 p. Undergraduate Thesis (Mechatronics Engineer – São Carlos School of Engineering, University of São Paulo, São Carlos – SP, 2023.

This thesis is a study about Gradient Boosting Machines (GBMs) algorithms, which are supervised machine learning algorithms that have been getting excellent results on a wide range of problems and winning numerous machine learning competitions. In this work we will approach three libraries that implement the GBMs, these libraries are: XGBoost, CatBoost and LightGBM. A comparison was made between the techniques and the results from each algorithm, and then we started to optimize the hyperparameters using Optuna. When building a machine learning model, hyperparameter optimization can become a challenging and time-consuming task, depending on the number of hyperparameters and the space of the optimization. Defining the hyperparameters is a fundamental task. It is required understand how they affect the result of the model and from that, we implement Optuna, a hyperparameter optimization framework, to the three algorithms. The data sets from this experiment are obtained from the Kaggle and OpenML platform. The first experiment was a comparison between the algorithms using validation metrics. Then we attempt to identify the hyperparameters influence on the model by changing their values. Initially, it was not possible to reach a conclusion. Subsequently, a more complex analysis was carried out using Optuna. In these results we were able to identify which hyperparameters had bigger impact on the importance of the study and whether there were changes in these hyperparameters in each data set. Furthermore, it was possible to improve all predictive models with only the use of Optuna. And yet, for the model with the greatest complexity, Optuna proved to be effective and resulted in increased performance. With the use of Shapley Additive exPlanations (SHAP Values), we were able to interpret the models and how each variable impacted on the training set.

**Keywords:** Gradient Boosting, Supervised Machine Learning, Hyperparameter Optimization, Classification, Model Selection, Empirical Study, Shapley Values..

# LISTA DE ILUSTRAÇÕES

# LISTA DE ALGORITMOS

# LISTA DE CÓDIGOS-FONTE

# LISTA DE TABELAS

# LISTA DE SÍMBOLOS

$x_i$ — o exemplo i-enésimo de um conjunto de dados, vetor ou escalar

$\mathbb{R}$ — Conjunto dos números reais

$\mathbb{Z}$ — Conjunto dos números inteiros

$\mathcal{T}$ — Conjunto de treino

$H_0$ — Hipótese Nula

$H_1$ — Hipótese Alternativa

$\mathcal{H}$ — Espaço de Hiperparâmetro

$\eta, LR$ — *learning rate*, taxa de aprendizado

$MD$ — max_depth, hiperparâmetro da profundidade máxima

$RegL$ — reg_lambda, hiperparâmetro da regularização $L_2$

# SUMÁRIO

CAPÍTULO

1

# INTRODUÇÃO

Técnicas de *Machine Learning* estão sendo utilizadas cada vez mais na indústria e ciência com o principal objetivo de obter informações através dos dados. O aumento do uso dessas técnicas se deve ao volume cada vez maior de dados sendo produzidos, bem como o aumento na performance para processar esses dados e a redução de custos para armazená-los. Os modelos de *machine learning* estão sendo usados em diversas áreas da ciência, desde o diagnóstico de algumas doenças pelo reconhecimento de imagens em laboratórios médicos até o reconhecimento de padrões e comportamentos com o objetivo de mapear consumidores e facilitar a tomada de decisão para a venda ou recomendação de algum produto. Com esse forte crescimento, a pesquisa na área teórica e na aplicação de técnicas de *machine learning* vem se se destacando, mostrando a importância de um modelo robusto de *machine learning* para as aplicações científica e industrial.

*Gradient Boosting Machines* (GBMs) vem obtendo resultados no estado da arte, principalmente em casos de dados estruturados, ou seja, dados que possuem um padrão bem definido. Por exemplo: dados organizados através de linhas e colunas, onde não é permitido dados diferentes das estruturas preestabelecidas. Se a coluna de uma tabela foi criada para ser numérica, ela não aceitará outro tipo de dados. Em uma pesquisa feita pela plataforma Kaggle,*State of Data Science and Machine Learning 2022* (KAGGLE, 2022), GBMs como XGBoost, LightGBM e CatBoost estão entre os algoritmos mais utilizado pelos respondentes da pesquisa. Esses algoritmos são baseados em árvore de decisão e utilizam uma estrutura de *Gradient boosting* para aumentar a performance do modelo.

Além da escolha do melhor algoritmo para construir o seu modelo de *Machine Learning* temos o desafio de escolher os hiperparâmetros para o treino do modelo. Eles são de extrema importância pois são atributos que controlam o treinamento do modelo de *machine learning*. Eles previnem o modelo de aprender apenas com os dados mostrados, ou seja, previnem *overfitting* e *underfitting*, tornando-o capaz de generalizar para outras situações possíveis. Nas implementações mencionadas – XGboost, LightGBM e CatBoost – todos os hiperparâmetros costumam ter

valores pré-estabelecidos. Encontrar os melhores hiperparâmetros é uma tarefa extremamente desafiadora na área de *Machine Learning*. Após o comparativo dos modelos de *boosting*, o melhor modelo será escolhido para utilizar uma ferramenta de automatização de hiperparâmetros, o Optuna. Assim, podemos avaliar se apenas o ajuste dos hiperparâmetros é capaz de melhorar o resultado do modelo treinado.

Neste capítulo será abordada a revisão bibliográfica sobre o tema aprendizado de máquina e o embasamento teórico do problema de classificação. Também serão apresentadas as características do conjunto de dados, explicando o conceito de cada variável e a distinção entre variáveis explicativas e variável resposta.

## 1.1 Conceitos Iniciais

Diversos autores definem aprendizagem como a capacidade de um programa computacional conseguir melhorar seu desempenho através do aprendizado (HASTIE; TIBSHIRANI; FRIEDMAN, 2009) (MITCHELL, 1997), tornando suas ações mais precisas (MARSLAND, 2014) ou inferir regras gerais através da observação de exemplos (LUXBURG; SCHOELKOPF, 2008). Ou seja, a partir de algoritmos de aprendizagem o computador consegue aprender tarefas específicas. O conceito chave para as máquinas é essa aprendizagem a partir dos dados e que com isso a máquina consiga generalizar, definida como a identificação de similaridade entre diferentes situações, é o que torna a aprendizagem útil, pois o conhecimento adquirido pode ser aplicado em diversos cenários.

### 1.1.1 Aprendizado de Máquina

O aprendizado de máquina mescla ideias da matemática, física, neurociência e biologia. Segundo estes autores, o aprendizado de máquina, diferente da inteligência artificial, não procura gerar um comportamento inteligente, mas descobrir mecanismos através dos quais tarefas específicas podem ser aprendidas por computadores, ou seja com o foco na capacidade de generalização (HASTIE; TIBSHIRANI; FRIEDMAN, 2009) (MARSLAND, 2014). O aprendizado de máquina tem origens na estatística, ciência da computação e inteligencia artificial e agora estabeleceu-se como uma disciplina por direito próprio (LUXBURG; SCHOELKOPF, 2008).

De maneira geral os problemas de aprendizado podem ser classificados como:

- **Aprendizado Supervisionado:** um conjunto de respostas corretas é fornecido para esse aprendizado de tal forma que o algoritmo escolhido consiga generalizar para responder corretamente as possíveis novas entradas. Alguns algoritimos são ,por exemplo, regressão linear, análise de regressão logística, *K-Nearest Neighbors*, S*upport Vector Machines*, *Random Forests* e Árvores de Decisão.

- **Aprendizado Não Supervisionado:** o conjunto de respostas corretas não é fornecido para o algoritmo e, portando, o algoritmo procura identificar padrões ou similaridades entre os objetos de entrada para que eles apresentem algo em comum e que seja possível agrupá-los. Nesse aprendizado, busca-se encontrar o padrão entre as diferentes amostras e separar as que possuem as mesmas características. Um exemplo desse aprendizado são os algorítimos de clusterização que tem como objetivo agrupamento dos dados que possuem características semelhantes, por exemplo, *K-Means*, DBSCAN e *Hierarchical Cluster Analysis* (HCA).

- **Aprendizado Por Reforço:** O sistema de aprendizado, que nesse contexto receberá o nome de agente, é treinado para tomar uma sequência de decisões. Nesse tipo de aprendizado o agente executa alguma ação e recebe recompensas ou penalidades em troca. Seu objetivo é ser capaz de tomar as melhores decisões para obter o maior número de recompensas ao longo do tempo, usando testes totalmente aleatórios no início e obtendo a melhor solução ao final. Diferente dos outros, este aprendizado não precisa de dados rotulados para tomar a sua ação. Alguns algoritmos são, por exemplo, Processo de Decisão de Markov, *Q-Learning*, Monte Carlo, *Deep Q-Learning* e *Deep Deterministic Policy Gradient*.

### 1.1.2 Problema de Classificação

Em um problema de classificação temos dois tipos de espaço para ser considerado, conforme abordado em (LUXBURG; SCHOELKOPF, 2008), temos o espaço $X$ das instancias ou objetos e o espaço $Y$ das saídas ou categorias. No caso que será abordado neste trablaho, a classificação binária, o espaço de saídas é um indicador booleano dado pelo conjunto $(0,+1)$ – em alguns casos podemos ter o conjunto $(-1,+1)$. O objetivo é encontrar um mapeamento $f : X \to Y$ chamado de classificador. Um algoritmo de classificação, portanto, é um procedimento que toma como entrada os exemplos de treinamento e produz como saída um classificador $f$. Este tópico será abordado com maiores detalhes no Capítulo 2.

De maneira geral, os tópicos que descrevem os processos para resolver um problema de aprendizado supervisionado se resumem em (HASTIE; TIBSHIRANI; FRIEDMAN, 2009) (MARSLAND, 2014):

1. **Coleta e preparação dos dados:** os dados precisam ser coletados e tratados antes de colocar no treino do modelo. É preciso fazer uma análise estatística do conjunto de dados e realizar os tratamentos necessários para os dados faltantes e/ou *outliers*.

2. **Seleção de variáveis:** requer o conhecimento prévio do problema e dos dados coletados para identificar quais variáveis podem ser úteis para resolver o problema e, se necessário, criar novas variáveis a partir das existentes ou até aplicar redução de dimensionalidade nas variáveis.

3. **Escolha do algoritmo:** definidos o problema e o conjunto de dados, um ou mais algoritmos de aprendizado de máquina são selecionados.

4. **Determinação dos parâmetros:** alguns algoritmos necessitam que alguns parametros sejam colocados manualmente, por exemplo, número de interações, tolerância ao erro, tamanho da amostra, etc.

5. **Determinação dos Hiperparâmetros:** são parâmetros que definem a arquitetura de um método de aprendizado. Enquanto os parâmetros do modelo especificam como transformar os objetos de entrada na saída desejada, os hiperparâmetros definem como o modelo é realmente estruturado. Os hiperparâmetros não podem ser estimados diretamente do conjunto de treinamento. Alguns exemplos de hiperparâmetros são taxa de aprendizado, número de épocas, número de galhos e folhas em uma árvore de decisão, profundidade máxima da folha, etc.

6. **Treinamento:** com os dados e os algorítimos essa etapa consiste no uso computacional a fim de construir uma função $\hat{f}(x)$ que tem como objetivo prever as saídas a partir de novos dados.

7. **Validação:** esta etapa inclui a seleção de métricas para avaliação de desempenho do modelo construído, neste caso um classificador. São utilizados dados que não foram usados no conjunto de treinamento do modelo.

8. **Tuning do Modelo:** esta etapa consiste em melhorar a performance do modelo apenas alterando os parâmetros e/ou hiperparâmetros. É um processo complexo que é utilizado para melhorar a performance dos modelos treinados. Consiste em realizar diversos treinos alterando os hiperparâmetros de forma sistemática e, no final, escolher o modelo com a melhor métrica de performance.

9. **Predições:** com o modelo finalizado, fazer as predições significa fornecer a ele dados novos e obter as saídas. Essa é a última etapa da resolução de um problema de aprendizado supervisionado, onde podemos concluir que o problema foi resolvido.

### 1.1.3 Variáveis e Conjunto de Dados

O conjunto de dados é representado por um vetor de características que descreve as observações. As variáveis são mensuradas em cada elemento da amostra ou população que apresentam características diferentes de indíviduo para indivíduo. Quando qualquer medição de uma caracteristica está ausente, dizemos que aquele valor é *missing*. Cada observação, também denominada objeto ou registro, corresponde a uma ocorrência dos dados e cada variável está associada a uma propriedade específica dos objetos, podendo esta propriedade retratar noções físicas ou abstratas. Uma coleção de observações assim constituídas compõe um conjunto de

dados. As variáveis podem assumir valores numéricos e não numéricos, podemos classifica-las em:

- **Numéricas ou quantitativas:**
    1. **Contínuas:** são variáveis mensuráveis que se originam de algum processo de medição e que podem assumir qualquer valor em uma escala contínua seja $x_i$ para uma variável contínua temos que $x_i \in \mathbb{R}$ .Exemplos: salário, pressão arterial, altura e etc.
    2. **Discretas:** são variáveis que são originadas de um processo de contagem, que podem assumir um número finito ou infinito contável de valores. Seja $x_i$ uma variável discreta, temos que $x_i \in \mathbb{Z}$ . Exemplos: número de filhos, quantidade de quartos em uma casa e etc.
- **Categóricas ou qualitativas:**
    1. **Ordinal:** os valores apresentam uma ordem definida e é possível determinar qual é o maior valor e o menor valor. Por exemplo: escolaridade, *rating* de risco de um país, meses do ano, nota de um teste e etc.
    2. **Nominal:** os valores não apresentam uma ordem definida são penas nomes diferentes que representam alguma característica. Por exemplo: sexo, cor dos olhos, profissão, religião e etc.

Os conjuntos de dados apresentam atributos de entrada, também denominados **variáveis explicativas, preditoras ou independentes**, e podem apresentar um atributo alvo, também chamado de *target*, **variável resposta ou variável dependente**, que representa o evento que queremos realizar as nossas previsões. Nos problemas de regressão, a varíavel resposta contém valores númericos contínuos enquanto que, nos problemas de classificação, o atributo alvo é uma variável discreta, geralmente um indicador booleano $(0,1)$.

Um conjunto de dados é representado por uma matriz de objetos $X_{n\times d}$ em que $n$ é o número de objetos e $d$ é o número de atributos de entrada. Este valor define a dimensão da base de dados e cada elemento $x_{ij}$ da matriz contém o valor da $j$-ésima característica para o $i$-ésmo objeto com $j = 1,2,3,...,d$ e $i = 1,2,3,...,n$. De maneira análoga, a $i$-ésima observação da variável resposta é representada por $y_i$.

## 1.2 Objetivos

O principal objetivo desse trabalho é o estudo dos três algorítimos de *boosting* XGboost, LGBM e CatBoost em um problema de classificação binária, comparando a performance e o custo computacional (tempo de execução) de cada algoritmo em diferentes *datasets*. Também é feita uma análise dos hiperparâmetros em cada uma das implementações, avaliando como eles

afetam o resultado final dos modelos. E por fim, os hiperparâmetros do LGBM são otimizados utilizando o Optuna, um framework Open Source voltado para automatizar a busca pelos melhores hiperparâmetros. Podemos sumarizar o estudo deste trabalho nos seguintes pontos:

1. Quais as principais diferenças de cada algoritmo? O conjunto de dados afeta o resultado final em cada algoritmo ? Qual o algoritmo que obteve a melhor performance tanto do ponto de vista computacional e na métrica de validação ?
2. Como os hiperparâmetros afetam o treino do modelo?
3. Como funciona o Optuna? Após a aplicação do Optuna, conseguimos melhorar a métrica de validação do modelo?

## 1.3 Descrições dos Capítulos

Nos próximos dois capítulos, Capítulo 2 e Capítulo 3 serão introduzidos os fundamentos teóricos para o aprendizado supervisionado, as métricas de validações, os algoritmos de boosting e o que são hiperparâmetros. No Capítulo 4 temos as implementações das funções dos algoritmos e do optuna, de tal forma que o experimento fique replicável e comparável entre os algoritmos. Além disso, discriminamos a estrutura dos testes realizados. No Capítulo 5, temos o resultado dos testes implementados no Capítulo 4, além de uma análise estatística dos conjuntos de dados utilizados. Buscamos ainda interpretar quais variáveis impactam o modelo. No Capítulo 6 será abordada a análise final, comparando os trés algoritmos, suas métricas de validação, de performance e custo computacional. Em seguida, teremos a análise de alguns hiperparâmetros e, por final, o modelo otimizado pelo Optuna e sua performance. Culminamos no Capítulo 7, onde teremos a conclusão final de todo o trabalho realizado neste estudo, uma análise das limitações do mesmo, bem como sugestões de trabalhos futuros.

Nos apêndices, se encontram os materiais complementares que contém inclusive o Jupyter Notebook, com os códigos implementados. Há também a referência da página no *GitHub*, onde o código fonte final deste trabalho está disponibilizado.

CAPÍTULO

# 2

# FUNDAMENTOS DE APRENDIZADO SUPERVISIONADO

No aprendizado de máquina, distinguimos três tipos de aprendizado: supervisionado, não supervisionado e de reforço. A distinção geralmente é feita pelo feedback o agente recebe. A aprendizagem supervisionada é a tarefa de aprender uma função a partir de um conjunto de dados rotulados. Isso significa que os dados consistem em exemplos de treinamento: as entradas e seus valores de saída desejados. O problema pode ser uma classificação, se a saída for uma classe, ou um regressão, se a saída for um ou vários valores reais. Em ambos os casos, o feedback é a saída correta que corresponde à entrada fornecida. Um modelo treinado totalmente supervisionado serve como uma função que podemos usar para mapear novas entradas. O objetivo é ser capaz de rotular dados novos corretamente: exemplos que não estavam no conjunto de treinamento.

Em resumo, o principal objetivo de *Machine Learning* é descobrir padrões a partir do conjunto de dados para que se possa obter *insights* e resolver diversos problemas em campos diferentes. Problemas de aprendizado supervisionado lidam especificamente com dados no formato $(\mathbf{x}^{(i)}, y^{(i)})$, em que $y^{(i)} \in \mathbb{R}$ é a saída desejada. Ou seja, a saída também está contida no conjunto de dados.

Em (RUSSELL; NORVIG, 2021) o principal objetivo do aprendizado supervisionado pode ser definido como:

Dado um **conjunto de treino** com $N$ pares de entrada-saída

$$\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), ..., (\mathbf{x}^{(N)}, y^{(N)})\}$$

onde cada $y^{(i)}$ foi gerado por uma função desconhecida $y = f(x)$, devemos descobrir uma função $h$ que se aproxima da real função $f$.

A função $h$ é chamada de **Hipótese** e está contida em um **Espaço de Hipóteses** denominado de $\mathcal{H}$ ou seja o espaço contem a classe de funções consideradas por um algoritmo de

aprendizagem, que irá escolher a função $h \in \mathcal{H}$ baseado em uma função de erro calculada no **conjunto de treino** em que a saída $y_i$ é o valor que estamos tentando prever com o nosso modelo.

Em seguida, listamos alguns exemplos clássicos de aprendizado supervisionado: Detecção de fraude em cartões de credito em que $\mathbf{x}^{(i)}$ seriam informações da transação (local, valores, forma de pagamento) e o $y^{(i)}$ um indicador booleano se a $i$-th é uma transação fraudulenta ou não. Predição de preços de imóveis em que $\mathbf{x}^{(i)}$ seriam informações do imóvel (local, área do imóvel, quantidade de quartos, ano de construção) e o $y^{(i)}$ é o valor do imóvel. Classificação de dígitos manuscritos, onde $\mathbf{x}^{(i)}$ é uma imagem ou a representação de um número e $y^{(i)}$ o número correspondente.

## 2.1 Função Aproximação e Otimização

A tarefa do aprendizado supervisionado pode ser interpretada como um problema de otimização. Conforme abordado em (HASTIE; TIBSHIRANI; FRIEDMAN, 2009), temos o paradigma da *function-fitting*. Do ponto de vista de aprendizado de máquina, vamos assumir que os erros são aditivos e que temos o modelo $Y = f(x) + \varepsilon$. A aprendizagem supervisionada tenta aprender $f$ por exemplo através de um *professor*.

Observa-se o sistema em estudo, tanto as entradas quanto as saídas, e monta-se um conjunto de treinamento de observações $\mathcal{T} = (\mathbf{x}^{(i)}, y^{(i)}), i = 1, ..., N$ . Os valores de entrada observados para o sistema $\mathbf{x}^{(i)}$ também são alimentados em um sistema artificial, conhecido como um algoritmo de aprendizagem, que também produz saídas $\hat{f}(x^{(i)})$ em resposta. O algoritmo de aprendizado tem a propriedade de poder modificar sua relação de entrada/saída $\hat{f}$ em resposta às diferenças $y^{(i)} - \hat{f}(x^{(i)})$ entre as saídas originais e geradas. Esse processo é conhecido como *aprendizado por exemplo*. Após a conclusão do processo de aprendizagem, o objetivo final é que as saídas artificiais e as saídas reais estejam próximas o suficiente para o modelo consiga generalizar para todos os conjuntos de entradas.

As diferentes funções $\hat{f}$ podem ser estimadas a partir de diferentes modelos em que a hipótese escolhida se aproximará da verdadeira função $f$. Alguns exemplos são estimativa por máxima verossimilhança, o método dos mínimos quadrados, expansões lineares e transformação *sigmoid*.

Para estimar os parâmetros dessas aproximações, os algoritmos de aprendizado geralmente realizam uma otimização sobre uma *função de perda* (KUHN; JOHNSON, 2013) que depende do aproximador escolhido.

## 2.2 Viés-Variância *Tradeoff*

O objetivo dos modelos de aprendizado de máquina é estimar a função que melhor ajusta aos dados de entrada para obter previsões corretas de forma generalizada. A melhor maneira de

medir e otimizar o desempenho do modelo é levar em consideração o viés e a variância.

O problema básico de aprendizado supervisionado geralmente consiste em dois conjuntos de dados: **conjunto de treinamento** e o **conjunto de teste**. Um modelo é treinado usando dados do conjunto de treinamento, resultando em um modelo $\hat{f}$, então este modelo prevê os dados no conjunto de teste, e com o previsto $\hat{y}^{(i)}$ é possível estimar o erro de previsão do modelo.

Se estivermos em uma situação com bastante volume de dados, a melhor abordagem é dividir aleatoriamente o conjunto de dados em três partes: um conjunto de treinamento, um conjunto de validação e um conjunto de teste. O conjunto de treinamento é usado para ajustar os modelos, a validação é usado para estimar o erro de previsão para a seleção do modelo, o conjunto de teste é utilizado para avaliação do erro do modelo final escolhido.

Idealmente, o conjunto de teste deve ser mantido separado dos outros conjuntos e ser utilizado apenas no final da análise de dados. Suponha que usamos o conjunto de teste repetidamente, escolhendo o modelo com o menor erro desse conjunto, então o erro definitivo do modelo final escolhido irá subestimar o verdadeiro erro de teste, às vezes substancialmente. É difícil dar uma regra geral sobre como escolher o número de observações em cada uma das três partes, pois isso depende do conjunto de dados em si, quantidade de ruído e o tamanho da amostra de treinamento, porém uma divisão típica pode ser 50% para treinamento e 25% cada para validação e teste:

Figura 1 – Divisão de Treino, Validação e Teste


O *tradeoff* de viés-variância (HASTIE; TIBSHIRANI; FRIEDMAN, 2009) na modelagem preditiva afeta a capacidade de um método de aprendizagem para generalizar. Alta variância indica que o modelo ajusta o ruído aleatório no conjunto de treinamento, o que geralmente resulta em baixo poder de generalização, causando o **overfitting**. Por outro lado, um modelo com viés alto tem uma diferença muito baixa no erro de previsão entre o conjunto de treinamento e conjunto de teste, mas geralmente tem desempenho ruim causando o **underfitting**.

No treinamento e avaliação do modelo, é importante verificar o desempenho da generalização, e isso geralmente é feito usando o conjunto de teste. No processo de avaliação, o **overfitting** e o **underfitting** podem ser detectados comparando o erro de previsão no conjunto de treinamento e teste.

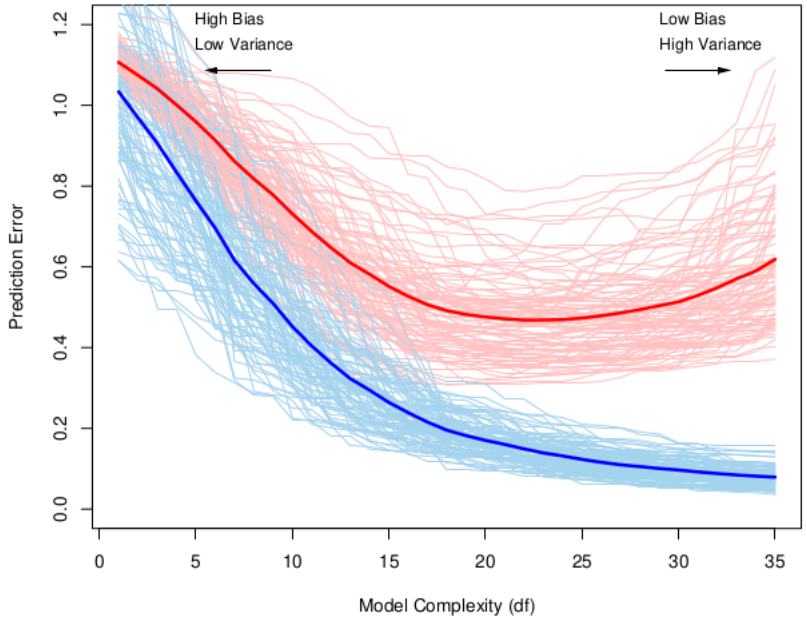
A Figura 2 ilustra uma questão importante na avaliação da capacidade de generalização do modelo. Considere primeiro o caso de um quantitativo ou intervalo resposta de escala. Temos uma variável de resposta $Y$ , um vetor de entradas $\mathbf{X}$ e um modelo de predição $\hat{f}(X)$ que foi estimado a partir de um conjunto de treinamento $\mathcal{T}$ .

A função para medição do erro entre $Y$ e $\hat{f}(X)$ é denotada por $L(Y, \hat{f}(X))$. Ainda neste

Capítulo serão abordadas demais formas para medir o desempenho, mas algumas escolhas típicas para esse erro são:

$$L(Y,\hat{f}(X)) = \begin{cases} (Y-\hat{f}(X))^2 & \text{erro quadrático} \\ |Y-\hat{f}(X)| & \text{erro absoluto} \end{cases} \tag{2.1}$$

Figura 2 – Comportamento das amostra de teste e treino conforme a complexidade do modelo é variada. As curvas em azul claro mostram o erro $\overline{err}$ de treinamento, enquanto o curvas vermelhas claras mostram o erro de teste condicional $Err_{\mathcal{T}}$ para 100 conjuntos de treinamento de tamanho 50 cada, à medida que a complexidade do modelo aumenta. As curvas sólidas mostram o erro de teste esperado $Err$ e o erro de treinamento esperado $E[\overline{err}]$ (HASTIE; TIBSHIRANI; FRIEDMAN, 2009)




O erro do teste em uma amostra independente é dado por:

$$Err_{\mathcal{T}} = E[L(Y,\hat{f}(X))|\mathcal{T}] \tag{2.2}$$

Em que tanto $X$ quanto $Y$ são amostras aleatórias da população. Neste caso, o conjunto de treino $\mathcal{T}$ é fixo e o erro do conjunto de teste é referente a este conjunto de treino fixo $\mathcal{T}$. Podemos escrever esse erro como:

$$Err = E[L(Y,\hat{f}(X))] = Err_{\mathcal{T}} \tag{2.3}$$

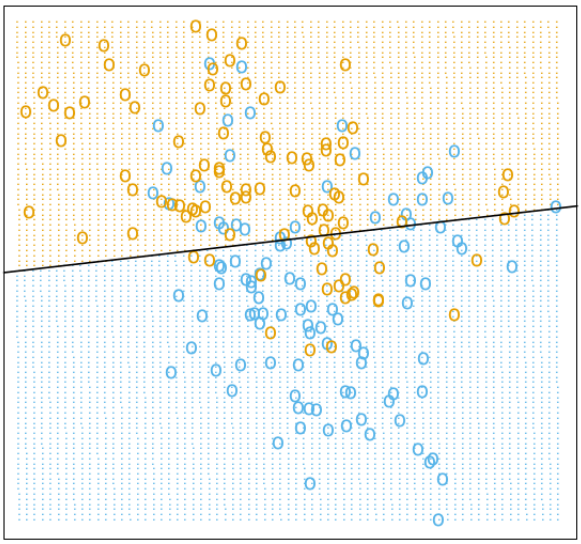
O objetivo é estimar o $Err_{\mathcal{T}}$.

## 2.3 Problema de Classificação

Conforme foi abordado no Capítulo 1, quando os valores de $f$ pertencem a um conjunto finito temos um problema de classificação e especialmente quando o valor de $f$ assume apenas

dois possíveis valores denotamos de **classificação binária**.

Figura 3 – Exemplo de limite de decisão para um classificador binário (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).



O exemplo de detecção de fraude no cartão de crédito que foi abordado no começo deste Capítulo é um problema de classificação bińaria.

## 2.4 Métricas de Avaliação de Performance

Existem diversas métricas para avaliar a performance de modelos de classificação. Neste estudo as métricas abordadas serão a *AUC*, que é um acrônimo da área em baixo da curva característica de operação do receptor, *Logloss*, que é a perda logarítmica, e o *KS*, que é o Teste Kolmogorov-Smirnov. Um breve resumo de cada métrica:

- **AUC:** é amplamente utilizado em pesquisas, e é uma boa métrica insensível ao desequilíbrio de classes, tendo um valor limitado entre 0.5 e 1.

- **Logloss:** *logloss* é a função direta otimizada nos experimentos, ou seja, as etapas de aumento de gradiente estão tentando reduzir a perda logarítmica em cada etapa. O *logloss* leva em conta a incerteza da previsão e quanto o rótulo real difere dela, tendo seu valor limitado a $]\infty, 0]$

- **KS:** o teste Kolgomorov-Smirnov (KS) busca avaliar a distância entre uma distribuição conhecida e uma distribuição que foi observada empiricamente. A hipótese nula do KS é que a amostra segue a mesma distribuição que a normal, tendo o valor limitado entre 0 e 1.

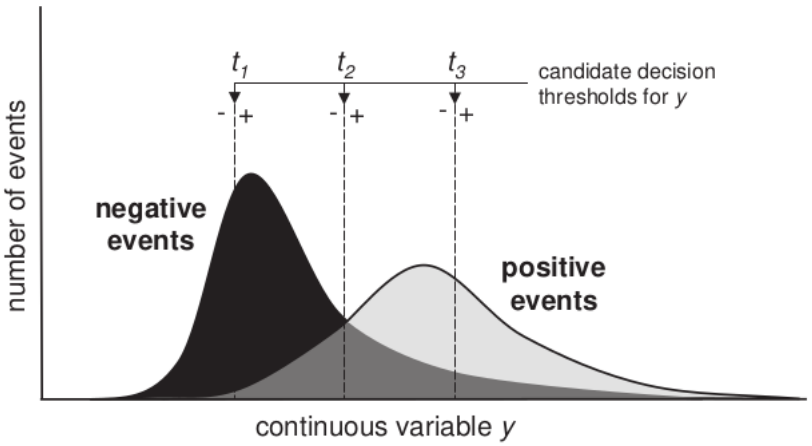
### 2.4.1 AUC

A curva característica de operação do receptor é uma curva de probabilidade que mede a previsão de um classificador binário dado um *threshold*. Os resultados de uma classificação binária podem ser resumidos em uma matriz de confusão, ilustrada na Tabela 1.Os Verdadeiro-positivo (VP), Falso-positivo (FP), Falso-negativo (FN) e Verdadeiro-negativo (VN) são calculados dispondo todos os dados na matriz, dependendo da classe prevista e da classe real de cada ponto no conjunto de dados.

Tabela 1 – Matriz de confusão para o problema de classificação binária.

| **Resultado** | **Classe Real** | |
|---|---|---|
| **Classe Predita** | **Positivo** | **Negativo** |
| **Positivo** | Verdadeiro-positivo (VP) | Falso-positivo (FP) |
| **Negativo** | Falso-negativo (FN) | Verdadeiro-negativo(VN) |

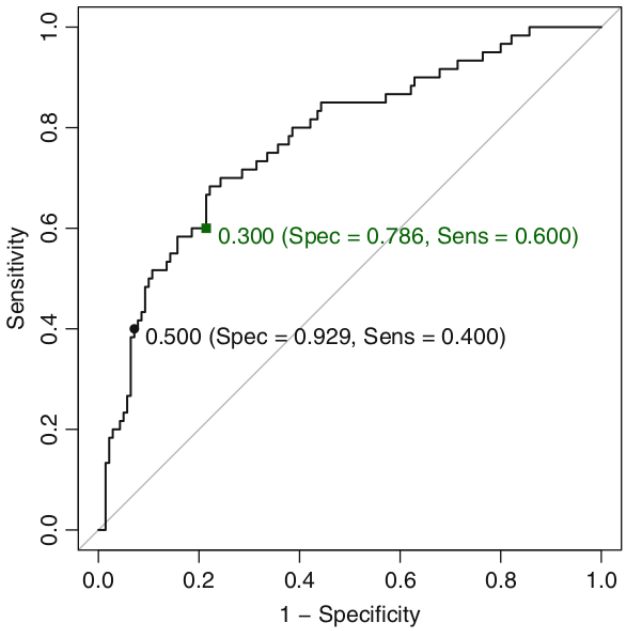
Figura 4 – Uma ilustração da ocorrência de eventos positivos e negativos quando ordenado pela variável contínua $y$ (BROWN; DAVIS, 2006).




Conforme abordado em (BROWN; DAVIS, 2006), (FAWCETT, 2006) e (KUHN; JOHNSON, 2013) a curva característica do operador (ROC) é calculada pelo gráfico de *Taxa de Verdadeiros Positivos (TVP)*, ou Sensibilidade, versus *Taxa de Falso Positivos (TFP)*, ou Especificidade, onde $\mathcal{T}$ é o *threshold*.

$$Roc_y(\mathcal{T}) = TVP_{\mathcal{T}} = \frac{VP_{\mathcal{T}}}{VP_{\mathcal{T}} + FN_{\mathcal{T}}}$$
$$Roc_x(\mathcal{T}) = TFP_{\mathcal{T}} = \frac{FP_{\mathcal{T}}}{FP_{\mathcal{T}} + VN_{\mathcal{T}}}$$

Figura 5 – Curva ROC de um classificador (KUHN; JOHNSON, 2013), dois pontos estão em destaques para mostrar diferentes valores de *Especificidade* e *Sensibilidade*




Todas as métricas dependem do *threshold* $\mathcal{T}$. A **AUC** é definida como a área em baixo da curva ROC e fornece uma medida agregada do desempenho do classificador em relação a todos os *threshold* possíveis. O valor é entre 0.5 (pior performance) e 1 (performance perfeita).

Abaixo temos os algoritmos para a geração dos pontos da curva **ROC** e do cálculo da **AUC** (FAWCETT, 2006).

**Algoritmo 1** – Algoritmo de geração de curva ROC

**Input:** $L$, o conjunto de exemplos de teste; $f(i)$, o modelo probabilístico dos classificadores que estimam que o exemplo $i$ é positivo, $P$ e $N$, o número de exemplos positivos e negativos.
**Output:** $R$, uma lista de pontos ROC aumentando pela taxa FP.
**Requer:** $P > 0$ e $N > 0$.

1: $L_{sorted}$ ▷ $L$ ordenada pelos valores decrescentes de $f$
2: $FP \leftarrow VP \leftarrow 0$
3: $R \leftarrow []$
4: $f_{prev} \leftarrow -\infty$
5: $i \leftarrow 1$
6: **enquanto** $i \leq |L_{sorted}|$ **faça**
7: **se** $f(i) \neq f_{prev}$ **então**
8: push $(\frac{FP}{N}, \frac{VP}{P})$ onto $R$
9: $f_{prev} \leftarrow f(i)$
10: **fim se**
11: **se** $L_{sorted}[i]$ for positivo **então**
12: $VP \leftarrow VP + 1$
13: **senão**
14: $FP \leftarrow FP + 1$
15: **fim se**
16: $i \leftarrow i + 1$
17: **fim enquanto**
18: push $(\frac{FP}{N}, \frac{VP}{P})$ onto $R$

**Algoritmo 2** – Algoritmo para calcular a área em baixo da curva ROC

**Input:** $L$, o conjunto de exemplos de teste; $f(i)$, o modelo probabilístico dos classificadores que estimam que o exemplo $i$ é positivo, $P$ e $N$, o número de exemplos positivos e negativos.
**Output:** $A$, a área em baixo da curva ROC.
**Requer:** $P > 0$ e $N > 0$.

$L_{sorted}$ ▷ $L$ ordenada pelos valores decrescentes de $f$
$FP \leftarrow VP \leftarrow 0$
$FP_{prev} \leftarrow VP_{prev} \leftarrow 0$
$A \leftarrow 0$
$f_{prev} \leftarrow -\infty$
$i \leftarrow 1$
**enquanto** $i \leq |L_{sorted}|$ **faça**
    **se** $f(i) \neq f_{prev}$ **então**
        $A \leftarrow A +$ AREA_TRAPEZIO$(FP, FP_{prev}, VP, VP_{prev})$
        $f_{prev} \leftarrow f(i)$
        $FP_{prev} \leftarrow FP$
        $VP_{prev} \leftarrow VP$
    **fim se**
    $i \leftarrow i + 1$
    **se** $i$ for positivo **então**
        $VP \leftarrow VP + 1$
    **senão**
        $FP \leftarrow FP + 1$
    **fim se**
    $i \leftarrow i + 1$
**fim enquanto**
$A \leftarrow A +$ AREA_TRAPEZIO$(N, FP_{prev}, N, VP_{prev})$
$A \leftarrow \frac{A}{P \times N}$
**função** AREA_TRAPEZIO$(X1, X2, Y1, Y2)$
    Base $\leftarrow |X1 - X2|$
    Altura $\leftarrow \frac{Y1+Y2}{2}$
    **retorna** Base $\times$ Altura
**fim função**

### 2.4.2 Logloss

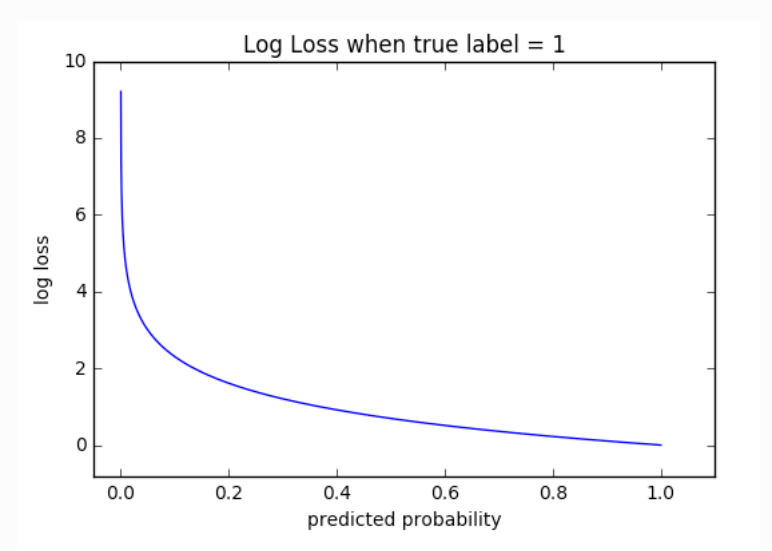
Perda Logarítmica ou **Logloss** é uma função que mede a precisão de um classificador penalizando erros com base na incerteza na previsão, ou seja, verifica se as probabilidades emitidas pelo modelo estão alinhadas com a proporção real de ocorrências do alvo (VOVK, 2015). Nessa métrica, quanto menor o valor melhor: o 0 é a predição perfeita. Conforme veremos no Capítulo 3, essa é a função de perda que é normalmente otimizada pelos algorítimos de boosting.

A equação da *logloss* para classificadores binários é dada por:

$$Logloss = -\frac{1}{N}\sum_{i=1}^{n}[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})] \tag{2.4}$$

O gráfico abaixo mostra a faixa de possíveis valores de perda dada uma observação verdadeira (*target* = 1). À medida que a probabilidade prevista se aproxima de 1, a perda de logarítmica diminui lentamente enquanto que à medida que a probabilidade prevista diminui, a perda de logarítmica aumenta rapidamente. A perda de logarítmica penaliza ambos os tipos de erros, mas especialmente aquelas previsões que não são confiáveis.

Figura 6 – *Logloss* em função das predições de uma observação (GLOSSARY, 2022)




.

### 2.4.3 Teste de Kolmogorov-Smirnov

O Teste de Kolmogorov-Smirnov (KOLMOGOROV, 1933; SMIRNOV, 1948) também é conhecido como teste de KS ou teste K-S. É um teste de aderência: verifica o grau de concordância entre a distribuição de um conjunto de valores (por exemplo, os escores observados pelo modelo) e alguma distribuição teórica. Esse teste pode ser usado para verificar se os dados seguem, por exemplo, a distribuição normal. A distribuição de frequência acumulada é comparada com a distribuição de frequência observada. A distribuição teórica representa o que seria esperado sobre $H_0$ , hipótese nula. É determinado o ponto em que as duas distribuições, teórica e observada, tem a maior diferença. (SIGEL; CASTELLAN, 1988).

Seja $F_0(X)$ uma função de distribuição de frequências relativas acumuladas, a distribuição teórica sob $H_0$, pra qualquer valor de $X$, o valor de $F_0(X)$ e a proporção de casos esperados com escores menores ou iguais a $X$.

Seja $\hat{F}_n$ a distribuição de frequências relativas acumuladas observada de uma amostra aleatória de $N$ observações, se $X_i$ é um escore qualquer, então $\hat{F}(X_i) = \frac{F_i}{N}$ em que $F_i$ é o número de observações menores ou iguais a $X_i$. As hipóteses do testes são $H_0$: a amostra vem de uma

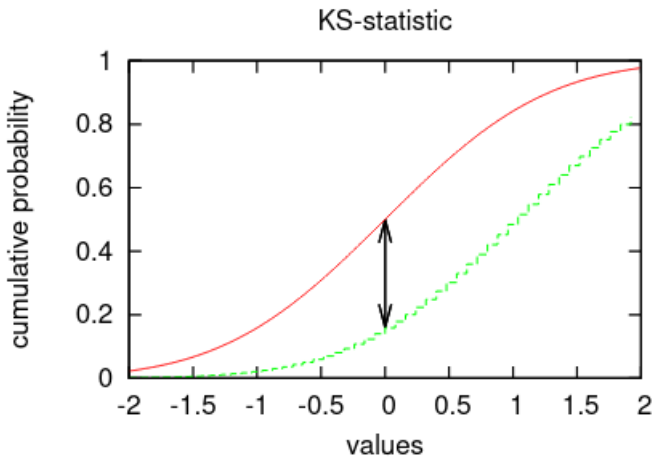
distribuição teórica específica (por exemplo a distribuição normal), ou $H_1$ , a amostra não provém de uma distribuição teórica especifica.

O teste espera que quando $H_0$ é verdadeira as diferenças entre $\hat{F}(X_i)$ e $F_0(X_i)$ sejam pequenas e estando dentro do limite dos erros aleatórios. O maior dos desvios é usado, chamado de desvio máximo

$$D = \max(F_0(X_i) - \hat{F}(X_i)) \tag{2.5}$$

A hipótese é verificada através do teste *p-valor* com $D = \max(F_0(X_i) - \hat{F}(X_i)) < D_{(N,\alpha)}$, onde $D_{(N,\alpha)}$ é um valor tabelado . Nesse caso, não rejeitamos $H_0$ e caso no contrario, $D = \max(F_0(X_i) - \hat{F}(X_i)) > D_{(N,\alpha)}$, rejeitamos $H_0$. (BUSSAB; MORETTIN, 2013).

Figura 7 – KS: diferença entre as funções de distribuição (ASHWIN, 2015).




Para o problema de classificação binária o KS pode ser calculado a partir da *Taxa de Verdadeiros Positivos (TVP)* e da *Taxa de Falso Positivos (TFP)*.

$$KS = \max(TVP - TFP) \tag{2.6}$$

Abaixo temos o algoritmo para o cálculo do KS de *OneSample* (ASHWIN, 2015).

**Algoritmo 3** – KS: OneSample($Q, n, F$)

**Input:** Quantil $Q$, número de dados $n$ e a função de distribuição $F$.
**Output:** $\hat{D}$, o valor estimado de KS.

1: Seja $X_{i1} \leq X_{i2} \leq ... \leq X_{ik}$ valores em $Q$
2: $\hat{D} \leftarrow 0$
3: **se** cada $x \in (X_{i1}, ..., X_{ik})$ **então**
4: $\quad j = \max(p | X_{ip} \leq x)$
5: $\quad$ seja $\hat{i_j}$ aproximadamente o índice de $X_{ij}$
6: $\quad \hat{E}_x = |\frac{\hat{i_j}}{n - F(x)}$
7: $\quad \hat{D} = \max(\hat{D}, \hat{E}_x)$
8: **fim se**
9: **retorna** $\hat{D}$

## 2.5 Hiperparâmetros e Model Tuning

Conforme vimos no Capítulo 1 e será abordado nos próximos Capítulos, muitos modelos de aprendizado supervisionado possuem parâmetros que não podem ser diretamente estimados pelo conjunto de treino. Muitos desses parâmetros afetam a qualidade do modelo prever resultados a partir de novos dados de entrada. Esses parâmetros são intrínsecos dos modelos pois representam características deles, como o número de interações, o termo de regularização, o tamanho da árvore e etc. Esses parâmetros precisam ser escolhidos antes do início do processo de aprendizado do algoritmo, e são denominados de hiperparâmetros e o processo de escolha desses parâmetros visando a melhoria da performance do modelo é conhecido como *tuning hyperparameter* (GOODFELLOW; BENGIO; COURVILLE, 2016; KUHN; JOHNSON, 2013; PROBST; BISCHL; BOULESTEIX, 2018; YANG; SHAMI, 2020; HUTTER; KOTTHOFF; VANSCHOREN, 2019; ZHENG, 2022).

Para lidar com os hiperparâmetros existem duas abordagens: escolher eles manualmente ou utilizar um algoritmo para escolher automaticamente. A escolha manual requer o entendimento profundo da influência dos hiperparâmetros e como os modelos de aprendizagem de máquina alcançam uma boa generalização. Geralmente consiste em retreinar o modelo varias vezes com apenas a alteração dos hiperparâmetros e validando as performance de cada treino. Por outro lado, a seleção automática de hiperparâmetros reduz a necessidade de entender a fundo sua influência, porém requer um maior poder de processamento computacional.

Idealmente, um algoritmo de aprendizagem usaria apenas o conjunto de dados e geraria uma função, não exigindo ajuste manual de hiperparâmetros. A popularidade de vários algoritmos como regressão logística e SVMs decorre, em parte, de sua capacidade de trazer resultados satisfatórios com apenas um ou dois hiperparâmetros ajustados. As redes neurais podem também funcionar bem com apenas um pequeno número de hiperparâmetros ajustados, mas muitas vezes se beneficiam significativamente do ajuste de quarenta ou mais hiperparâmetros. O ajuste de

hiperparâmetros pode funcionar muito bem quando o usuário tem um bom ponto de partida, como aquele determinado por outros que trabalharam no mesmo tipo de aplicativo e arquitetura, ou quando o usuário tem meses ou anos de experiência em explorar valores de hiperparâmetros para redes neurais aplicadas a tarefas semelhantes. No entanto, para muitas aplicações, esses pontos de partida não estão disponíveis. Nesses casos, algoritmos automatizados podem encontrar valores úteis dos hiperparâmetros(MANTOVANI *et al.*, 2018).

Estamos tentando encontrar um valor dos hiperparâmetros que otimize uma função objetivo, como o erro de validação, às vezes sob restrições. Portanto, é possível, em princípio, desenvolver algoritmos de otimização de hiperparâmetros que envolvam um algoritmo de aprendizado de máquina e escolha seus hiperparâmetros, ocultando do usuário do algoritmo de aprendizado o ajuste dos hiperparâmetros. Infelizmente, a otimização de hiperparâmetros dos algoritmos geralmente têm seus próprios hiperparâmetros, como o intervalo de valores que deve ser explorado para cada um dos hiperparâmetros do algoritmo de aprendizado de máquina. No entanto, esses hiperparâmetros secundários são geralmente mais fáceis de escolher, no sentido de que um desempenho aceitável pode ser alcançado em uma ampla gama de tarefas usando os mesmos hiperparâmetros secundários para todas as tarefas.

O processo de tunar os hiperparâmetros geralmente é tratado como uma problema de otimização caixa-preta. De maneira formal, podemos definir o problema como: Seja:

$$\mathcal{H} = \mathcal{H}_1 \times \mathcal{H}_2 \times ... \times \mathcal{H}_k$$

O espaço dos hiperparâmetros e um algoritmo $a \in \mathcal{A}$ em que $\mathcal{A}$ é um conjunto de algoritmos de aprendizado de máquina. Cada $\mathcal{H}_i$ representa um conjunto de possíveis valores para o hiperparâmetro $i^{th}$ de $a(i \in \{1,...,k\})$. Seja $\mathcal{D}$ um conjunto de dados em que $D \in \mathcal{D}$ é um conjunto de dados de $\mathcal{D}$. A função $f : \mathcal{A} \times \mathcal{D} \times \mathcal{H} \to \mathbb{R}$ calcula a performance do modelo pelo algoritmo $a \in \mathcal{A}$ no conjunto de dados $D \in \mathcal{D}$ dado a configuração de hiperparâmetros $h = (h_1, h_2, ..., h_k) \in \mathcal{H}$.

Dado $a \in \mathcal{A}, \mathcal{H}$ e $D \in \mathcal{D}$ o objetivo da tunar os hiperparâmetros é encontrar $h^* = (h_1^*, h_2^*, ..., h_k^*)$ tal que :

$$h^* = arg\max_{h \in \mathcal{H}} f(a, D, h) \tag{2.7}$$

Existem diversas técnicas que buscam otimizar os hiperparâmetros, irei exemplificar algumas, para um estudo mais completo consultar (YANG; SHAMI, 2020; HUTTER; KOTTHOFF; VANSCHOREN, 2019; ZHENG, 2022).

### 2.5.1 Grid Search

É um dos métodos mais comuns utilizado para explorar o espaço de configuração de hiperparâmetros é o Grid Search. Ele pode ser considerado uma busca exaustiva ou um método de força bruta que avalia todas as combinações de hiperparâmetros dadas à uma 'grade' de

configurações. O Grid Search funciona avaliando o produto cartesiano de um conjunto finito de valores especificado pelo usuário, ele não pode explorar as regiões de bom desempenho por si só. Portanto, para identificar os ótimos globais, o seguinte procedimento precisa ser realizado manualmente (ELSHAWI; MAHER; SAKR, 2019; HUTTER; KOTTHOFF; VANSCHOREN, 2019; ZÖLLER; HUBER, 2019).

1. Comece com um grande espaço de busca e tamanho de passo definidos.
2. Restrinja o espaço de pesquisa e o tamanho do passo com base nos resultados anteriores de hiperparâmetros.
3. Repita a etapa 2 várias vezes até que um ótimo seja alcançado.

O Grid Search pode ser facilmente implementado e paralelizado. No entanto, a sua principal desvantagem é a sua ineficiência para alta dimensionalidade de espaço de configuração de hiperparâmetros, já que o número de interações aumenta exponencialmente à medida que o número de hiperparâmetros cresce, esse crescimento exponencial é conhecido como a maldição da dimensionalidade. Para Grid Search, assumindo que existem $k$ parâmetros, e cada um deles com $n$ valores distintos, sua complexidade computacional é de $\mathcal{O}(n^k)$. Logo, o Grid Search só se torna um algorítimo eficiente quando o espaço de configurações de hiperparâmetros é pequeno.

### 2.5.2 Random Search

Para superar certas limitações do Grid Search, o Random Search foi proposto (JAMES, 2012; HUTTER; KOTTHOFF; VANSCHOREN, 2019; ZÖLLER; HUBER, 2019). Ele é bem semelhante ao Grid Search mas ao invés de testar todos os valores no espaço de configuração ele seleciona aleatoriamente um número pré definido de amostrar entre os limites superior e inferior e em seguida treina essas amostras. Ele é capaz de explorar um espaço de busca maior que o Grid Search, ele é facilmente paralelizado pois cada avaliação é independente. Como o número total de avaliações no Random Search é definido para um valor fixo $n$ antes do início do processo de otimização, a complexidade computacional é $\mathcal{O}(\backslash)$. Além disso, Random Search pode detectar o ótimo global ou um ótimo local quando fornecido poder computacional o suficiente.

A principal limitação desses dois métodos é que cada avaliação é independente da avaliação anterior, ou seja, eles perdem muito tempo avaliando combinações ruins de hiperparâmetros.

### 2.5.3 Otimização Bayesiana

Para resolver a limitação do Random Search e Grid Search, temos os algoritmos de otimização Bayesiana, que são algorítimos interativos que determinam o ponto de avaliação futura com base no resultado obtido na interação passada. Para determinar a próxima configuração de hiperparâmetro, o algoritmo usa dois componentes principais: um modelo substituto e uma

função de aquisição (ou seleção). O modelo substituto (processo gaussiano) visa atender a todos os pontos observados na função objetivo. Depois de obter a distribuição posterior de funções que melhor descreve a função que você deseja otimizar a função de aquisição determina o uso de diferentes pontos para equilibrar o trade-off entre *exploration* e *exploitation* (HUTTER; KOTTHOFF; VANSCHOREN, 2019; SNOEK; LAROCHELLE; ADAMS, 2012). *Exploration* consiste em amostrar as instâncias nas áreas que não foram amostrados, enquanto *exploitation* é para amostrar nas atuais regiões promissoras onde o ótimo global é mais provável, com base na distribuição posterior. Os modelos de otimização Bayesiana equilibram ambos os processos para detectar as atuais regiões mais prováveis e evitar a perda de melhores configurações nas áreas inexploradas. Os procedimentos do algoritmo são:

1. Construir um modelo substituto probabilístico da função objetivo.
2. Detecte os valores ideais de hiperparâmetros no modelo substituto.
3. Aplique esses valores de hiperparâmetros à função objetiva para avaliação.
4. Atualize o modelo substituto com novos resultados.
5. Repita as etapas 2 a 4 até que o número máximo de iterações seja alcançado.

Assim, o algoritmo de otimização Bayesiana funciona atualizando o modelo substituto após cada avaliação na função objetivo. A otimização Bayesiana é mais eficiente que Grid Search e o Random Search, pois pode detectar a combinação ideal de hiperparâmetros analisando os valores testados anteriormente.

## 2.6 Árvore de Decisão

A maioria dos algoritmos de boosting utilizam árvores de decisão como aprendizes básicos. Uma árvore é uma coleção de elementos chamados de nós, dentre os quais um é distinguido como uma raiz, juntamente com uma relação de "paternidade" que impõe uma estrutura hierárquica sobre os nós.

Uma Árvore de Decisão é:

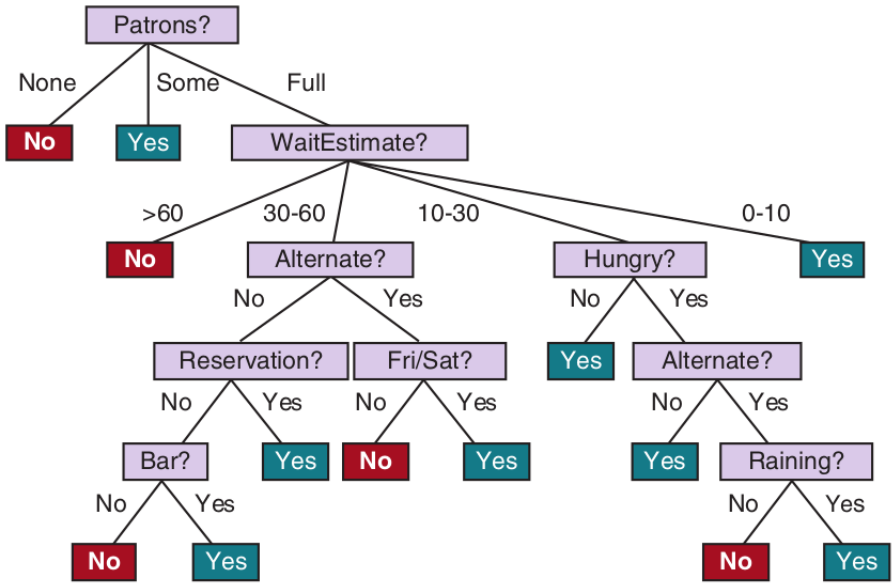
- um nó folha (ou nó resposta) que contém o nome de uma classe ou o símbolo nulo (nulo indica que não é possível atribuir nenhuma classe ao nó por não haver nenhum exemplo que corresponda a esse nó)
- um nó interno (ou nó de decisão) que contém o nome de um atributo; para cada possível valor do atributo, corresponde um ramo para uma outra árvore de decisão.

E temos uma estrutura típica para uma árvore de decisão:

- Nós internos são rotulados com atributos.
- Folhas são rotuladas com classes
- Ramos são rotulados com valores (atributos categóricos) ou com intervalos (atributos numéricos).

Uma árvore de decisão é uma representação de uma função que mapeia um vetor de valores de atributos para um único valor de saída, uma "decisão". Uma árvore de decisão chega à sua decisão realizando uma sequencia de testes, começando na raiz e seguindo o ramo apropriado até que uma folha seja alcançada. Cada nó interno na árvore corresponde a um teste do valor de uma das entradas atributos, os ramos do nó são rotulados com os possíveis valores do atributo, e os nós folha especificam qual valor deve ser retornado pela função. O processo de aprendizado de uma árvore de decisão é chamado de indução da árvore de decisão. A ideia do algoritmo é escolher as divisões internas da árvore que melhor explicam os dados e é, em si, uma pequena árvore. Na figura 8 temos um exemplo de uma aŕvore de decisão treinada (RUSSELL; NORVIG, 2021).

Figura 8 – Exemplo de uma árvore de decisão (RUSSELL; NORVIG, 2021).




O processo de aprendizado das árvores de classificação consiste em várias iterações para escolher as melhores divisões a partir dos dados de treinamento fornecidos, de acordo com critérios. Os critérios mais comuns de otimização são geralmente advindos da Teoria da Informação, como o Índice de Gini e a Entropia Cruzada. O índice de Gini, $G(D)$ no nó $S$ na árvore de decisão pode ser definido como:

$$G(S) = 1 - \sum_{i=1}^{c} p_i^2$$

O algoritmo 4 mostra o pseudo código para o algoritmo de árvore de decisão (ZHANGA; CHIA; ZHANGA, 2018).

**Algoritmo 4** – Árvore de decisão.

**Input:** Conjunto de treino $D = \mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(i)}, y^{(j)}$. Atributos $A = (a_1, a_2, ...a_k)$. Método de seleção do atributo: Utilizaremos o *Gain Ratio* para separar o atributo e o sub conjunto de dados $GR(D,a) = \frac{G(D,a)}{Entropia(a)}$, em que $Entropia(D) = \sum_{i=1}^{c} -p_i log_2 p_i$ e o $G(D,a) = Entropia(D) - \sum_{v \in Values(a)} \frac{|D_v|}{D} Entropia(D_v)$, em que $Values(a)$ são todos os possíveis valores do atributo $a$ e $D_v$ é um sub conjunto do nosso conjunto de dados $D$ que para cada atributo tem o valor $v$.
**Output:** Árvore de decisão.

criar o nó $N$
**se** as amostras de $D$ estão na mesma classe $C$ **então**
    **retorna** $N$ como nó folha rotulado com a classe $C$
**fim se**
**se** $A \neq 0$ ou o valor do atributo em $D$ são o mesmo **então**
    **retorna** $N$ como nó folha rotulado com a classe em $D$
**fim se**
ache o melhor atributo $a \in A$ utilizando um método de seleção de atributo

**para** cada valor $a^y$ de $a_*$ **faça**
    **se** $D_v \neq 0$ **então**
        colocar a folha com a maioria das classes em $D$ para o nó $N$
    **senão**
        colocar o nó retornado pelo $TreeGenerate(D_V, A)$ para o nó $N$
    **fim se**
**fim para**

Hiperparâmetros típicos de árvores de classificação referem-se à estrutura real da árvore. Como por exemplo o hiperparâmetro de profundidade máxima controla a profundidade que a árvore pode crescer, ou seja, quantas divisões serão tomadas, amostras mínimas de folhas controla o número mínimo de amostras que cada folha precisa ter. A tabela 2 abaixo mostra alguns exemplos de hiperparâmetros de modelos de árvore de decisão (MANTOVANI *et al.*, 2018) (GOODFELLOW; BENGIO; COURVILLE, 2016).

Tabela 2 – Exemplo de hiperparâmetros em árvore de decisão(PROBST; BISCHL; BOULESTEIX, 2018)(GOODFELLOW; BENGIO; COURVILLE, 2016).

| **Hiperparâmetro** | **Descrição** | **Tipo da variável** |
|---|---|---|
| *min_samples_split* | O número mínimo de amostras necessárias para dividir um nó interno. | int ou float |
| *max_depth* | A profundidade máxima da árvore. Se Nenhum, os nós serão expandidos até que todas as folhas sejam puras ou até que todas as folhas contenham menos que min_samples_split amostras. | int |
| *splitter* | A estratégia usada para escolher a divisão em cada nó. As estratégias suportadas são “melhores” para escolher a melhor divisão e “aleatórias” para escolher aleatoriamente. | string |
| *min_weight_fraction_leaf* | A estratégia usada para escolher a divisão em cada nó. As estratégias suportadas são “melhores” para escolher a melhor divisão e “aleatórias” para escolher aleatoriamente. | string |
| *min_samples_leaf* | A fração ponderada mínima da soma total dos pesos (de todas as amostras de entrada) necessária para estar em um nó folha. | float |
| *max_features* | O número de features a serem considerados ao procurar a melhor divisão. | int ou float |
| *random_state* | Gerador de números aleatórios. | int |
| *min_impurity_decrease* | Um nó será dividido se esta divisão induzir uma diminuição da impureza maior ou igual a este valor.. | float |
| *class_weight* | Pesos associados a classes no formato (class_label: weight). Se não for fornecido, todas as classes devem ter peso um. Para problemas de múltiplas saídas, uma lista de dicionários pode ser fornecida na mesma ordem das colunas de *y*. | dicionário, lista de dicionário, “balanced” ou None |

CAPÍTULO

# 3

# ALGORITMOS DE GRADIENT BOOSTING

*Gradient Boosting Machines* (GBM) é um algoritmo de aprendizado de máquina, baseado na ideia de modelos aditivos de estatística e descida de gradiente. O GBM funciona construindo um modelo aditivo avançado *stagewise* realizando gradiente descendente no espaço de função, conforme proposto por (FRIEDMAN, 2001). O *Gradient Boosting* é um algoritmo de aprendizado de máquina amplamente utilizado devido à sua eficiência, precisão e interpretabilidade.

## 3.1 Additive Model

Um modelo aditivo é uma técnica de regressão onde a ideia básica é aproximar um conjunto de dados usando uma soma de funções suaves do indivíduo características dos dados observados, em vez de uma superfície de regressão geral complexa (FRIEDMAN; STUETZLE, 1981).

Considere o conjunto de dados $\{(\mathbf{x}^{(1)},y^{(1)}),(\mathbf{x}^{(2)},y^{(2)}),...,(\mathbf{x}^{(i)},y^{(i)})\}$, em que $\mathbf{x}$ são os preditores e $y$ a saída, o modelo aditivo pode ser escrito da seguinte forma

$$E|y^{(i)}|x^{(1)},x^{(2)},...,x^{(i)}| = \beta_0 + \sum_{j=1}^{p} f_j(x^{(ij)}) \tag{3.1}$$

Ou

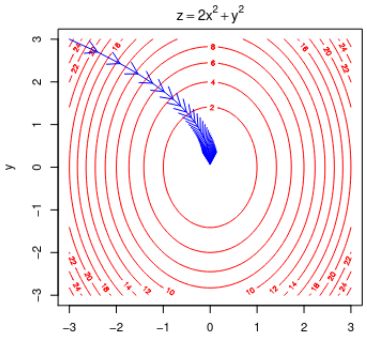
$$Y = \beta_0 + \sum_{j=1}^{p} f_j(\mathbf{X}^{(:,j)}) + \varepsilon \tag{3.2}$$

As funções $f_j(x^{(ij)}$ são conhecidas como funções de suavização, ou seja, ao invés de utilizar uma regra complexa de regressão, usamos a soma de funções de suavização.

## 3.2 Gradient Descent

Podemos separar os métodos de solução de problemas de otimização entre determinísticos e métodos estocásticos. Abordagens determinísticas são geralmente mais simples que as estocásticas, porém, o risco de ficar preso em um mínimo local é significativamente maior. O gradiente descendente é um método estocásticos.

Figura 9 – Exemplo de otimização pelo gradiente descendente.




A versão básica do algoritmo tem como objetivo minimizar uma função objetiva $F(\theta)$. Esse algorítimo tem uma taxa de aprendizado $\eta$ para controlar o quanto os coeficientes de $\theta$ podem mudar em cada interação (RUSSELL; NORVIG, 2021; HASTIE; TIBSHIRANI; FRIEDMAN, 2009; GOODFELLOW; BENGIO; COURVILLE, 2016; RUDER, 2016; E.; MCCLELLAND; GROUP, 1986; NESTEROV, 2004).

Podemos escrever o gradiente descendente como:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla\theta_{t-1} F(\theta_{t-1}; \mathbf{x}^{(i)}, y^{(i)}) \tag{3.3}$$

Uma execução de descida de gradiente executará a atualização acima $t = M$ vezes e pode ser interpretada como $M$,a atualização, gerando vetores $v_m$ da forma $v_m = -\eta \cdot \nabla\theta_{t-1} F(\theta_{t-1}; \mathbf{x}^{(i)}, y^{(i)})$. Denotando $v_0 = \theta_0$, os valores iniciais dos parâmetros antes da otimização, os parâmetros finais podem ser escritos como:

$$\theta^* = \sum_{m=0}^{M-1} v_m \tag{3.4}$$

## 3.3 Boosting

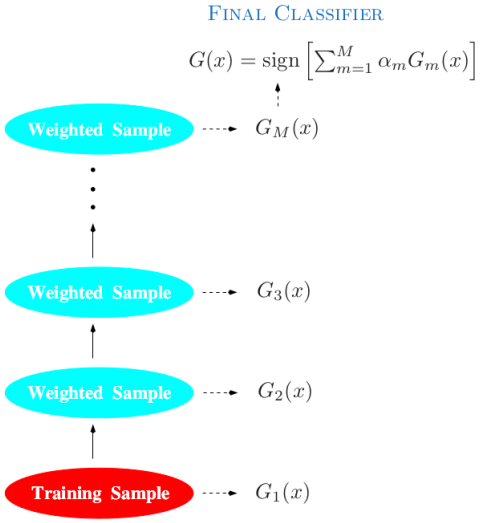
*Boosting* é uma das ideias de aprendizado mais poderosas introduzidas nas últimas décadas. Ele foi originalmente projetado para problemas de classificação mas pode ser utilizado para problemas de regressão. A ideia por tras do *boosting* é combinar a saídas de muitos classificadores “fracos” para produzir um “comitê” poderoso, ou seja, criar um aprendiz "poderoso"a partir de uma combinação de aprendizes "fracos"(SCHAPIRE, 1999; HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

Um classificador fraco é aquele cuja taxa de erro é apenas ligeiramente melhor do que uma adivinhação aleatória. O objetivo do *boosting* é aplicar sequencialmente o algoritmo de classificação "fraco"para várias versões, produzindo assim uma sequência de classificadores fracos $G_m(x), m = 1,2,3,...,M$. As previsões de todos eles são combinadas por meio de uma ponderação para produzir a previsão final:

$$G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x)) \tag{3.5}$$

Em que $\alpha_1, \alpha_2, \alpha_M$ são calculados pelo algoritmo de boosting e cada contribuição segue um peso em $G_m(x)$. Esse algoritmo mostrado pela equação acima é conhecido como AdaBoost (E.; YOAV, 1999). Abaixo temos um esquemático de como os classificadores são treinados na versão com peso do conjunto de dados e depois combinado com a predição final.

Figura 10 – Esquemático do algoritmo AdaBoost (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).




O algoritmo 5 mostra o detalhe do algoritmo de AdaBoost (E.; YOAV, 1999; HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

**Algoritmo 5** – AdaBoost M1

1: Iniciar as observações com pesos $w_i = 1/N, i = 1,2,3,...,N$
2:
3: **para** $m = 1$ até $M$: **faça**
4: Fit o classificador $G_m(x)$ no conjunto de treino utilizando os pesos $w_i$
5: calcule o erro $err_m = \frac{\sum_{i=1}^{N} w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
6: Calcule $\alpha_m = log((1 - err_m)/err_m)$
7: $w_i \rightarrow w_i \cdot exp[\alpha_m \cdot \mathbf{1}(y_i \neq G_m(x_i))], i = 1,2,...,N$
8: **fim para**
9: **retorna** $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$

## 3.4 GBMs

*Gradient Boosting Machine* utiliza o conceito de modelo aditivo e é uma combinação do gradiente descente com o *boosting* (FRIEDMAN, 2001). O algoritmo GBM funciona otimizando qualquer função de perda diferenciável, usando Gradiente descendente. No entanto, a otimização não é feita em termos de otimização numérica (ou seja, atualizando um vetor de parâmetros $\theta$), mas por funções de "bossting"na direção do gradiente da função de perda. Como os GBMs lidam com dados finitos, eles otimizam as funções em termos do conjunto de dados supervisionado, nosso conjunto $(\mathbf{x}^{(i)}, y^{(i)})$, o modelo final de GBM será:

$$F_M(x) = F_0(x) + \sum_{m=1}^{M} F_m(x) \tag{3.6}$$

As funções $F_m(x)$ são funções construídas de forma *stagewise*, assim como o $\theta_t$ em gradiente descendente. As funções base são aprendizes, e podem ser parametrizadas como $\beta_m h(\mathbf{x}; a_m)$, onde $\beta_m$ é um peso, e $\alpha_m$ os parâmetros do aprendiz $h$. Na maioria das implementações, as funções básicas são aprendizes baseados em árvore, mas podem ser qualquer aprendiz em que seja seja possível atribuir pesos. Também, dada uma função de perda $L(y_i, F_m(x_i))$, gostaríamos de encontrar todos os valores ótimos de $\beta_m$ e $\alpha_m$ que minimizam a função perda, ou seja:

$$\{\beta_m, \alpha_m\}_1^M = \arg\min_{\{\beta'_m, \alpha'_m\}_1^M} \sum_{i=1}^{n} L\left(y^{(i)}, \sum_{m=1}^{M} \beta'_m h(\mathbf{x}^{(i)}; \alpha'_m)\right)$$

No entanto, na maioria das situações, a otimização acima é inviável, portanto, a abordagem *"greedy-stagewise"* é otimizar cada par $\beta_m$ e $\alpha_m$ em um modelo *stagewise*, ou seja, para cada $m = 1, 2, ..., M$.

$$(\beta_m, \alpha_m) = \arg\min_{\beta,\alpha} \sum_{i=1}^{n} L\left(y^{(i)}, F_{m-1}\mathbf{x}^{(i)} + \beta h(\mathbf{x}^{(i)}; \alpha)\right) \tag{3.7}$$

Utilizando a notação vetorizada

$$F_m(\mathbf{X}) = F_{m-1}(\mathbf{X}) + \eta \Delta_m(X) \tag{3.8}$$

$\beta_m h(\mathbf{x}^{(i)}; \alpha_m)$ pode ser interpretado como o melhor passo "greedy"em direção à estimativa, $F^*(x)$, esta pode ser vista como uma atualização do método do gradiente descendente. O análogo de $\nabla\theta_{t-1}$ no gradiente descente numérico é o gradiente da função de perda $L$ com relação à última estimativa $F_{m1}(x)$ (FRIEDMAN, 2001).

$$-g_m(\mathbf{x}^{(i)}) = -\left[\frac{\partial L(y^{(i)}, c^{(i)})}{(\mathbf{x}^{(i)})}\right] \tag{3.9}$$

Em que $F(x) = F_{m-1}(x)$.

Na literatura, esse gradiente da função de perda $L$ em relação à última previsão $\hat{y}_{m1}$. Essa última previsão às vezes é chamada de pseudo-residual e definida como $\mathbf{r}_{m-1}$.

$$\mathbf{r}_{m_1} = \nabla F_{m-1}(\mathbf{X}) L(y, F_{m-1}(\mathbf{X})) = \nabla \hat{y}_{m-1} L(y, \hat{y}_{\mathbf{m-1}}) \tag{3.10}$$

O algoritmo GBM ajusta um aprendiz $h(x; \alpha_m)$ com peso $\beta$ usando os pseudo-resíduos, não o $\mathbf{X}$ original. A versão final do algoritmo é definida como:

(HASTIE; TIBSHIRANI; FRIEDMAN, 2009; FRIEDMAN, 2001).

**Algoritmo 6** – Gradient Boost($\mathbf{X}, y, M, \eta$)

$F_0(\mathbf{X} = \arg\min_v \sum_{i=1}^{n} L(y^{(i)}, v)$
**para** $m = 1$ até $M$: **faça**
  $\mathbf{r}_{m_1} = \nabla \hat{y}_{m-1} L(y, \hat{y}_{\mathbf{m-1}})$ ▷ Treinar o aprendiz base para minimizar o erro quadrático
  $\alpha = \arg\min_{\alpha,\beta} \sum_{i=1}^{n} (\mathbf{r}_{m-1}^{(i)} - \beta h(\mathbf{x}^{(i)}; \alpha))^2$
  $\beta = \arg\min_{\beta} \sum_{i=1}^{n} L(y^{(i)}, F_{m-1}(\mathbf{x}^{(i)}) + \beta h(\mathbf{x}^{(i)}); \alpha_m)$
  $\Delta_m(X) = \beta_m h(\mathbf{X}; \alpha_m)$
  $F_m(\mathbf{X}) = F_{m-1}(\mathbf{X}) + \eta \Delta_m(X)$
**fim para**
**retorna** $F_m$

## 3.5 XGBoost, CatBoost e LightGBM

Os três algoritmos no escopo do trabalho: XGBoost (*Xtreme Gradient Boosting*), CatBoost (*Category Boosting*) e LightGBM (*Light Gradient Boosting Machine*) são todos variantes de algoritmos de *Gradient Boosting*. Todos podem ser utilizados como Regressor (prevendo variáveis contínuas) ou um Classificador (prevendo variáveis categóricas).

XGBoost é uma solução altamente escalável, flexível e versátil, foi projetado para explorar recursos corretamente e superar as limitações do aumento de gradiente anterior. A principal diferença entre XGBoost e outros algoritmos de gradiente é que ele usa uma nova técnica de regularização para controlar o overfitting. Portanto, é mais rápido e mais robusto durante o ajuste do modelo. A técnica de regularização é feita adicionando um novo termo a função de perda, como:

$$L(\phi) = \sum_{i=1}^{n} L(\hat{y}_i, y_i) + \sum_{m=1}^{M} \Omega(f_k) \tag{3.11}$$

Em que :

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda ||w||^2 \tag{3.12}$$

XGBoost usa um novo ganho

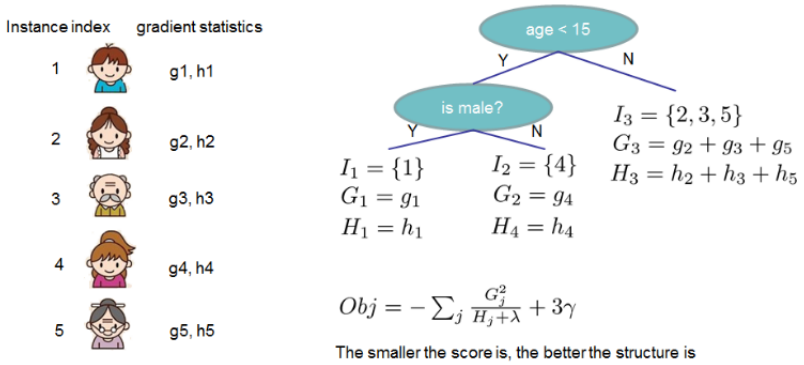
$$Gain = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in R_L} g_i)^2}{\sum_{i\in R_L} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{3.13}$$

Em que

$$g_i = \partial_{\hat{y}_i} L(\hat{y}_i, y_i) h_i = \partial^2_{\hat{y}_i} L(\hat{y}_i, y_i) \tag{3.14}$$

Assuma que $I_L$ e $I_R$ são os conjuntos de instâncias da esquerda e nós direitos após a divisão e $I = I_L \cup I_R$ (CHEN; GUESTRIN, 2016; CHEN; GUESTRIN, ).

Figura 11 – Exemplo do cálculo da estrutura do cálculo do score para uma árvore (CHEN; GUESTRIN, 2016).




CatBoost (para "impulsionamento categórico") concentra-se em colunas categóricas usando técnicas de permutação, one_hot_max_size (OHMS) e estatísticas baseadas em alvo. CatBoost resolve o crescimento exponencial da combinação de recursos usando o método greedy em cada nova divisão da árvore atual. Além disso a construção é feita por arvores simétricas e o algoritmo suporta colunas categóricas (DOROGUSH *et al.*, 2017; DOROGUSH *et al.*, ).

O LightGBM é parecido com o XGBoost com algumas alterações: uma nova técnica para estimar o ganho de informação chamada *gradient-based one-side sampling* (GOSS). Uma vez que uma das tarefas mais demoradas no processo de aprendizagem no aumento de gradiente é encontrar a divisão para as árvores, geralmente algum tipo de amostragem é feito nesta etapa para fins de eficiência (KE *et al.*, 2017; MICROSOFT, ).

Temos abaixo um comparativo com maiores detalhes de cada algoritmo.

### 3.5.1 Divisões de nó

Antes de começar o aprendizado os algortimos precisam criar pares de divisões de recursos, por exemplo (idade,$<15$), (idade,$>20$), (quantidade, >1000). Esses pares de divisão de recurso são construídos com base em histograma e são usados durante o processo de aprendizado

como possíveis divisões de nó. Esse método de pré-processamento é mais rápido do que o algoritmo *greedy*, que enumera linearmente todas as divisões possíveis para recursos contínuos.

O **XGboost** não utiliza nenhuma técnica de amostragem ponderada, ele utiliza algoritmos puramente baseados em histogramas o que torna seu processo de divisão mais lento em comparação com GOSS e MVS.

O **Catboost** oferece uma nova técnica chamada *Minimal Variance Sampling* (MVS), que é uma versão de amostragem ponderada do *Stochastic Gradient Boosting*. Nesta técnica, a amostragem ponderada ocorre no nível da árvore e não no nível da divisão. As observações para cada árvore de reforço são amostradas de forma a maximizar a precisão da pontuação dividida.

O **LightGBM** oferece amostragem unilateral baseada em gradiente (GOSS) que seleciona a divisão usando todas as instâncias com grandes gradientes (ou seja, grande erro) e uma amostra aleatória de instâncias com pequenos gradientes. Para manter a mesma distribuição de dados ao calcular o ganho de informação, o GOSS introduz um multiplicador constante para as instâncias de dados com pequenos gradientes. Assim, o GOSS consegue um bom equilíbrio entre aumentar a velocidade, reduzindo o número de instâncias de dados, e manter a precisão das árvores de decisão aprendidas.

### 3.5.2 Crescimento da Árvore

O **XGboost** divide as árvores até o hiperparâmetro max_depth especificado e logo em seguida começa a podar a árvore para trás e remove as divisões além das quais não há ganho positivo. Ele usa essa abordagem, pois às vezes uma divisão sem redução de perda pode ser seguida por uma divisão com redução de perda. O XGBoost também pode executar o crescimento da árvore folha a folha (como LightGBM).

**Catboost** cresce uma árvore equilibrada. Em cada nível dessa árvore, o par de divisão de recursos que traz a menor perda (de acordo com uma função de penalidade) é selecionado e é usado para todos os nós do nível. É possível alterar sua política usando o parâmetro grow_policy.

O **LightGBM** usa crescimento de árvore folha a folha (melhor primeiro). Opta por cultivar a folha que minimiza a perda, permitindo um crescimento de uma árvore desequilibrada. Como não cresce em nível, mas em folha, o *overfitting* pode acontecer quando os dados são pequenos. Nesses casos, é importante controlar a profundidade da árvore.

A figura 12 mostra o comparativo do crescimento de árvore de cada algoritmo.

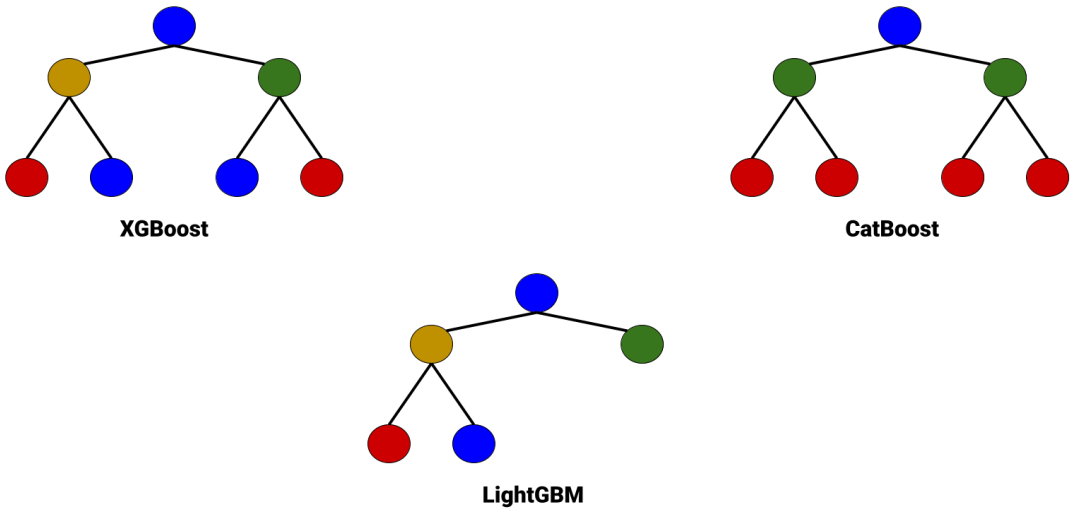



Figura 12 – Comparativo do crescimento da Árvore no XGBoost, CatBoost e LightGBM

### 3.5.3 Valores Missing

No **XGboost** e **LightGBM**, os valores *missing* serão alocados para o lado que reduz a perda em cada divisão. Enquanto o **Catboost** tem dois modos para processar valores ausentes, "Min" e "Max". Em "Min", os valores ausentes são processados como o valor mínimo para um recurso (eles recebem um valor menor que todos os valores existentes). Dessa forma, é garantido que uma divisão que separa os valores ausentes de todos os outros valores seja considerada ao selecionar as divisões. "Max" funciona exatamente da mesma forma que "Min", apenas com valores máximos.

### 3.5.4 Feature Importance

O **XGboost** e o **LightGBM** têm dois métodos semelhantes: O primeiro é "Ganho", que foi explicado na equação 3.13. É a melhoria na precisão (ou ganho total) trazida por um recurso para os ramos em que está. O segundo método tem um nome diferente em cada pacote: *split* (LightGBM) e *Frequency/Weight* (XGBoost). Este método calcula o número relativo de vezes que um determinado recurso ocorre em todas as divisões das árvores do modelo. Esse método pode ser influenciado por recursos categóricos com um grande número de categorias.

O **XGboost** possui mais um método, *Coverage*, que é o número relativo de observações relacionadas a uma feição. Para cada recurso, contamos o número de observações usadas para decidir o nó folha.

**Catboost** tem dois métodos: O primeiro é *PredictionValuesChange*. Para cada recurso, *PredictionValuesChange* mostra quanto, em média, a previsão muda se o valor do recurso mudar. Uma característica teria uma importância maior quando uma mudança no valor da característica causa uma grande mudança no valor previsto. Este é o método de cálculo de importância de

recurso padrão para métricas não classificadas. O segundo método é *LossFunctionChange*. Esse tipo de importância de recurso pode ser usado para qualquer modelo, mas é particularmente útil para modelos de classificação. Para cada característica o valor representa a diferença entre o valor de perda do modelo com esta característica e sem ela. Como é computacionalmente caro retreinar o modelo sem um dos recursos, esse modelo é construído aproximadamente usando o modelo original com esse recurso removido de todas as árvores do conjunto. O cálculo dessa importância de recurso requer um conjunto de dados.

### 3.5.5 Variáveis Categóricas

O **XGboost** não possui um método embutido para recursos categóricos. A codificação (*one-hot-encoding*, codificação de destino, etc.) deve ser realizada pelo usuário.

**Catboost** usa uma combinação de codificação one-hot e uma codificação média avançada. Para recursos com baixo número de categorias, ele usa *one-hot-encoding*. O número máximo de categorias para *one-hot-encoding* pode ser controlado pelo parâmetro one_hot_max_size. Para as colunas categóricas restantes, CatBoost usa um método eficiente de codificação, que é semelhante à codificação média, mas com um mecanismo adicional destinado a reduzir o *overfitting*.

**LightGBM** divide recursos categóricos particionando suas categorias em 2 subconjuntos. A ideia básica é classificar as categorias de acordo com o objetivo do treinamento em cada divisão, o valor final que vai ser treinado no modelo tem que ser um valor númerico.

A Tabela 3 resume essas principais diferenças entre os algoritmos.

Tabela 3 – Principais características do XGBoost, CatBoost e LightGBM

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| **Desenvolvedor** | DMLC | Yandex | Microsft |
| **Ano de Release** | 2014 | 2017 | 2016 |
| **Simetria da Árvore** | Assimétrica Level-wise | Simétrica | Assimétrica Leaf-wise |
| **Método de Splitting** | Algoritmos de histogramas | Greedy | GOSS |
| **Colunas Numéricas** | Suporta | Suporta | Suporta |
| **Colunas Categóricas** | Não Suporta (conveter utilizando one-hot-encoding) | Suporta | Não Suporta (converter para numérico ou ordinal) |
| **Colunas de Texto** | Não Suporta | Suporta | Não Suporta |
| **Valores missing** | Interpreta como NaN ou zero | Interpreta como NaN ou zero | Interpreta como NaN ou zero |

## 3.6 Hiperparâmetros

Conforme vimos na seção anterior, uma das principais diferença entre o XGBoost e CatBoost para o LGBM é que o LGBM cultiva folhas de árvores, enquanto que os outros algoritmos utilizam uma abordagem de profundidade. Isso impacta em como cada valor de hiperparâmetro deve ser escolhido e quais valores devem ser otimizados e estudados. Cada algoritmo pode ter mais de 20 hiperparâmetros. Abaixo temos os hiperparâmetros mais comuns e alguns deles serão analisados na primeira parte desse estudo. Na segunda parte, ao utilizar o Optuna para tunar o modelo podemos inserir mais hiperparâmetros.

- **learning_rate:** Taxa de aprendizado $\eta$, é o parâmetro que discutimos em 3.3, ele é responsável por controlar a influencia de cada novo aprendiz, ou seja, ele basicamente determina o tamanho do passo de cada interação enquanto se move em direção a um mínimo da função de perda.

- **num_leaves:**Este hiperparâmetro controla o número máximo de folhas a crescer em cada iteração, e é a principal forma de controlar a complexidade do modelo.

- **max_depth:** Profundidade máxima de uma árvore. Aumentar esse valor tornará o modelo mais complexo e com maior probabilidade de *overfit*.

- **n_estimators:** O número de estimadores ou iterações de aumento no processo de aumento de gradiente. Isto é o parâmetro $M$ na Equação 3.7 e controla quantas árvores são cultivadas no treinamento do modelo. Normalmente, quanto maior o número de iterações, melhor o modelo ficará, até que comece *overfitting* nos dados de treinamento.

- **early_stopping_rounds:** É um parâmetro utilizado para reduzir o *overfitting* ele recebe um valor inteiro que informa ao algoritmo quando parar se não houver mais melhorias na métrica de avaliação. Pode evitar o *overfitting* e melhorar o desempenho do seu modelo. Não necessariamente é um parâmetro que queremos 'tunar', mas vamos utilizar ele para reduzir o *overfitting*.

- **reg_lambda:** Termo de regularização $L2$ nos pesos.

## 3.7 Optuna

Uma das tarefas mais importantes na construção de modelos de aprendizado de máquina é a otimização de hiperparâmetros. Uma otimização correta dos hiperparâmetros se reflete diretamente no desempenho do modelo e nos últimos anos essa frente de otimização dos hiperparâmetros tem tido grande avanço na pesquisa e diversas soluções existem atualmente. Podemos citar Spearmint, Vizer, AutoSklearn, HyperOpt, dentre outras. Entretanto cada ferramente propõe uma própria abordagem de usabilidade podendo se tornar mais ou menos flexível dependo do

caso ou complexidade do modelo. Ou seja, a escalabilidade dessas aplicações pode ser dificultada. Com isso, surgiu o Optuna um framework Open Source para otimização de hiperparâmetros, cujo objetivo é unificar os paradigmas de otimização sob uma filosofia apoiada em três pilares: *API design-by-run*, implementação eficiente, facilidade de configuração e versatilidade de arquitetura (AKIBA *et al.*, 2019)(OPTUNA, ).

Na Tabela 4 temos uma comparação de outros algoritmos de otimização de hiperparâmetros e o Optuna (AKIBA *et al.*, 2019).

Tabela 4 – Comparativo de outros algorítimos de otimização de hiperparâmetros (AKIBA *et al.*, 2019).

| Framework | API | Pruning | Lightweight | Distributed | Dashboard | OSS |
|---|---|---|---|---|---|---|
| SMAC | define and run | ✗ | ✓ | ✗ | ✗ | ✓ |
| GPyOpt | define and run | ✗ | ✓ | ✗ | ✗ | ✓ |
| Spearmint | define and run | ✗ | ✓ | ✓ | ✗ | ✓ |
| Hyperopt | define and run | ✗ | ✓ | ✓ | ✗ | ✓ |
| Autotune | define and run | ✓ | ✗ | ✓ | ✓ | ✗ |
| Vizier | define and run | ✓ | ✗ | ✓ | ✓ | ✗ |
| Katib | define and run | ✓ | ✗ | ✓ | ✓ | ✓ |
| Tune | define and run | ✓ | ✗ | ✓ | ✓ | ✓ |
| Optuna | define by run | ✓ | ✓ | ✓ | ✓ | ✓ |

Por padrão o Optuna utiliza Tree-Structured Parzen Estimator, um algorítimo de otimização Bayesiana, conforme foi abordado no capitulo 2, mas podem ser utilizados outros algoritmos de otimização. Para mais informações sobre esse tipo de otimização e os outros algoritmos consulte (OPTUNA, ; YANG; SHAMI, 2020; HUTTER; KOTTHOFF; VANSCHOREN, 2019; ZHENG, 2022; WATANABE; HUTTER, 2022).

Antes de começar a implementar a otimização com Optuna, é necessário definir uma função objetivo. A função objetivo conterá toda a lógica de um processo regular de definição, treinamento e teste de modelo. Após a avaliação do modelo, ele deve retornar a métrica de avaliação que também é escolhida pelo usuário. A classe Trial será usada para armazenar informações de uma combinação específica de hiperparâmetros usados posteriormente pelo modelo de aprendizado de máquina. Um objeto de estudo pode então ser chamado para otimizar a função objetivo para encontrar a melhor combinação de hiperparâmetros. Em seguida, ele executará testes iterativamente até um teste ou tempo máximo definido pelo usuário. O ensaio com os melhores hiperparâmetros será armazenado em study.best_trial.

**Código-fonte 1** – Definição de uma função objetivo e o processo de otimização no Optuna

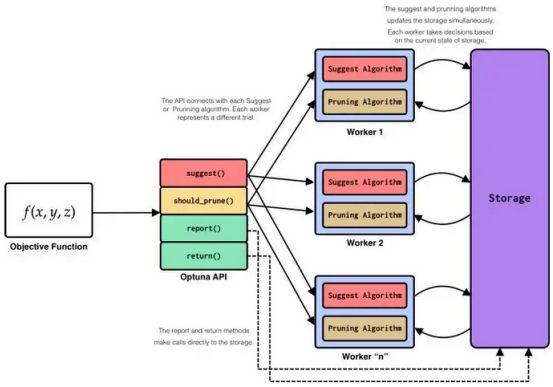
```
1: import optuna
2:
3: def objective(trial):
4:     x = trial.suggest_float('x', -10, 10)
5:     return (x - 2) ** 2
```

```
 6:
 7: study = optuna.create_study()
 8: study.optimize(objective, n_trials=100)
 9: trial = study.best_trial
10: print("Best Trial: ", trial.value)
```

Uma das principais vantagens do Optuna é sua estrutura avançada de visualizações para a interpretação de todo o processo de otimização, podemos utilizar a biblioteca para entender a relação entre os hiperparâmetros.

Além disso, o Optuna é projetado para facilidade de implementação, flexibilidade e escalabilidade. A otimização de experimentos em larga escala, por exemplo, pode ser realizada de maneira paralela e distribuída. Optuna é framework agnóstico, ou seja, pode ser facilmente integrado com qualquer um dos frameworks de aprendizado de máquina e aprendizado profundo, como: PyTorch, Tensorflow, Keras, Scikit-Learn, XGBoost, LGBM e etc.

Figura 13 – Arquitetura do Optuna




## 3.8 *SHAP e Shapley Values*

Entender a importância das variáveis de um modelo predito é um conceito extremamente necessário e importante. As variáveis podem explicar a dinâmica do problema no mundo real, que em muitos casos são desconhecidas, aprimorando o conhecimento do do problema para o cientista de dados ou pesquisadores envolvidos na construção do modelo. Também podem ser usadas para auditar modelos complexos e entender se o modelo não está discriminando algo errado (MENGNAN; NINGHAO; XIA, 2018).

Essa tarefa é extremamente complexa, pois como vimos os modelos que vamos utilizar neste estudo, GBMs, fazem diversas operações no conjunto de dados, inclusive operações não-lineares. Isso torna a quantificação da importância de cada variável extremamente complexa.

*SHAP (SHapley Additive exPlanations)* é uma abordagem que vem da teoria de jogos para explicar a saída de qualquer modelo de aprendizado de máquina. Ele conecta a saída do

modelo com as explicações locais usando os valores de Shapley. A principal ideia do SHAP é criar um modelo mais simples, para que se possa explicar as como as variáveis locais impactam no modelo final (LUNDBERG; LEE, 2017; LUNDBERG *et al.*, 2020; LUNDBERG; LEE, ).

Figura 14 – Exemplo do SHAP mostrando a importância de cada variável de entrada na saída do modelo (LUNDBERG; LEE, ).

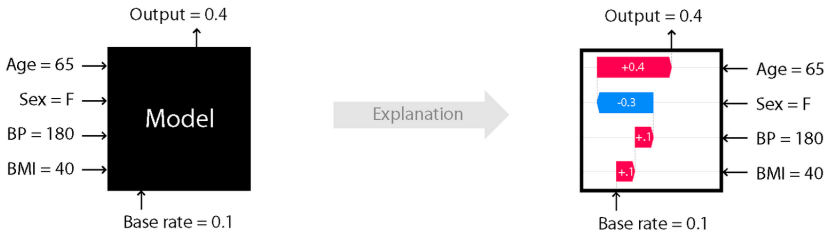
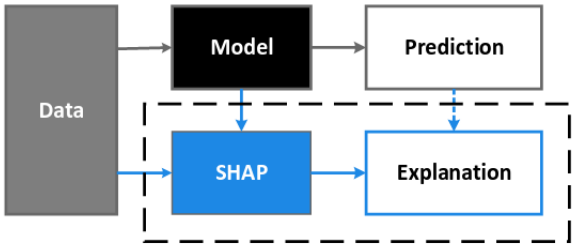



Figura 15 – Esquemático de como usar o SHAP para interpretar as predições do modelo (WANG *et al.*, 2020)




Estes métodos têm um modelo de explicação que é uma função linear de variáveis binárias:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{3.15}$$

Em que $z' \in \{0,1\}^M$, $M$ é o número de características de entrada simplificada e $\phi_i \in \mathbb{R}$. Os métodos atribuem um efeito $\phi_i \in \mathbb{R}$ a cada variável e a soma de todos os efeitos se aproxima do modelo original que queremos explicar. Uma das maiores contribuições do SHAP é propor a sua própria lógica de computar estes valores que garantem que eles tenham outras propriedades interessantes além do seu somatório aproximar a resposta do modelo original. E toda esta lógica surge dos *Shapley Values*.

Os *Shapley Values* foram introduzidos por Lloyd Shapley, no contexto da teoria dos jogos. Para um jogo cooperativo qualquer, os *Shapley Values* distribuem uma quantidade total de contribuição para cada jogador da equipe de forma justa, ou seja tenta quantificar a contribuição individual para o ganho total (SHAPLEY, 1952). Em aprendizado de máquina, os valores de

Shapley são a contribuição média de um valor de feature para a previsão total (WANG *et al.*, 2020).

$$\phi_i(f,x') = \sum_{z' \subseteq x'} \frac{|z'|!(M-|z'|-1)!}{M!}[f_x(z') - f_x(z' \setminus i)] \tag{3.16}$$

Figura 16 – SHAP valores atribuem a cada feição a mudança na previsão do modelo esperado ao condicionar esse recurso. Eles explicam como chegar do valor base $E[f(z)]$ que seria previsto se não conhecêssemos nenhum recurso para a saída atual $f(x)$. Este diagrama mostra um único pedido. Quando o modelo é não linear ou as características de entrada são não independentes, no entanto, a ordem em que os recursos são adicionados à expectativa importa, e os valores SHAP surgem da média dos valores $\phi_i$ em todas as ordenações possíveis. (LUNDBERG; LEE, 2017)

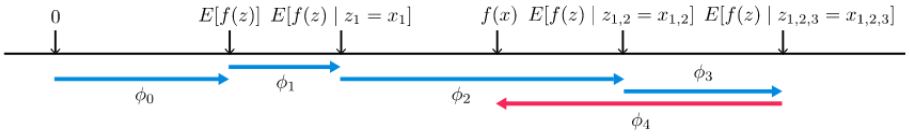
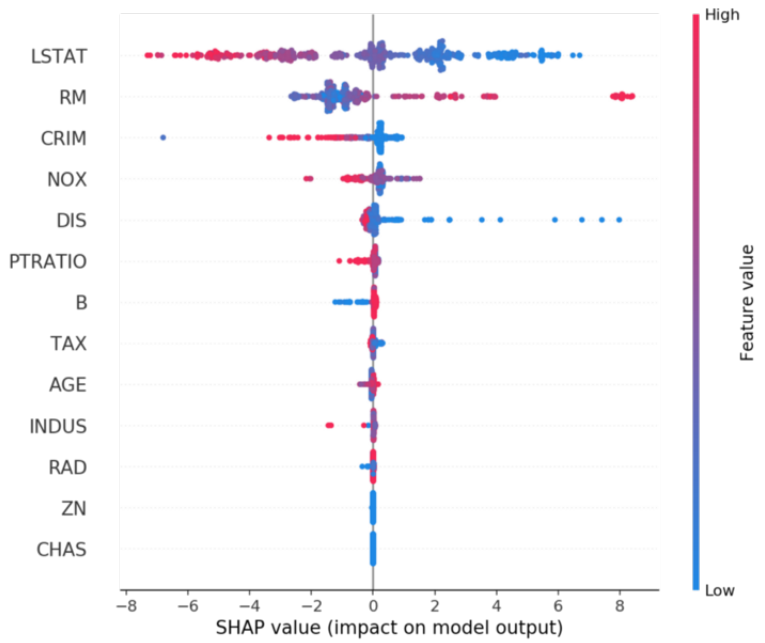



Figura 17 – Gráfico do SHAP para o dataset Boston Housing.



CAPÍTULO

# 4

# DESENVOLVIMENTO

O principal objetivo deste trabalho é estudar e comparar os algoritmos de boosting em 4 datasets diferentes e analisar a influência dos hiperparâmetros no modelo. O estudo será feito com os parâmetros *Default* de cada algoritmo, depois alguns deles serão alterados e, por final, utilizaremos o Optuna para otimizá-los.

O objetivo da escolha dos conjuntos de dados se baseou em analisar o comportamento dos modelos construídos e dos hiperparâmetros em conjuntos de dados com propriedades distintas, como por exemplo maior presença de variáveis categóricas ou apenas variáveis numéricas.

Nas próprias seções, será abordada uma visão geral da metodologia, uma análise dos conjuntos de dados e as implementações no *jupyter-notebook*.

## 4.1 Bibliotecas e Ferramentas

Todo o pipeline de estudo e análise é feito usando Python 3.8.8. No código final no GitHub será detalhado todas as versões em um arquivo *requeriments.txt*, mas de modo geral as principais bibliotecas utilizadas nesse estudo são:

- **numpy:** biblioteca em python para trabalhar com arrays n-dimensional e algumas funções matemáticas de álgebra linear.
- **pandas:** biblioteca em python utilizada para manipulação e análise de conjunto de dados e séries temporais.
- **matplotlib:** biblioteca em python para visualização de dados.
- **seaborn:** biblioteca em python para visualização de dados baseada no **matplotlib:**.
- **scipy:** biblioteca em python de pacote científico computacional

- **sys:** biblioteca que oferece funções e variáveis usadas para manipular diferentes partes do ambiente e do tempo de execução do python.
- **timeit:** biblioteca para medir o tempo de execução de programas python.
- **gc:** é o garbage collector utilizado nas linguagens de programação para rastrear os objetos na memória.
- **sklearn:** biblioteca de aprendizado de máquina em python.
- **xgboost:** biblioteca do XGBoost.
- **catboost:** biblioteca do CatBoost.
- **lightbm:** biblioteca do LightGBM.
- **optuna:** biblioteca do Optuna.
- **shap:** biblioteca do SHAP.

## 4.2 Data Cleaning, Data Preparation e Análise Descritiva

Nesta seção iremos fazer uma breve descrição dos conjuntos de dados, análise descritivas das variáveis e a distribuição nos conjuntos de dados. Primeiramente, precisamos ajustar dados que foram inseridos incorretamente, lidar com os valores *missing* e realizar as transformações, ou conversões, das variáveis para cada modelo, por exemplo variáveis categóricas para numéricas no caso do LightGBM e XGBoost. Todas as análises nos conjunto de dados seguem a mesma metodologia: identificar o tipo de cada coluna (int, float, str) verificar se existem valores *missing* e realizar as transformações ou criações de variáveis para cada modelo.

Serão utilizados 4 conjunto de dados diferentes para este estudo, todos eles podem ser encontrados no OpenML, Kaggle ou UCI Machine Learning Repository.

### 4.2.1 Diabetes

O primeiro conjunto de dados é o Diabetes Dataset 2019 Este conjunto de dados é originalmente do *Prediction of Type 2 Diabetes using Machine Learning Classification Methods*. O objetivo é prever, com base em medidas de diagnóstico, se um paciente tem diabetes. As variáveis são:

- **Idade:** idade faixa de anos.
- **Sexo:** sexo do paciente [M: Masculino, F: Feminino].
- **Diabetes na Família:** se tem diabetes no histórico familiar.

- **Pressão Alta:** se o paciente possui pressão alta.
- **Atividade Física:** se pratica atividade física.
- **IMC:** índice de massa corporal (peso em $kg$/(altura em $m$)$^2$).
- **Fumante:** indica se o paciente é fumante.
- **Alcoólatra:** indica se o paciente é alcoólatra.
- **Horas de sono:** quantidade de horas de sono
- **Sono Profundo:** horas de sono profundo.
- **Medicamentos Regulares:** indica se o paciente utiliza medicamentos
- **JunkFood:** se o paciente consome comidas gordurosas.
- **Stress:** stress do paciente.
- **BPLevel:** nível da pressão arterial.
- **Gravidezes:** número de gravidezes.
- **Pdiabetes:** se possui pré-diabetes.
- **Frequência Urinaria:** frequência urinaria.
- **Resultado:** variável alvo (0 ou 1), se possui ou não possui diabete.

O primeiro passo é identificar o nome de cada coluna, seus atributos e os valores *NaN* em cada linha de cada coluna. E para as variáveis numéricas uma análise descritiva, média, desvio padrão, contagem, mínimo e máximo e os quantis (25,50,75).

| Coluna | dtypes | *NaN* |
|---|---|---|
| Age | object | 0 |
| Gender | object | 0 |
| Family_Diabetes | object | 0 |
| highBP | object | 0 |
| PhysicallyActive | object | 0 |
| BMI | float64 | 4 |
| Smoking | object | 0 |
| Alcohol | object | 0 |
| Sleep | int64 | 0 |
| SoundSleep | int64 | 0 |
| RegularMedicine | object | 0 |
| JunkFood | object | 0 |
| Stress | object | 0 |
| BPLevel | object | 0 |
| Pregancies | float64 | 42 |
| Pdiabetes | object | 1 |
| UriationFreq | object | 0 |
| Diabetic | object | 1 |

(a) Tabela com o tipo de cada variável e quantidade de linhas *NaN* em cada coluna.

| | BMI | Sleep | SoundSleep | Pregancies |
|---|---|---|---|---|
| **count** | 948 | 952 | 952 | 910 |
| **mean** | 25.7637 | 6.9495 | 5.4957 | 0.3868 |
| **std** | 5.4025 | 1.2731 | 1.8656 | 0.9094 |
| **min** | 15 | 4 | 0 | 0 |
| **25%** | 22 | 6 | 4 | 0 |
| **50%** | 25 | 7 | 6 | 0 |
| **75%** | 29 | 8 | 7 | 0 |
| **max** | 45 | 11 | 11 | 4 |

(b) Estatística descritiva das variáveis numéricas.

Tabela 5 – Descritivo das variáveis no conjunto de dados Diabetes.

Precisamos realizar um tratamento nessas linhas com *NaN* e na variável resposta, Diabetic, transformando ela em 0 ou 1. Após isso devemos conferir as variáveis categorias estão escritas corretamente e podemos analisar a distribuição de algumas variáveis na população com diabete e sem diabete.

Figura 18 – População com Diabetes e sem Diabetes.

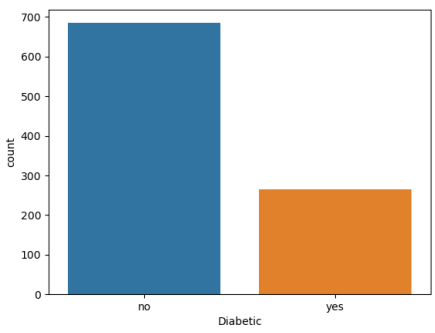
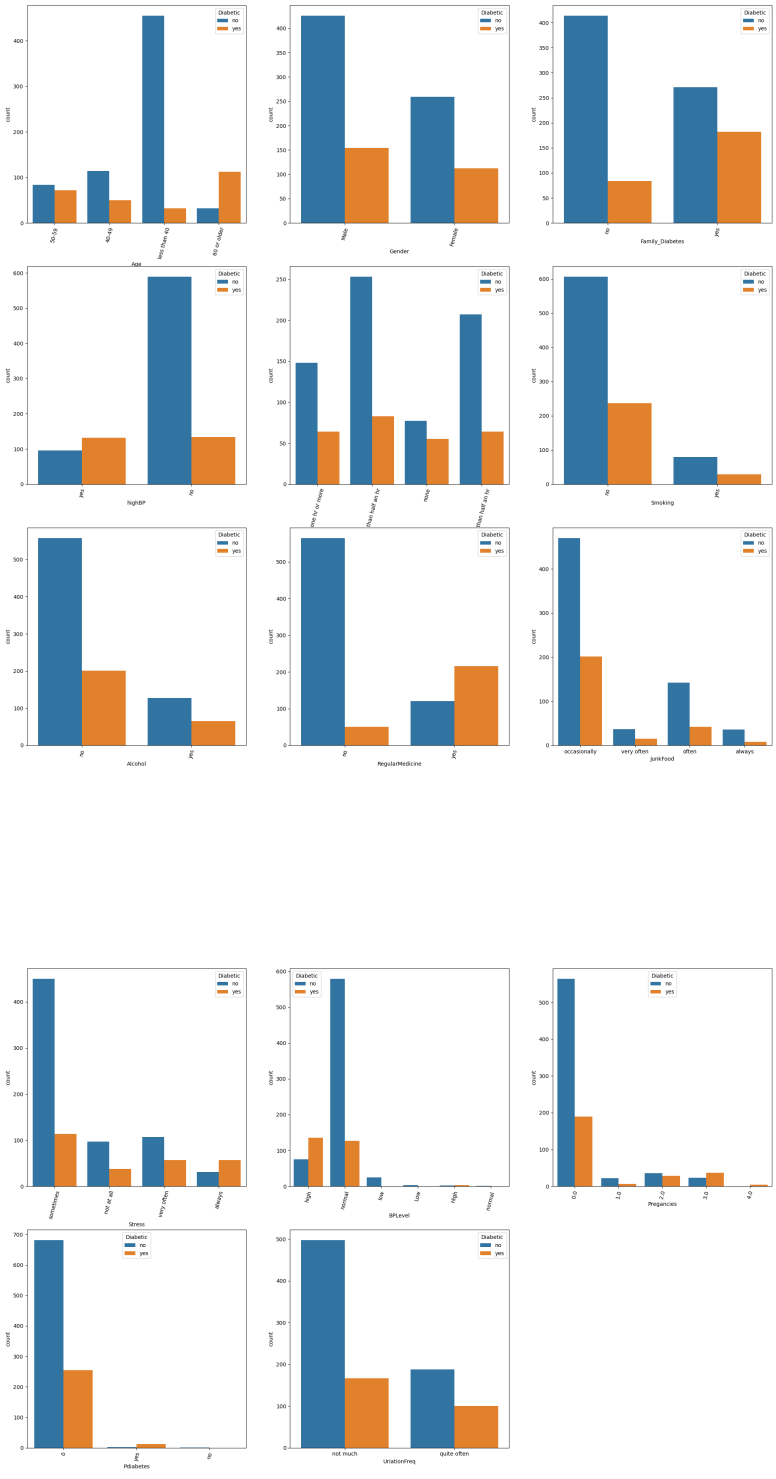


Figura 19 – Distribuição das Variáveis categóricas no conjunto de dados de Diabetes.



(a) Histograma das variáveis numéricas na população total.

(b) Histograma das variáveis numéricas na população com Diabete.

(c) Histograma das variáveis numéricas na população sem Diabete.

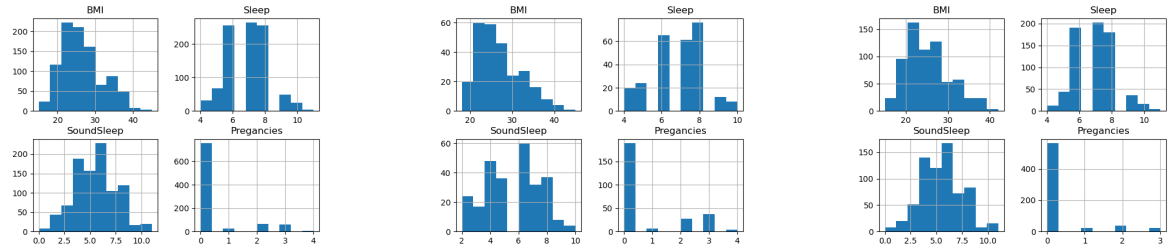



Figura 20 – Distribuição das variáveis numéricas no conjunto de dados de Diabetes.

A próxima etapa consistirá em dividir esse conjunto de dados em dois, o primeiro é com as próprias variáveis categóricas para o CatBoost enquanto que para o modelo do LightGBM e XGBoost iremos transformar as variáveis em ordinais ou utilizaremos *one_hot_enconding*. A partir disto já teremos os nossos conjuntos de dados para o treinamento dos modelos.

### 4.2.2 Insuficiência Cardíaca

O segundo conjunto de dados é o Heart Failure Prediction Dataset Este conjunto de dados foi criado combinando diferentes conjuntos de dados já disponíveis de forma independente, mas não combinados antes. Neste conjunto de dados, 5 conjuntos de dados cardíacos são combinados em 11 características comuns. O objetivo é identificar a insuficiência cardíaca com base nas seguintes variáveis:

- **Sexo:** sexo do paciente [M: Masculino, F: Feminino].
- **TipoDorPeito:** tipo de dor no peito.
- **Pressão arterial em repouso:** pressão arterial em repouso (mmHg).
- **Colesterol:** colesterol sérico[mm/dl]
- **JejumBS:** açúcar no sangue em jejum [1: se JejumBS > 120 mg/dl, 0: caso contrário].
- **ECG em repouso::** resultados do eletrocardiograma em repouso [Normal: Normal, ST: com anormalidade da onda ST-T (inversões da onda T e/ou elevação ou depressão do ST $>$ 0,05 mV).
- **HVE:** mostrando hipertrofia ventricular esquerda provável ou definitiva pelos critérios de Estes]
- **MaxHR:** frequência cardíaca máxima alcançada [Valor numérico entre 60 e 202].
- **ExerciseAngina:** angina induzida por exercício [S: Sim, N: Não].

- **Oldpeak:** oldpeak = ST [Valor numérico medido na depressão].

- **ST_Slope:** a inclinação do segmento ST do exercício de pico [Up: ascendente, Flat: plano, Down: descendente].

- **Idade:** idade em anos.

- **Resultado:** variável alvo (0 ou 1), se possui ou não possui insuficiência cardíaca.

| **Coluna** | **dtypes** | ***NaN*** |
|---|---|---|
| Age | int64 | 0 |
| Sex | object | 0 |
| ChestPainType | object | 0 |
| RestingBP | int64 | 0 |
| Cholesterol | int64 | 0 |
| FastingBS | int64 | 0 |
| RestingECG | object | 0 |
| MaxHR | int64 | 0 |
| ExerciseAngina | object | 0 |
| Oldpeak | float64 | 0 |
| ST_Slope | object | 0 |
| HeartDisease | int64 | 0 |

Tabela 6 – Tabela com o tipo de cada variável e quantidade de linhas *NaN* em cada coluna no conjunto de dados de Insuficiência Cardíaca.

| | **Age** | **RestingBP** | **Cholesterol** | **FastingBS** | **MaxHR** | **Oldpeak** | **HeartDisease** |
|---|---|---|---|---|---|---|---|
| **count** | 918.000 | 918.000 | 918.000 | 918.000 | 918.000 | 918.000 | 918.000 |
| **mean** | 53.511 | 132.397 | 198.800 | 0.233 | 136.809 | 0.887 | 0.553 |
| **std** | 9.433 | 18.514 | 109.384 | 0.423 | 25.460 | 1.067 | 0.497 |
| **min** | 28.000 | 0.000 | 0.000 | 0.000 | 60.000 | -2.600 | 0.000 |
| **25%** | 47.000 | 120.000 | 173.250 | 0.000 | 120.000 | 0.000 | 0.000 |
| **50%** | 54.000 | 130.000 | 223.000 | 0.000 | 138.000 | 0.600 | 1.000 |
| **75%** | 60.000 | 140.000 | 267.000 | 0.000 | 156.000 | 1.500 | 1.000 |
| **max** | 77.000 | 200.000 | 603.000 | 1.000 | 202.000 | 6.200 | 1.000 |

Tabela 7 – Estatística descritiva das variáveis numéricas no conjunto de dados de Insuficiência Cardíaca.

Neste conjunto de dados não precisamos fazer nenhum tratamento, pois os valores foram inseridos corretamente e não temos *missing*.

Figura 21 – População com Insuficiência Cardíaca e sem Insuficiência Cardíaca.

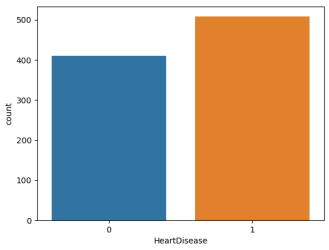
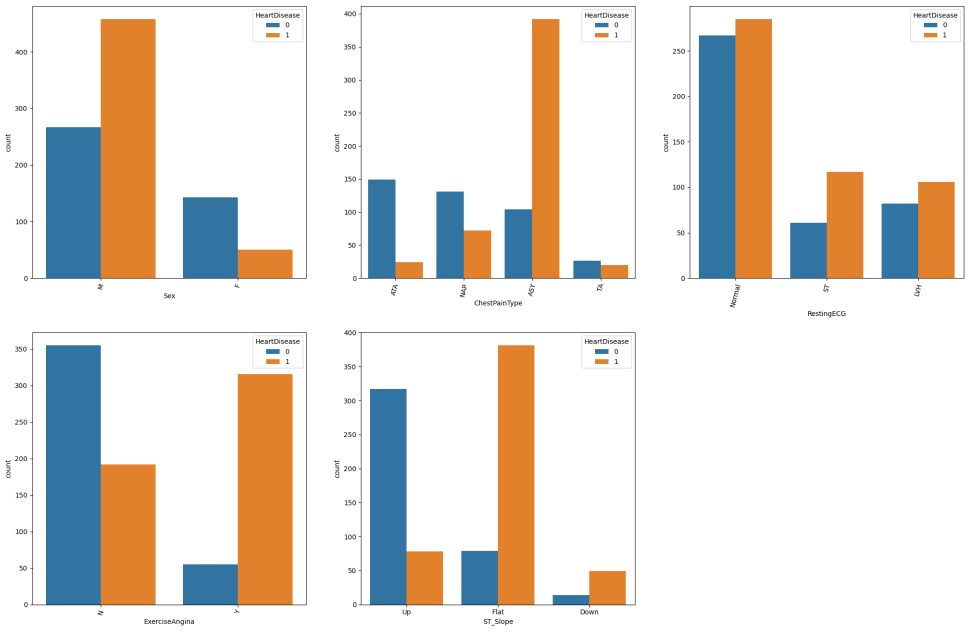



Figura 22 – Distribuição das variáveis categóricas no conjunto de dados de Insuficiência Cardíaca.



(a) Histograma das variáveis numéricas na população total.

(b) Histograma das variáveis numéricas na população com Insuficiência Cardíaca.

(c) Histograma das variáveis numéricas na população sem Insuficiência Cardíaca.

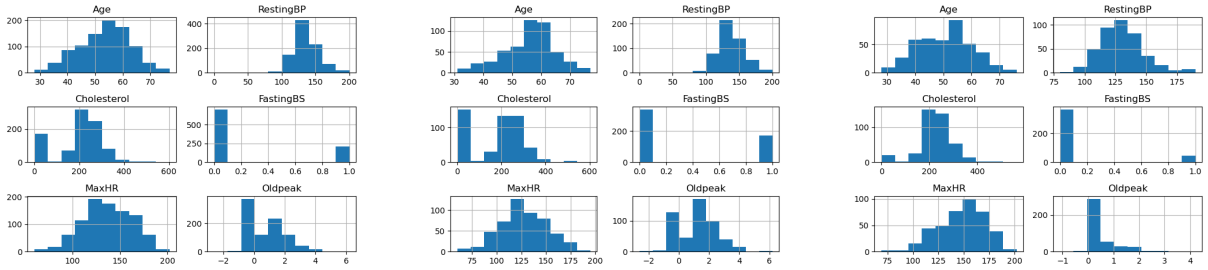



Figura 23 – Distribuição das variáveis numéricas no conjunto de dados de Insuficiência Cardíaca.

Neste caso iremos transformar as colunas categóricas em ordinais para o conjunto de treinamento nos modelos XGBoost e LightGBM.

### 4.2.3 Cálculo Renal

O terceiro conjunto de dados é o Kidney Stone Prediction based on Urine Analysis Este conjunto de dados pode ser usado para prever a presença de cálculos renais com base na análise de urina. As 79 amostras de urina foram analisadas para determinar se certas características físicas da urina podem estar relacionadas com a formação de cristais de oxalato de cálcio. Os dados são obtidos de *Physical Characteristics of Urines With and Without Crystals*, Springer Series in Statistics. E as variáveis são:

- **Gravidade Específica:** a densidade da urina em relação à água
- **pH:** pH, o logaritmo negativo do íon hidrogênio.
- **Osmolaridade** uma unidade usada em biologia e medicina, é proporcional à concentração de moléculas em solução.
- **Condutividade:** a condutividade é proporcional à concentração de carga íons em solução.
- **Concentração de Ureia:** concentração de ureia em mm por litro.
- **Concentração de Cálcio** concentração de cálcio.
- **Resultado:** variável alvo (0 ou 1), se possui ou não possui cálculo renal.

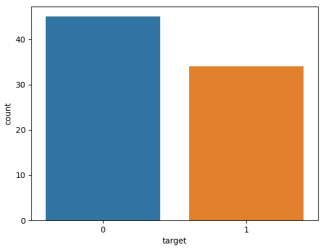
| **Coluna** | **dtypes** | ***NaN*** |
|---|---|---|
| gravity | float64 | 0 |
| ph | float64 | 0 |
| osmo | int64 | 0 |
| cond | float64 | 0 |
| urea | int64 | 0 |
| calc | float64 | 0 |
| target | int64 | 0 |

Tabela 8 – Tabela com o tipo de cada variável e quantidade de linhas *NaN* em cada coluna no conjunto de dados de Insuficiência Renal.

| | **gravity** | **ph** | **osmo** | **cond** | **urea** | **calc** | **target** |
|---|---|---|---|---|---|---|---|
| **count** | 79.000 | 79.000 | 79.000 | 79.000 | 79.000 | 79.000 | 79.000 |
| **mean** | 1.018 | 6.028 | 612.848 | 20.814 | 266.405 | 4.139 | 0.430 |
| **std** | 0.007 | 0.724 | 237.515 | 7.939 | 131.255 | 3.260 | 0.498 |
| **min** | 1.005 | 4.760 | 187.000 | 5.100 | 10.000 | 0.170 | 0.000 |
| **25%** | 1.012 | 5.530 | 413.000 | 14.150 | 160.000 | 1.460 | 0.000 |
| **50%** | 1.018 | 5.940 | 594.000 | 21.400 | 260.000 | 3.160 | 0.000 |
| **75%** | 1.023 | 6.385 | 792.000 | 26.550 | 372.000 | 5.930 | 1.000 |
| **max** | 1.040 | 7.940 | 1236.000 | 38.000 | 620.000 | 14.340 | 1.000 |

Tabela 9 – Estatística descritiva das variáveis numéricas no conjunto de dados de Insuficiência Renal.

Figura 24 – População com Insuficiência Renal e sem Insuficiência Renal.




(a) Histograma das variáveis numéricas na população total.

(b) Histograma das variáveis numéricas na população com Insuficiência Renal.

(c) Histograma das variáveis numéricas na população sem Insuficiência Renal.

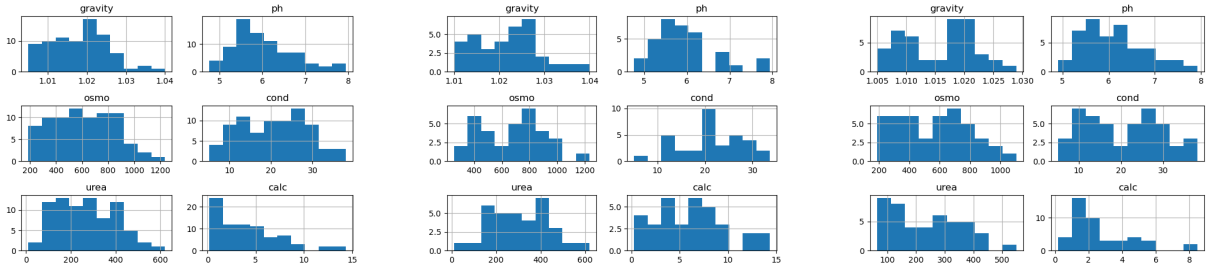



Figura 25 – Distribuição das variáveis numéricas no conjunto de dados de Insuficiência Renal.

Neste conjunto de dados todos os valores das colunas já são numéricos e não tem *missing*, nenhum tratamento para os dados será necessário. Podendo já utilizar eles na construção dos modelos.

### 4.2.4 Carcinoma da Mama

O quarto conjunto de dados é o Breast Cancer Wisconsin (Diagnostic) Data Set. Esse conjunto de dados possui características calculadas a partir de uma imagem digitalizada de um aspirado com agulha fina (PAAF) de uma massa mamária. Eles descrevem características dos núcleos celulares presentes na imagem. E o objetivo é diagnosticar o carcinoma da mama como benigno ou maligno.

- **Raio:** média das distâncias do centro aos pontos do perímetro.
- **Textura:** desvio padrão dos valores da escala de cinza.
- **Perímetro:** perímetro calculado para cada núcleo celular
- **Área:** área calculada para cada núcleo celular
- **Suavidade:** variação local nos comprimentos dos raios.
- **Compacidade:** perímetro$^2$ / área - 1,0.
- **Dimensão Fractal:** "aproximação da costa- 1
- **Simetria:** perímetro$^2$ / área - 1,0.
- **Pontos Côncavos:** número de porções côncavas do contorno.
- **Diagnóstico:** M = maligno, B = benigno.

| **Coluna** | **dtypes** | ***NaN*** |
|---|---|---|
| id | int64 | 0 |
| diagnosis | object | 0 |
| radius_mean | float64 | 0 |
| texture_mean | float64 | 0 |
| perimeter_mean | float64 | 0 |
| area_mean | float64 | 0 |
| smoothness_mean | float64 | 0 |
| compactness_mean | float64 | 0 |
| concavity_mean | float64 | 0 |
| concave points_mean | float64 | 0 |
| symmetry_mean | float64 | 0 |
| fractal_dimension_mean | float64 | 0 |
| radius_se | float64 | 0 |
| texture_se | float64 | 0 |
| perimeter_se | float64 | 0 |
| area_se | float64 | 0 |
| smoothness_se | float64 | 0 |
| compactness_se | float64 | 0 |
| concavity_se | float64 | 0 |
| concave points_se | float64 | 0 |
| symmetry_se | float64 | 0 |
| fractal_dimension_se | float64 | 0 |
| radius_worst | float64 | 0 |
| texture_worst | float64 | 0 |
| perimeter_worst | float64 | 0 |
| area_worst | float64 | 0 |
| smoothness_worst | float64 | 0 |
| compactness_worst | float64 | 0 |
| concavity_worst | float64 | 0 |
| concave points_worst | float64 | 0 |
| symmetry_worst | float64 | 0 |
| fractal_dimension_worst | float64 | 0 |

Tabela 10 – Tabela com o tipo de cada variável e quantidade de linhas *NaN* em cada coluna no conjunto de dados de Carcinoma da Mama.

Figura 26 – Estatística descritiva das variáveis numéricas no conjunto de dados de Carcinoma da Mama.

| | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean |
|---|---|---|---|---|---|---|---|---|
| count | 569 | 569 | 569 | 569 | 569 | 569 | 569 | 569 |
| mean | 14.12729174 | 19.28964851 | 91.96903339 | 654.8891037 | 0.0963602812 | 0.1043409842 | 0.08879931582 | 0.04891914587 |
| std | 3.524048826 | 4.301035768 | 24.29898104 | 351.9141292 | 0.01406412814 | 0.05281275793 | 0.07971980871 | 0.03880284486 |
| min | 6.981 | 9.71 | 43.79 | 143.5 | 0.05263 | 0.01938 | 0 | 0 |
| 25% | 11.7 | 16.17 | 75.17 | 420.3 | 0.08637 | 0.06492 | 0.02956 | 0.02031 |
| 50% | 13.37 | 18.84 | 86.24 | 551.1 | 0.09587 | 0.09263 | 0.06154 | 0.0335 |
| 75% | 15.78 | 21.8 | 104.1 | 782.7 | 0.1053 | 0.1304 | 0.1307 | 0.074 |
| max | 28.11 | 39.28 | 188.5 | 2501 | 0.1634 | 0.3454 | 0.4268 | 0.2012 |

| | symmetry_mean | fractal_dimension_mean | radius_se | texture_se | perimeter_se | area_se | smoothness_se | compactness_se |
|---|---|---|---|---|---|---|---|---|
| count | 569 | 569 | 569 | 569 | 569 | 569 | 569 | 569 |
| mean | 0.1811618629 | 0.06279760984 | 0.4051720562 | 1.216853427 | 2.866059227 | 40.33707909 | 0.0070497891 | 0.02547813884 |
| std | 0.02741428134 | 0.007060362795 | 0.277312733 | 0.5516483926 | 2.021854554 | 45.49100552 | 0.003002517944 | 0.01790817933 |
| min | 0.106 | 0.04996 | 0.1115 | 0.3602 | 0.757 | 6.802 | 0.001713 | 0.002252 |
| 25% | 0.1619 | 0.0577 | 0.2324 | 0.8339 | 1.606 | 17.85 | 0.005169 | 0.01308 |
| 50% | 0.1792 | 0.06154 | 0.3242 | 1.108 | 2.287 | 24.53 | 0.00638 | 0.02045 |
| 75% | 0.1957 | 0.06612 | 0.4789 | 1.474 | 3.357 | 45.19 | 0.008146 | 0.03245 |
| max | 0.304 | 0.09744 | 2.873 | 4.885 | 21.98 | 542.2 | 0.03113 | 0.1354 |

| | concavity_se | concave points_se | symmetry_se | fractal_dimension_se | radius_worst | texture_worst | perimeter_worst |
|---|---|---|---|---|---|---|---|
| count | 569 | 569 | 569 | 569 | 569 | 569 | 569 |
| mean | 0.03189371634 | 0.01179613708 | 0.02054229877 | 0.003794903866 | 16.26918981 | 25.6772232 | 107.2612127 |
| std | 0.03018606032 | 0.006170285174 | 0.008266371529 | 0.002646070967 | 4.83324158 | 6.146257623 | 33.60254227 |
| min | 0 | 0 | 0.007882 | 0.0008948 | 7.93 | 12.02 | 50.41 |
| 25% | 0.01509 | 0.007638 | 0.01516 | 0.002248 | 13.01 | 21.08 | 84.11 |
| 50% | 0.02589 | 0.01093 | 0.01873 | 0.003187 | 14.97 | 25.41 | 97.66 |
| 75% | 0.04205 | 0.01471 | 0.02348 | 0.004558 | 18.79 | 29.72 | 125.4 |
| max | 0.396 | 0.05279 | 0.07895 | 0.02984 | 36.04 | 49.54 | 251.2 |

| | area_worst | smoothness_worst | compactness_worst | concavity_worst | concave points_worst | symmetry_worst | fractal_dimension_worst |
|---|---|---|---|---|---|---|---|
| count | 569 | 569 | 569 | 569 | 569 | 569 | 569 |
| mean | 880.5831283 | 0.132368594 | 0.2542650439 | 0.2721884833 | 0.1146062232 | 0.2900755712 | 0.08394581722 |
| std | 569.3569927 | 0.0228324294 | 0.1573364889 | 0.2086242806 | 0.0657323412 | 0.06186746754 | 0.01806126735 |
| min | 185.2 | 0.07117 | 0.02729 | 0 | 0 | 0.1565 | 0.05504 |
| 25% | 515.3 | 0.1166 | 0.1472 | 0.1145 | 0.06493 | 0.2504 | 0.07146 |
| 50% | 686.5 | 0.1313 | 0.2119 | 0.2267 | 0.09993 | 0.2822 | 0.08004 |
| 75% | 1084 | 0.146 | 0.3391 | 0.3829 | 0.1614 | 0.3179 | 0.09208 |
| max | 4254 | 0.2226 | 1.058 | 1.252 | 0.291 | 0.6638 | 0.2075 |

Figura 27 – Distribuição de Carcinoma da Mama Maligno e Benigno na população.

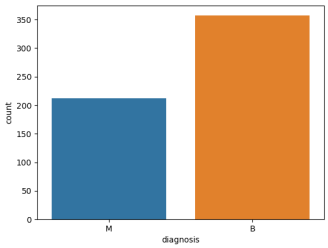



Figura 28 – Distribuição das variáveis numéricas no conjunto de dados de Carcinoma da Mama.

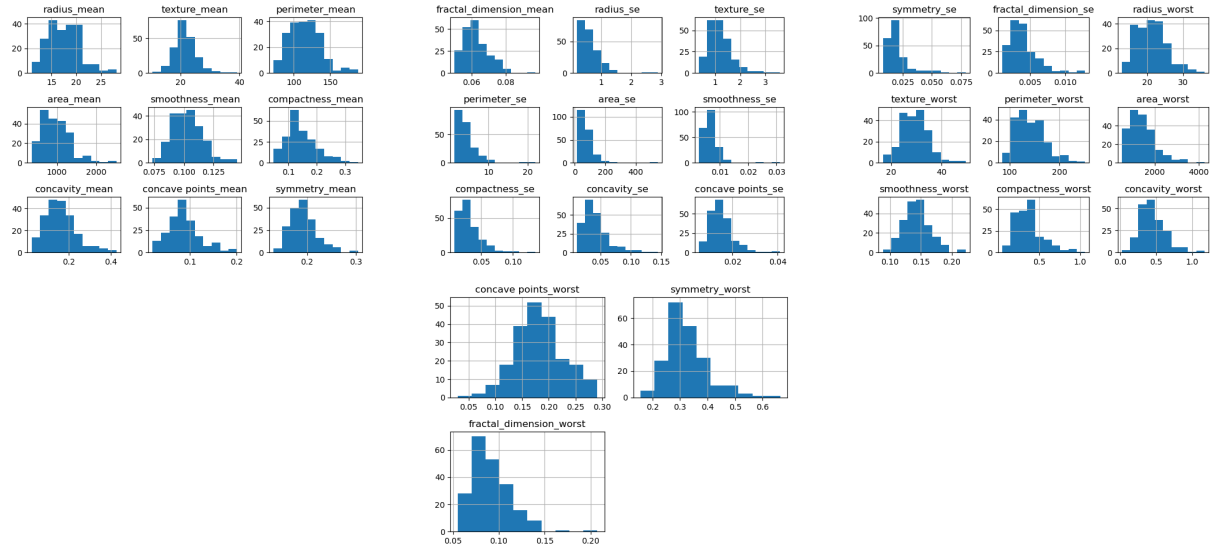



Figura 29 – Distribuição das variáveis numéricas no conjunto de dados da população com Carcinoma da Mama Maligno.

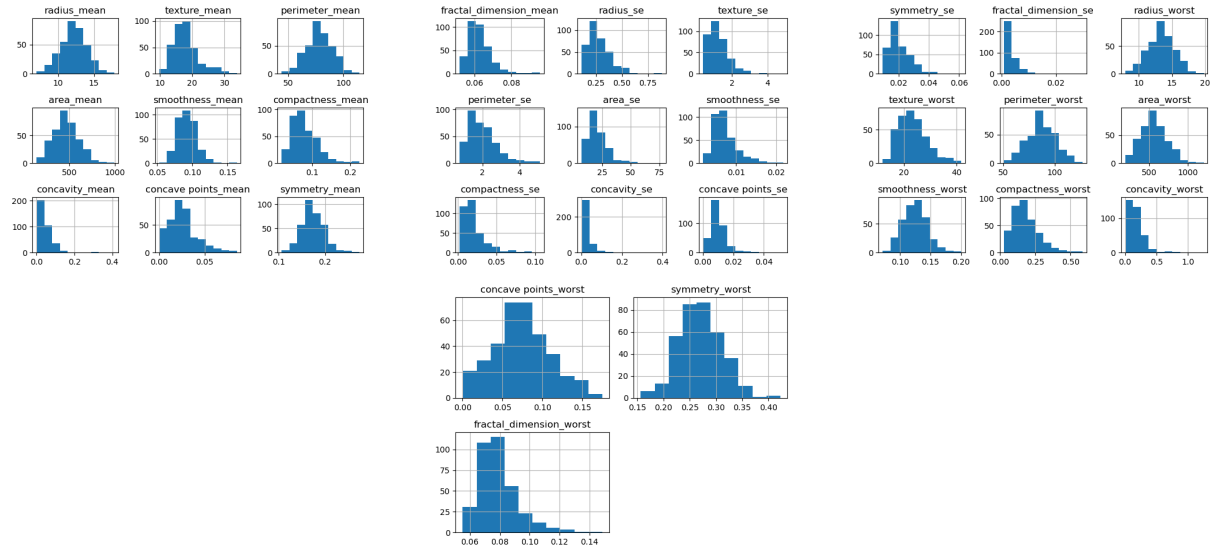



Figura 30 – Distribuição das variáveis numéricas no conjunto de dados da população com Carcinoma da Mama Benigno.

Neste conjunto de dados é necessário criar a variável target 0 ou 1 a partir da coluna **diagnosis** e a remoção da coluna **Unnamed: 32** do conjunto de dados. Após isso os dados já podem ser utilizados para o treino dos modelos preditivos.

## 4.3 Métricas de Validação de Performance e Desempenho

Para as métricas de performance, utilizamos a AUC, logloss e o KS. O código CITAR mostra a implementação em python dessas métricas.

**Código-fonte 2** – Código implementado o cálculo da AUC, logloss e o KS.

```
def auc_logloss_ks(y_test, y_prob):
    '''
    Input:
        y_prob: model predict prob
        y_test: target
    Output: Metrics of validation
        auc, ks, log_loss
    '''
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_prob)
    auc = metrics.auc(fpr, tpr)
    log_loss = sklearn.metrics.log_loss(y_test, y_prob)
    ks = max(tpr - fpr)
    return auc, log_loss, ks
```

Para o custo computacional, utilizou-se a biblioteca **timeit** para o cálculo em nanosegundos. Basta iniciar antes do treino de cada modelo e no final teremos o tempo calculado. O código 3 exemplifica o processo.

**Código-fonte 3** – Importação da biblioteca timeit e o cálculo em segundos da execução do código.

```
import timeit
start = timeit.default_timer()
model_XGB = train(X_train, y_train, X_test, y_test, balanced='
    balanced', method='XGBoost', number_trials=100)
stop = timeit.default_timer()
stop - start
```

## 4.4 Divisão de Treino e Teste

Iremos utilizar a função *train_test_split* do **sklearn** para separar o nosso conjunto de treino e teste. X representa as variáveis explicativas que entraram no treino do modelo e o $y$ representa a variável resposta, target. Não será utilizado o conjunto de validação, pois temos pouco volume de dados em alguns conjunto de dados.

Para os conjunto de dados que possuem apenas dados numéricos será utilizado o mesmo conjunto de treino e teste para todos os modelos, enquanto que os conjunto de dados que possuem variáveis categóricas serão divididos em um conjunto de dados só numéricos para o XGBoost e LightGBM e outro conjunto com variáveis numéricas e categóricas para o CatBoost.

**Código-fonte 4** – Exemplo da divisão dos conjuntos de treino e teste.

```
1: X_train, X_test, y_train, y_test = train_test_split(X, y,
       test_size=0.3, random_state=42)
```

## 4.5 Espaço de Hiperparâmetros

Conforme abordado em 3.6, temos alguns parâmetros que são comuns para os três algoritmos, nesta primeira parte do estudo em que vamos alterar os hiperparâmetros manualmente vamos focar neles: **learning_rate , max_depth, n_estimators, reg_lambda**. Vamos denotar os hiperparâmetros respectivamente como *LR*, *MD*, *RegL*. E os valores que vamos testar são:

$$LR = \{0.1, 0.3\}$$
$$MD = \{3, 6\}$$
$$RegL = \{0.01, 0.05\}$$

## 4.6 Tuning do modelo com o Optuna

Para o tuning com o Optuna foi criada a função objetivo, tuning e o train, de forma que a existência de variáveis categóricas no conjunto de dados já é identificada e, para o caso do *CatBoost*, as separada automaticamente.

O código 5 mostra todos os parâmetros das funções e o espaço dos hiperparâmetros que o Optuna irá percorrer para realizar o processo de tuning em cada um dos algoritmos diferentes. Foi utilizado 1 hora como o tempo máximo para o Optuna tunar cada algoritmo, ou seja, cada conjunto de dados demorou cerca de 3 horas para obter os resultados finais.

**Código-fonte 5** – Código implementado para o Tuning com o Optuna.

```
 1: import numpy as np
 2: import pandas as pd
 3: import xgboost as xgb
 4: import sys
 5: import timeit
 6: import gc
 7: import sklearn
 8: import seaborn
 9:
10: from sklearn import metrics
11: from xgboost import XGBClassifier
```

```
12: from catboost import CatBoostClassifier
13: from sklearn.model_selection import train_test_split
14: import lightgbm
15: from lightgbm import LGBMClassifier
16: from sklearn.metrics import roc_auc_score
17: from sklearn.metrics import accuracy_score
18: import optuna
19: from sklearn.model_selection import GridSearchCV
20: import matplotlib.pylab as plt
21: from sklearn.model_selection import cross_val_score
22: from sklearn.model_selection import StratifiedKFold
23: from optuna.visualization import plot_optimization_history
24: from optuna.visualization import plot_param_importances
25:
26: def auc_logloss_ks(y_test, y_prob):
27:     '''
28:     Input:
29:         y_prob: model predict prob
30:         y_test: target
31:     Output: Metrics of validation
32:         auc, ks, log_loss
33:     '''
34:     fpr, tpr, thresholds = metrics.roc_curve(y_test, y_prob)
35:     auc = metrics.auc(fpr, tpr)
36:     log_loss = sklearn.metrics.log_loss(y_test, y_prob)
37:     ks = max(tpr - fpr)
38:     return auc, log_loss, ks
39:
40: def objective(trial, X_train, y_train, X_test, y_test, balanced
        , method):
41:     '''
42:     Input:
43:         trial: trial of the test
44:         X_train:
45:         y_train:
46:         X_test:
47:         y_test:
48:         balanced:balanced or None
49:         method: XGBoost, CatBoost or LGBM
50:     Output: Metrics of validation
51:         auc, ks, log_loss
52:         auc_logloss_ks(y_test, y_pred)[0]
```

```
53:     '''
54:     gc.collect()
55:     if method=='LGBM':
56:         param_grid = {'learning_rate': trial.suggest_float('
      learning_rate', 0.0001, 0.1, log=True),
57:                       'num_leaves': trial.suggest_int('
      num_leaves', 2, 256),
58:                       'lambda_l1': trial.suggest_float("
      lambda_l1", 1e-8, 10.0, log=True),
59:                       'lambda_l2': trial.suggest_float("
      lambda_l2", 1e-8, 10.0, log=True),
60:                       'min_data_in_leaf': trial.suggest_int('
      min_data_in_leaf', 5, 100),
61:                       'max_depth': trial.suggest_int('max_depth
      ', 5, 64),
62:                       'feature_fraction': trial.suggest_float("
      feature_fraction", 0.4, 1.0),
63:                       'bagging_fraction': trial.suggest_float("
      bagging_fraction", 0.4, 1.0),
64:                       'bagging_freq': trial.suggest_int("
      bagging_freq", 1, 7),
65:
66:                      }
67:         model = LGBMClassifier(**param_grid)
68:
69:         print('LGBM - Optimization using optuna')
70:         model.fit(X_train, y_train)
71:
72:         y_pred = model.predict_proba(X_test)[:,1]
73:
74:     if method=='CATBoost':
75:         param_grid = {'learning_rate': trial.suggest_float('
      learning_rate', 0.0001, 0.1, log=True),
76:                       'depth': trial.suggest_int("depth", 4,
      10),
77:                       'max_bin': trial.suggest_int('max_bin',
      200, 400),
78:                       'min_data_in_leaf': trial.suggest_int('
      min_data_in_leaf', 1, 300),
79:                       'l2_leaf_reg': trial.suggest_float('
      l2_leaf_reg', 1e-8, 10, log = True),
80:                       'random_seed': 42,
```

```
81:                     'random_strength': trial.suggest_float("
    random_strength", 1e-8, 10.0, log=True),
82:                     'bagging_temperature': trial.
    suggest_float("bagging_temperature", 0.0, 10.0),
83:                     'od_type': trial.suggest_categorical("
    od_type", ["IncToDec", "Iter"]),
84:                     'od_wait': trial.suggest_int("od_wait",
    10, 50),
85:                     }
86:         if len(X_train._get_numeric_data().columns) != len(
    X_train.columns):
87:             categorical_features_indices = list(X_train.
    select_dtypes(exclude='number').columns)
88:             model = CatBoostClassifier(**param_grid)
89:             print('CATBoost - Optimization using optuna')
90:             model.fit(X_train, y_train,cat_features=
    categorical_features_indices,verbose=False)
91:             y_pred = model.predict_proba(X_test)[:,1]
92:         else:
93:             model = CatBoostClassifier(**param_grid)
94:             print('CATBoost - Optimization using optuna')
95:             model.fit(X_train, y_train,verbose=False)
96:             y_pred = model.predict_proba(X_test)[:,1]
97:
98:     if method=='XGBoost':
99:         param_grid = {'learning_rate': trial.suggest_float('
    learning_rate', 0.0001, 0.1, log=True),
100:                     'max_depth': trial.suggest_int('max_depth
    ', 3, 16),
101:                     'min_child_weight': trial.suggest_int('
    min_child_weight', 1, 300),
102:                     'gamma': trial.suggest_float('gamma', 1e
    -8, 1.0, log = True),
103:                     'alpha': trial.suggest_float('alpha', 1e
    -8, 1.0, log = True),
104:                     'lambda': trial.suggest_float('lambda',
    0.0001, 10.0, log = True),
105:                     'colsample_bytree': trial.suggest_float('
    colsample_bytree', 0.1, 0.8),
106:                     'booster': 'gbtree',
107:                     'random_state': 42,
108:                     }
```

```
109:         model = XGBClassifier(**param_grid)
110:         print('XGBoost - Optimization using optuna')
111:         model.fit(X_train, y_train,verbose=False)
112:         y_pred = model.predict_proba(X_test)[:,1]
113:
114:     auc_res, log_loss_res, ks_res = auc_logloss_ks(y_test,
      y_pred)
115:     print('auc:'+str(auc_res),', log_loss:'+str(log_loss_res),'
      , ks:'+str(ks_res))
116:     return auc_logloss_ks(y_test, y_pred)[0]
117:
118: def tuning(X_train, y_train, X_test, y_test, balanced, method):
119:     '''
120:     Input:
121:         trial:
122:         x_train:
123:         y_train:
124:         X_test:
125:         y_test:
126:         balanced:balanced or not balanced
127:         method: XGBoost, CatBoost or LGBM
128:     Output: Metrics of validation
129:         auc, ks, log_loss
130:         auc_logloss_ks(y_test, y_pred)[0]
131:     '''
132:     study = optuna.create_study(direction='maximize',
      study_name=method+' Classifier')
133:     func = lambda trial: objective(trial, X_train, y_train,
      X_test, y_test, balanced, method)
134:     print('Starting the optimization')
135:     time_max_tuning = 60*30 # max time in seconds to stop
136:     study.optimize(func, timeout=time_max_tuning)
137:     return study
138:
139: def train(X_train, y_train, X_test, y_test, balanced, method):
140:     '''
141:     Input:
142:         X_train:
143:         y_train:
144:         X_test:
145:         y_test:
146:         balanced:balanced or None
```

```
147:         method: XGBoost, CatBoost or LGBM
148:     Output: predict model
149:     '''
150:     print('Tuning')
151:     study = tuning(X_train, y_train, X_test, y_test, balanced,
        method)
152:     if method=='LGBM':
153:         model = LGBMClassifier(**study.best_params)
154:         print('Last Fit')
155:         model.fit(X_train, y_train, eval_set=[(X_test,y_test)],
156:                   callbacks = [lightgbm.early_stopping(
        stopping_rounds=100), lightgbm.log_evaluation(period=5000)])
157:     if method=='CATBoost':
158:         model = CatBoostClassifier(**study.best_params)
159:         if len(X_train._get_numeric_data().columns) != len(
        X_train.columns):
160:             categorical_features_indices = list(X_train.
        select_dtypes(exclude='number').columns)
161:             print('Last Fit')
162:             model.fit(X_train, y_train,cat_features=
        categorical_features_indices, eval_set=[(X_test,y_test)],
163:                   early_stopping_rounds=100,verbose = False)
164:         else:
165:             print('Last Fit')
166:             model.fit(X_train, y_train, eval_set=[(X_test,
        y_test)],
167:                   early_stopping_rounds=100,verbose = False)
168:     if method=='XGBoost':
169:         model = XGBClassifier(**study.best_params)
170:         print('Last Fit')
171:         model.fit(X_train, y_train, eval_set=[(X_test,y_test)],
172:                   early_stopping_rounds=100,verbose = False)
173:     return model, study
```

## 4.7 Explicabilidade dos Modelos

Para as versões finais de cada modelo, que são aquelas que obtiveram as melhores métricas de performance, iremos aplicar os Valores de SHAP para realizar a explicabilidade de cada variável e seu impacto no modelo final.

O código 6 demonstra como é a execução do SHAP.

**Código-fonte 6** – Importação da biblioteca shap e o cálculo dos shap values.

```
1: import shap
2: explainer = shap.TreeExplainer(model_XGB)
3: shap_values = explainer.shap_values(X_train)
4: shap.summary_plot(shap_values, X_train,plot_size=.7)
```

CAPÍTULO

# 5

# ANÁLISE EXPERIMENTAL

Nesta seção serão mostrados os resultados experimentais do capítulo 4. As saídas de alguns resultados podem ser encontradas no arquivo Jupyter-notebook no GitHub. As análises consistem em alterar os hiperparâmetros, conforme explicado no capítulo 4, depois aplicar o Optuna para os 3 modelos, com o tempo máximo de execução de trinta minutos para cada um.

## 5.1 Resultados de Conjunto de dados de Diabetes

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| **AUC** | 0.93639 | 0.89655 | 0.92066 |
| **logloss** | 1.69072 | 2.65684 | 2.29456 |
| **KS** | 0.87278 | 0.79311 | 0.84131 |
| **tempo (s)** | 0.17799 | 1.78698 | 0.08261 |

Tabela 11 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros *Default* no conjunto de dados de diabetes.

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| **AUC** | 0.92598 | 0.89428 | 0.89406 |
| **logloss** | 2.17379 | 3.13992 | 2.89838 |
| **KS** | 0.85195 | 0.78856 0 | 0.78812 |
| **tempo (s)** | 0.10825 | 1.05492 | 0.09304 |

Tabela 12 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de diabetes.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.92598 | 0.90448 | 0.92598 |
| **logloss** | 2.17379 | 2.41531 | 2.17379 |
| **KS** | 0.85195 | 0.80895 | 0.85195 |
| **tempo (s)** | 0.11743 | 1.00966 | 0.08180 |

Tabela 13 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de diabetes.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.93107 | 0.91534 | 0.92066 |
| **logloss** | 1.81149 | 2.41532 | 2.29456 |
| **KS** | 0.86215 | 0.83067 | 0.84131 |
| **tempo (s)** | 0.12579 | 1.82060 | 0.11067 |

Tabela 14 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 6$, $Reg_L = 0.01$ no conjunto de dados de diabetes.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.89406 | 0.89428 | 0.89406 |
| **logloss** | 2.89838 | 3.13992 | 2.89838 |
| **KS** | 0.78812 | 0.78856 | 0.78812 |
| **tempo (s)** | 0.09512 | 1.38364 | 0.08838 |

Tabela 15 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.05$ no conjunto de dados de diabetes.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.93107 | 0.89927 | 0.92598 |
| **logloss** | 1.81149 | 2.65685 | 2.17379 |
| **KS** | 0.86215 | 0.79854 | 0.85195 |
| **tempo (s)** | 0.13494 | 2.58085 | 0.09262 |

Tabela 16 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.01$ no conjunto de dados de diabetes.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.92836 | 0.90470 | 0.92598 |
| **logloss** | 1.81148 | 2.65685 | 2.17379 |
| **KS** | 0.85672 | 0.80940 | 0.85195 |
| **tempo (s)** | 0.13301 | 1.81456 | 0.10295 |

Tabela 17 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.05$ no conjunto de dados de diabetes.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.92598 | 0.90492 | 0.92598 |
| **logloss** | 2.17379 | 2.89839 | 2.17379 |
| **KS** | 0.85195 | 0.80984 | 0.85195 |
| **tempo (s)** | 0.10512 | 0.97739 | 0.09690 |

Tabela 18 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 3$, $Reg_L = 0.05$ no conjunto de dados de diabetes.

### 5.1.1 Optuna no Conjunto de dados de Diabetes

No final da execução, o Optuna nos retorna a quantidade de *trials* e qual foi a que obteve a melhor performance. Abaixo temos os códigos com as melhores saídas do Optuna de cada modelo.

**Código-fonte 7** – Resultado do Optuna no conjunto de dados de Diabetes.

```
1: XGBoost_model = train(X_train, y_train, X_test, y_test,
       balanced='balanced', method='XGBoost')
2: Trial 5143 finished with value: 0.9734596631205674 and
       parameters: {'learning_rate': 0.08503540937695053, '
       max_depth': 16, 'min_child_weight': 1, 'gamma':
       0.3765138366132899, 'alpha': 0.00011157064804299481, 'lambda
       ': 0.06609519167136822, 'colsample_bytree':
       0.7351361006258642}. Best is trial 5052 with value:
       0.9759530141843972.
3: ...
4: Trial 5052 finished with value: 0.9759530141843972 and
       parameters: {'learning_rate': 0.09999081729983263, '
       max_depth': 15, 'min_child_weight': 1, 'gamma':
       0.7591251036860868, 'alpha': 0.00010662969781278809, 'lambda
       ': 0.06239587908284276, 'colsample_bytree':
       0.7655930247173538}. Best is trial 5052 with value:
       0.9759530141843972.
5: XGBoost - Optimization using optuna
6: auc:0.9759530141843972 , log_loss:0.17300349809156104 , ks
       :0.8782136524822696
7:
8: CatBoost_model = train(X_train_cat, y_train_cat, X_test_cat,
       y_test_cat, balanced='balanced', method='CATBoost')
9: Trial 230 finished with value: 0.9786125886524822 and
       parameters: {'learning_rate': 0.008066790947749524, 'depth':
        10, 'max_bin': 282, 'min_data_in_leaf': 48, 'l2_leaf_reg':
```

```
    2.120609777416555, 'random_strength': 1.9674219542672667e
    -06, 'bagging_temperature': 6.752333264637867, 'od_type': '
    Iter', 'od_wait': 16}. Best is trial 160 with value:
    0.9811059397163121.
10: ...
11: Trial 160 finished with value: 0.9811059397163121 and
    parameters: {'learning_rate': 0.005585552379158199, 'depth':
     10, 'max_bin': 316, 'min_data_in_leaf': 11, 'l2_leaf_reg':
    1.1081451827078879, 'random_strength': 7.466692400471057e
    -07, 'bagging_temperature': 9.443675875030443, 'od_type': '
    IncToDec', 'od_wait': 17}. Best is trial 160 with value:
    0.9811059397163121.
12: CATBoost - Optimization using optuna
13: auc:0.9811059397163121 , log_loss:0.17088682855569354 , ks
    :0.8779920212765957
14:
15: LGBM_model = train(X_train, y_train, X_test, y_test, balanced='
    balanced', method='LGBM')
16: Trial 4540 finished with value: 0.9765625 and parameters: {'
    learning_rate': 0.07003556517783399, 'num_leaves': 191, '
    lambda_l1': 3.2225284078443206e-06, 'lambda_l2':
    4.1456610637867624e-08, 'min_data_in_leaf': 11, 'max_depth':
     55, 'feature_fraction': 0.9234036984685567, '
    bagging_fraction': 0.630684594822488, 'bagging_freq': 6}.
    Best is trial 1283 with value: 0.9816626773049646.
17: ...
18: Trial 1283 finished with value: 0.9806626773049646 and
    parameters: {'learning_rate': 0.08425779644832665, '
    num_leaves': 205, 'lambda_l1': 2.7481689793447196e-06, '
    lambda_l2': 4.307867154100011e-08, 'min_data_in_leaf': 9, '
    max_depth': 63, 'feature_fraction': 0.9232766751662997, '
    bagging_fraction': 0.9517850099076617, 'bagging_freq': 6}.
    Best is trial 1283 with value: 0.9806626773049646.
19: LGBM - Optimization using optuna
20: auc:0.9816626773049646 , log_loss:0.1859868135509998 , ks
    :0.8832003546099291
```

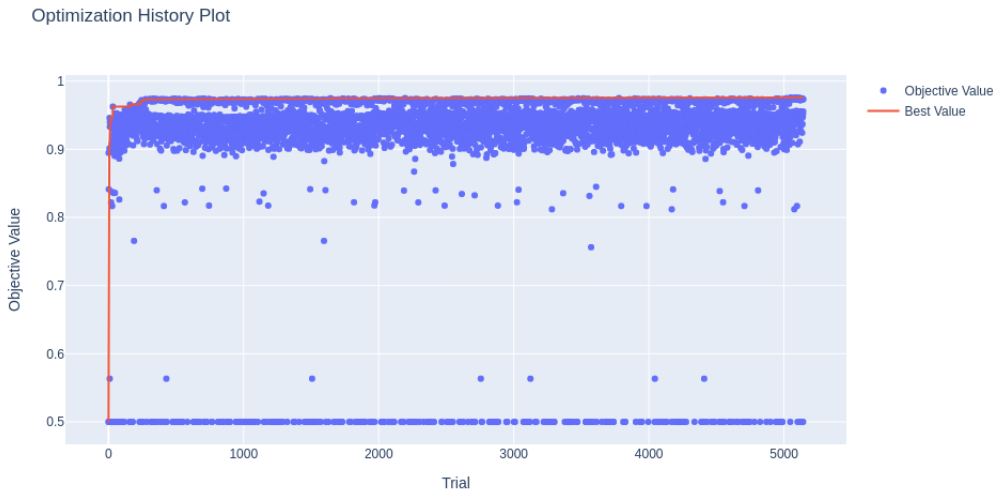
| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.97595 | 0.981106 | 0.98166 |
| **logloss** | 0.17300 | 0.17327 | 0.18599 |
| **KS** | 0.87821 | 0.87799 | 0.88320 |

Tabela 19 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros otimizados pelo Optuna no conjunto de dados de diabetes.

### 5.1.2 Resultados do estudo do Optuna no conjunto de dados de Diabetes utilizando o XGBoost.

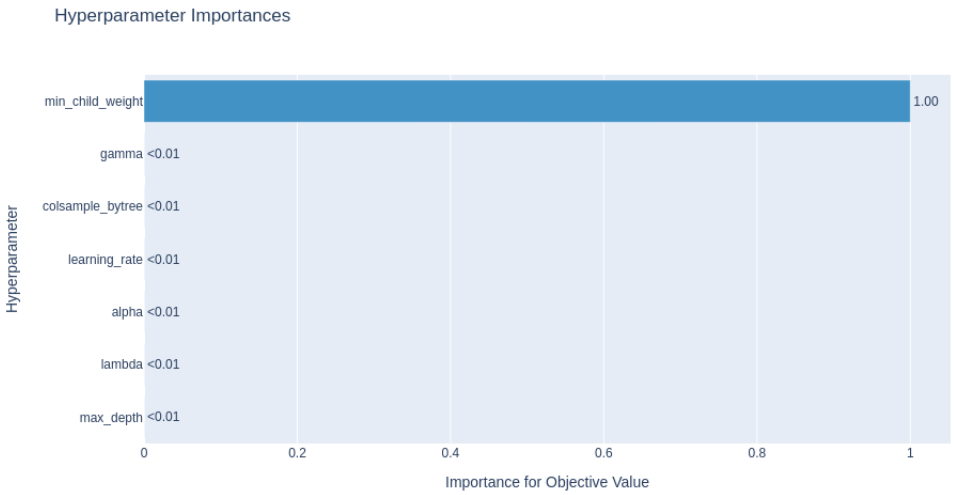
A primeira coisa é analisar todos os treinos e os valores encontrados para cada interação do Optuna para cada modelo.

Figura 31 – Valores do estudo do XGBoost no conjunto de dados de Diabetes pelo Optuna.




A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

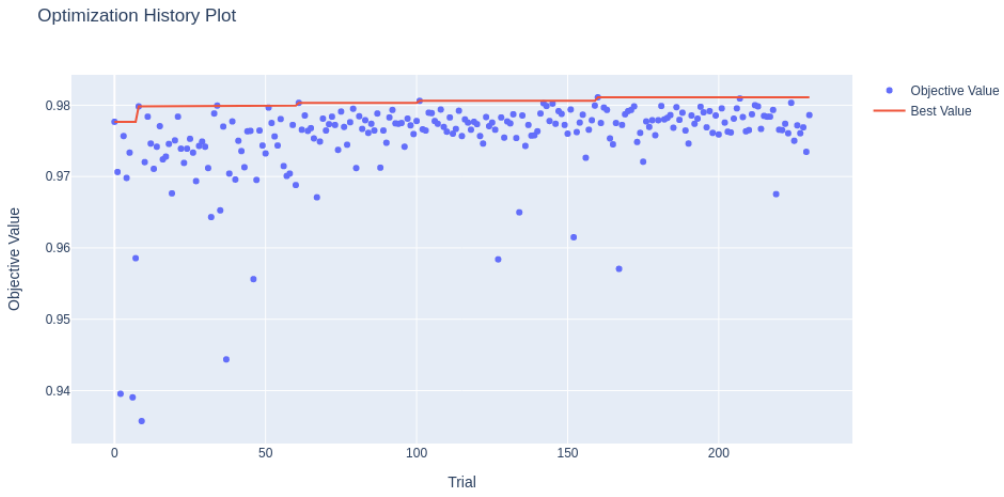
Figura 32 – Hiperparâmetros do XGBoost com maior importância no Optuna no dados de Diabetes.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foi o *min_child_weight*.

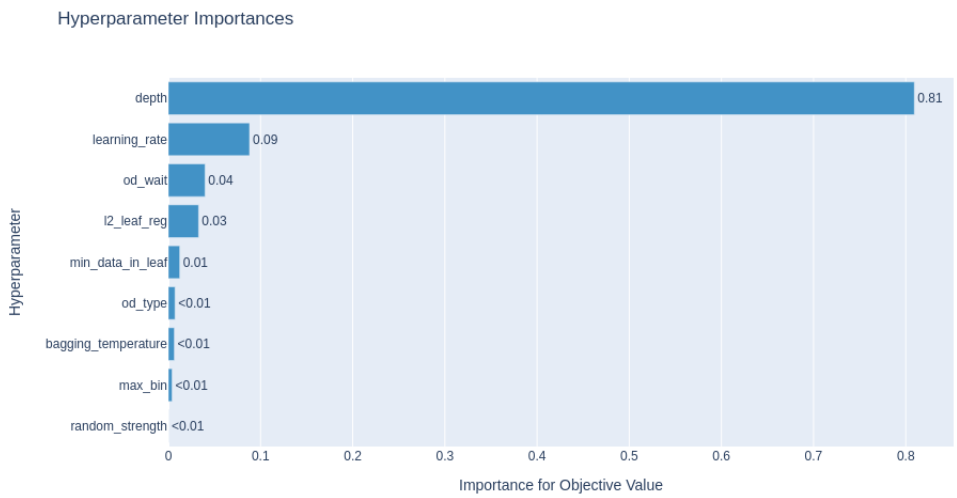
### 5.1.3 Resultados do estudo do Optuna no conjunto de dados de Diabetes utilizando o CatBoost.

Figura 33 – Valores do estudo do CatBoost no conjunto de dados de Diabetes pelo Optuna.




A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

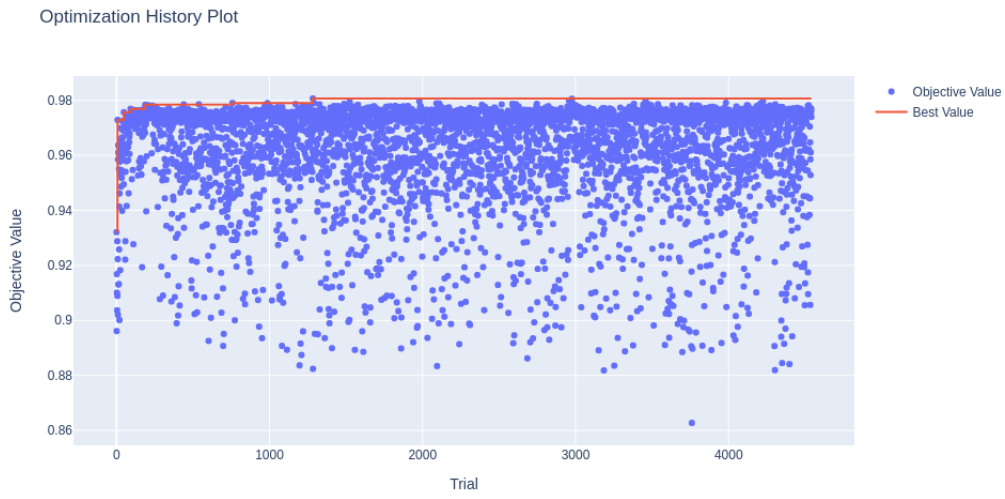
Figura 34 – Hiperparâmetros do XGBoost com maior importância no Optuna no dados de Diabetes.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foram o *depth* e o *learning_rate*.

### 5.1.4 Resultados do estudo do Optuna no conjunto de dados de Diabetes utilizando o LightGBM.

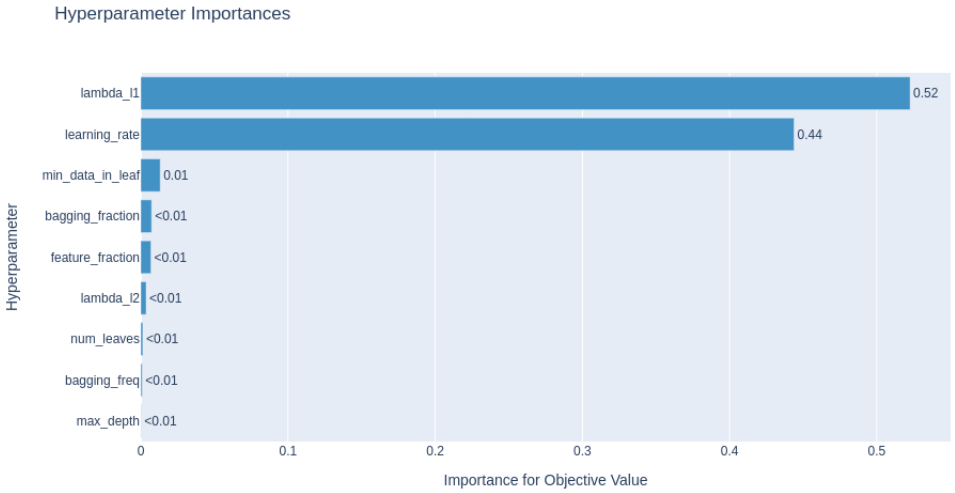
Figura 35 – Valores do estudo do LightGBM no conjunto de dados de Diabetes pelo Optuna.




.

A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

Figura 36 – Hiperparâmetros do LightGBM com maior importância no Optuna no dados de Diabetes.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foram o *lambda_l1* e o *learning_rate*.

## 5.2 Resultados Conjunto de dados de Insuficiência Cardíaca

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.84930 | 0.90048 | 0.86291 |
| **logloss** | 5.25595 | 3.50396 | 0.75538 |
| **KS** | 0.69861 | 0.80096 | 0.72583 |
| **tempo (s)** | 0.13467 | 1.41323 | 0.06788 |

Tabela 20 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros *Default* no conjunto de dados de insuficiência cardíaca.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.87021 | 0.87631 | 0.88219 |
| **logloss** | 4.63023 | 4.37995 | 4.25481 |
| **KS** | 0.74042 | 0.75261 | 0.76437 |
| **tempo (s)** | 0.10487 | 0.92945 | 0.05179 |

Tabela 21 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de insuficiência cardíaca.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.86738 | 0.84745 | 0.87021 |
| **logloss** | 4.63024 | 5.50622 | 4.63023 |
| **KS** | 0.73476 | 0.69490 | 0.74042 |
| **tempo (s)** | 0.12673 | 0.90088 | 0.10750 |

Tabela 22 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de insuficiência cardíaca.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.84016 | 0.87348 | 0.86574 |
| **logloss** | 5.63137 | 4.37996 | 4.75538 |
| **KS** | 0.68031 | 0.74695 | 0.73149 |
| **tempo (s)** | 0.09536 | 1.42294 | 0.06001 |

Tabela 23 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 6$, $Reg_L = 0.01$ no conjunto de dados de insuficiência cardíaca.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.86879 | 0.88850 | 0.87043 |
| **logloss** | 4.63024 | 3.87939 | 4.50510 |
| **KS** | 0.73759 | 0.77700 | 0.74085 |
| **tempo (s)** | 0.11721 | 0.94580 | 0.08204 |

Tabela 24 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.05$ no conjunto de dados de insuficiência cardíaca.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.84016 | 0.87348 | 0.85965 |
| **logloss** | 5.63137 | 4.37996 | 5.00566 |
| **KS** | 0.68031 | 0.74695 | 0.71929 |
| **tempo (s)** | 0.09273 | 1.45088 | 0.09684 |

Tabela 25 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.01$ no conjunto de dados de insuficiência cardíaca.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.84909 | 0.86433 | 0.86313 |
| **logloss** | 5.38108 | 4.75538 | 4.63025 |
| **KS** | 0.69817 | 0.72866 | 0.72626 |
| **tempo (s)** | 0.10147 | 1.44247 | 0.08749 |

Tabela 26 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.05$ no conjunto de dados de insuficiência cardíaca.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.85355 | 0.85823 | 0.88545 |
| **logloss** | 5.25594 | 5.00566 | 4.00453 |
| **KS** | 0.70710 | 0.71646 | 0.77091 |
| **tempo (s)** | 0.09487 | 0.87380 | 0.07974 |

Tabela 27 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 3$, $Reg_L = 0.05$ no conjunto de dados de insuficiência cardíaca.

### 5.2.1 Optuna no Conjunto de dados de Insuficiência Cardíaca

No final da execução o Optuna nos retorna a quantidade de *trials* e qual obteve a melhor performance. Abaixo temos os códigos com as melhores saídas do Optuna de cada modelo.

**Código-fonte 8** – Resultado do Optuna no conjunto de dados de Insuficiência Cardíaca.

```
1: XGBoost_model = train(X_train, y_train, X_test, y_test,
     balanced='balanced', method='XGBoost')
2: Trial 5428 finished with value: 0.9363567073170732 and
     parameters: {'learning_rate': 0.0027923512828505526, '
     max_depth': 11, 'min_child_weight': 6, 'gamma':
     1.49842650443622e-06, 'alpha': 3.3380595403852844e-05, '
     lambda': 1.4463893324744872, 'colsample_bytree':
     0.3115616538843963}. Best is trial 1049 with value:
     0.9585204703832753.
3: ...
4: Trial 1049 finished with value: 0.9555204703832753 and
     parameters: {'learning_rate': 0.08798533546026498, '
     max_depth': 8, 'min_child_weight': 6, 'gamma':
     1.0023609559031719e-07, 'alpha': 4.900000133344086e-07, '
     lambda': 0.0039762308968834745, 'colsample_bytree':
     0.7168230705897498}. Best is trial 1049 with value:
     0.9555204703832753.
5: XGBoost - Optimization using optuna
6: auc:0.9585204703832753 , log_loss:0.2824692901390929 , ks
     :0.7900696864111498
7:
8: CatBoost_model = train(X_train_cat, y_train_cat, X_test_cat,
     y_test_cat, balanced='balanced', method='CATBoost')
9: Trial 1071 finished with value: 0.9390243902439025 and
     parameters: {'learning_rate': 0.0008886403252391745, 'depth'
     : 6, 'max_bin': 369, 'min_data_in_leaf': 90, 'l2_leaf_reg':
     7.197661215691214, 'random_strength': 0.05598542767802503, '
     bagging_temperature': 6.059262032457767, 'od_type': '
```

```
      IncToDec', 'od_wait': 32}. Best is trial 645 with value:
      0.9501851045296168.
10: ...
11: Trial 645 finished with value: 0.9501851045296168 and
      parameters: {'learning_rate': 0.006239961585898258, 'depth':
       6, 'max_bin': 376, 'min_data_in_leaf': 90, 'l2_leaf_reg':
      9.62802219566606, 'random_strength': 0.4312538078007199, '
      bagging_temperature': 2.905228965519412, 'od_type': '
      IncToDec', 'od_wait': 50}. Best is trial 645 with value:
      0.9501851045296168.
12: CATBoost - Optimization using optuna
13: auc:0.9501851045296168 , log_loss:0.30122664951900086 , ks
      :0.7868031358885018
14:
15: LGBM_model = train(X_train, y_train, X_test, y_test, balanced='
      balanced', method='LGBM')
16: Trial 4574 finished with value: 0.9337979094076655 and
      parameters: {'learning_rate': 0.0028036602874199667, '
      num_leaves': 14, 'lambda_l1': 0.000827098906192435, '
      lambda_l2': 5.27506369644634, 'min_data_in_leaf': 24, '
      max_depth': 58, 'feature_fraction': 0.5068463692550502, '
      bagging_fraction': 0.9295403673354412, 'bagging_freq': 6}.
      Best is trial 3373 with value: 0.9583514808362369.
17: ...
18: Trial 3373 finished with value: 0.9583514808362369 and
      parameters: {'learning_rate': 0.0765705960307223, '
      num_leaves': 80, 'lambda_l1': 0.3822971805534857, 'lambda_l2
      ': 3.2542108432719523, 'min_data_in_leaf': 26, 'max_depth':
      45, 'feature_fraction': 0.4658823406792122, '
      bagging_fraction': 0.9753478288001076, 'bagging_freq': 6}.
      Best is trial 3373 with value: 0.9583514808362369.
19: LGBM - Optimization using optuna
20: auc:0.9583514808362369 , log_loss:0.29086650451368734 , ks
      :0.8094512195121951
```

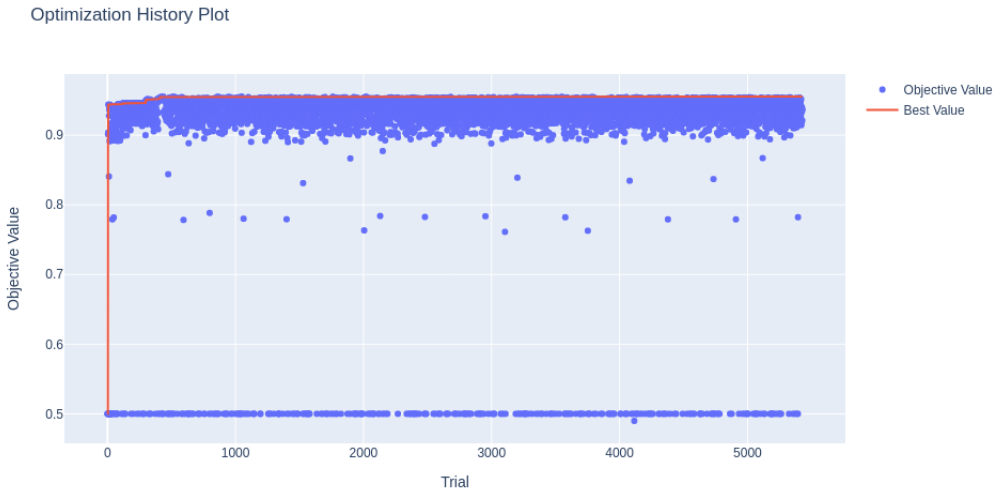
| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.95852 | 0.95019 | 0.95835 |
| **logloss** | 0.28247 | 0.30123 | 0.29087 |
| **KS** | 0.81598 | 0.78680 | 0.80945 |

Tabela 28 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros otimizados pelo Optuna no conjunto de dados de Insuficiência Cardíaca.

### 5.2.2 Resultados do estudo do Optuna no conjunto de dados de Insuficiência Cardíaca utilizando o XGBoost.

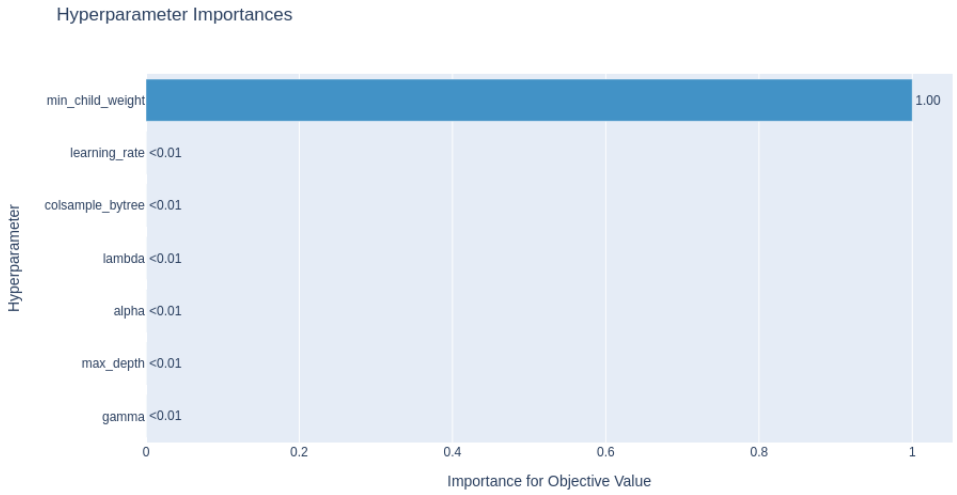
Novamente, vamos analisar os resultados do estudo do Optuna.

Figura 37 – Valores do estudo do XGBoost no conjunto de dados de Insuficiência Cardíaca pelo Optuna.




A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

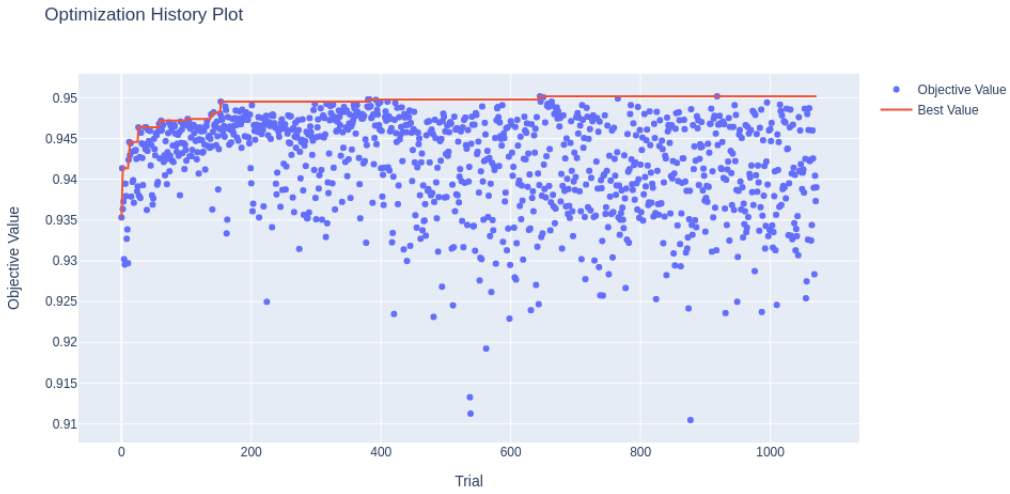
Figura 38 – Hiperparâmetros do XGBoost com maior importância no Optuna no dados de Insuficiência Cardíaca.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foi o *min_child_weight*.

### 5.2.3 Resultados do estudo do Optuna no conjunto de dados de Insuficiência Cardíaca utilizando o CatBoost.

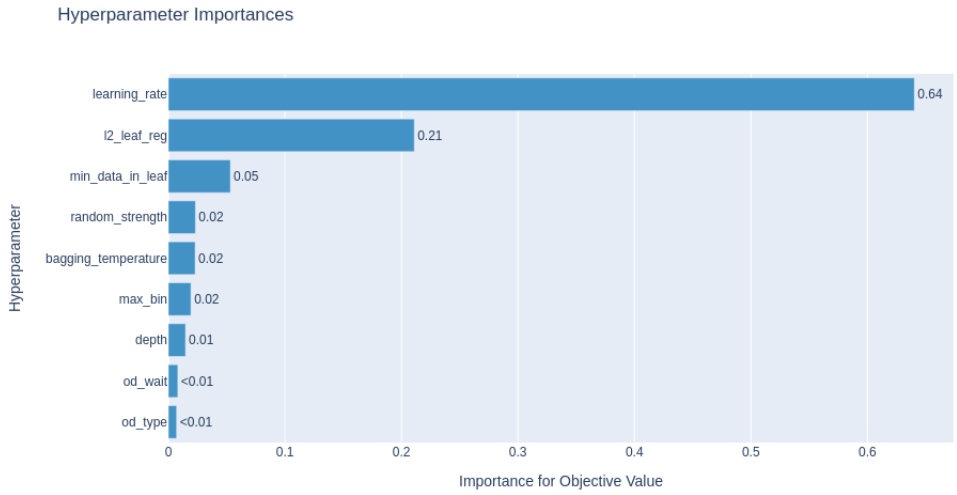
Figura 39 – Valores do estudo do CatBoost no conjunto de dados de Insuficiência Cardíaca pelo Optuna.




.

A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

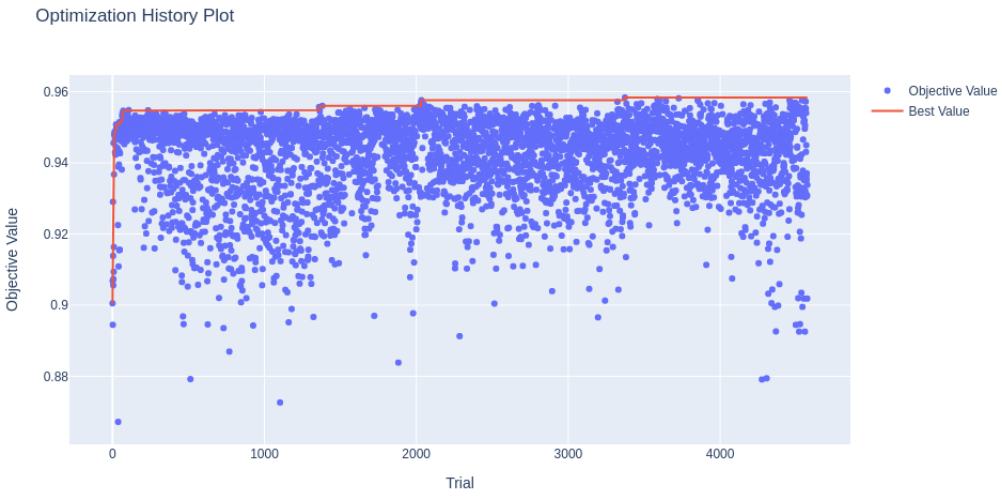
Figura 40 – Hiperparâmetros do XGBoost com maior importância no Optuna no dados de Insuficiência Cardíaca.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foram o *learning_rate* e o *l2_leaf_reg*.

### 5.2.4 Resultados do estudo do Optuna no conjunto de dados de Insuficiência Cardíaca utilizando o LightGBM.

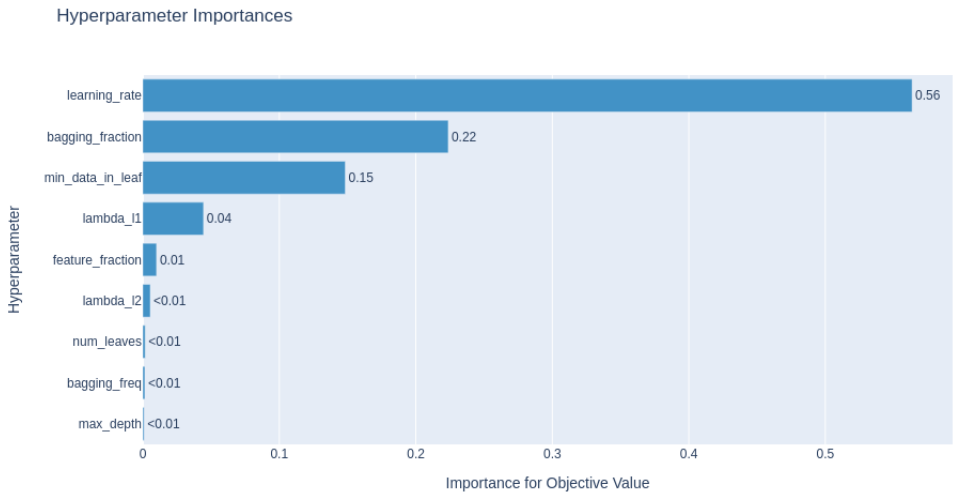
Figura 41 – Valores do estudo do LightGBM no conjunto de dados de Insuficiência Cardíaca pelo Optuna.




.

A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

Figura 42 – Hiperparâmetros do LightGBM com maior importância no Optuna no dados de Insuficiência Cardíaca.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foram o *learning_rate* e o *min_data_in_leaf*.

## 5.3 Resultados Conjunto de dados de Insuficiência Renal

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.60714 | 0.72857 | 0.72857 |
| **logloss** | 12.95217 | 8.63476 | 8.63476 |
| **KS** | 0.21429 | 0.45714 | 0.45714 |
| **tempo (s)** | 0.08901 | 0.47443 | 0.05795 |

Tabela 29 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros *Default* no conjunto de dados de insuficiência renal.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.67857 | 0.74286 | 0.72857 |
| **logloss** | 10.07388 | 8.63479 | 8.63476 |
| **KS** | 0.35714 | 0.48571 | 0.45714 |
| **tempo (s)** | 0.10748 | 0.27430 | 0.09398 |

Tabela 30 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de insuficiência renal.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.64286 | 0.74286 | 0.72857 |
| **logloss** | 11.51303 | 8.63479 | 8.63476 |
| **KS** | 0.28571 | 0.48571 | 0.45714 |
| **tempo (s)** | 0.03914 | 0.29207 | 0.02660 |

Tabela 31 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de insuficiência renal.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.67857 | 0.64286 | 0.72857 |
| **logloss** | 10.07388 | 11.51303 | 8.63476 |
| **KS** | 0.35714 | 0.28571 | 0.45714 |
| **tempo (s)** | 0.09602 | 0.41013 | 0.02825 |

Tabela 32 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.01$ no conjunto de dados de insuficiência renal.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.67857 | 0.74286 | 0.72857 |
| **logloss** | 10.07388 | 8.63479 | 8.63476 |
| **KS** | 0.35714 | 0.48571 | 0.45714 |
| **tempo (s)** | 0.10289 | 0.30287 | 0.03446 |

Tabela 33 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.05$ no conjunto de dados de insuficiência renal.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.67857 | 0.74286 | 0.72857 |
| **logloss** | 10.07388 | 8.63479 | 8.63476 |
| **KS** | 0.35714 | 0.48571 | 0.45714 |
| **tempo (s)** | 0.06784 | 0.30287 | 0.03446 |

Tabela 34 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.01$ no conjunto de dados de insuficiência renal.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.59286 | 0.72857 | 0.72857 |
| **logloss** | 12.95214 | 8.63476 | 8.63476 |
| **KS** | 0.18571 | 0.45714 | 0.45714 |
| **tempo (s)** | 0.05298 | 0.40619 | 0.07998 |

Tabela 35 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.05$ no conjunto de dados de insuficiência renal.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.64286 | 0.74286 | 0.72857 |
| **logloss** | 11.51303 | 8.63479 | 8.63476 |
| **KS** | 0.28571 | 0.48571 | 0.45714 |
| **tempo (s)** | 0.09940 | 0.27348 | 0.06459 |

Tabela 36 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 3$, $Reg_L = 0.05$ no conjunto de dados de insuficiência renal.

### 5.3.1 Optuna no Conjunto de dados de Insuficiência Renal

No final da execução o Optuna nos retorna a quantidade de *trials* e qual obteve a melhor performance. Abaixo temos os códigos com as melhores saídas do Optuna de cada modelo.

**Código-fonte 9** – Resultado do Optuna no conjunto de dados de Insuficiência Renal.

```
1: XGBoost_model = train(X_train, y_train, X_test, y_test,
     balanced='balanced', method='XGBoost')
2: Trial 5455 finished with value: 0.5 and parameters: {'
     learning_rate': 0.00017538956659154782, 'max_depth': 13, '
     min_child_weight': 7, 'gamma': 0.6809045773614, 'alpha':
     0.00030549017814601384, 'lambda': 0.0016283305853035024, '
     colsample_bytree': 0.7004592509281623}. Best is trial 10
     with value: 0.8
3: ...
4: Trial 10 finished with value: 0.8 and parameters: {'
     learning_rate': 0.008569334892039517, 'max_depth': 15, '
```

```
    min_child_weight': 4, 'gamma': 9.423677047435132e-06, 'alpha
    ': 1.185431670644414e-08, 'lambda': 0.012256251224167266, '
    colsample_bytree': 0.786661562086467}. Best is trial 10 with
     value: 0.8.
[I 2023-02-06 18:14:10,324] Trial 11 finished with value: 0.5
    and parameters: {'learning_rate': 0.007031543912682467, '
    max_depth': 16, 'min_child_weight': 16, 'gamma':
    1.0409510458372083e-05, 'alpha': 1.2193044690486938e-08, '
    lambda': 0.013018199439117679, 'colsample_bytree':
    0.7896246643321345}. Best is trial 10 with value: 0.8.
XGBoost - Optimization using optuna
auc:0.8 , log_loss:0.5702363587915897 , ks:0.5142857142857143

CatBoost_model = train(X_train_cat, y_train_cat, X_test_cat,
    y_test_cat, balanced='balanced', method='CATBoost')
Trial 2257 finished with value: 0.7785714285714286 and
    parameters: {'learning_rate': 0.0009648415391471772, 'depth'
    : 6, 'max_bin': 385, 'min_data_in_leaf': 211, 'l2_leaf_reg':
     1.6926351852497825, 'random_strength': 3.143996435524371, '
    bagging_temperature': 4.896899901794452, 'od_type': '
    IncToDec', 'od_wait': 48}. Best is trial 147 with value:
    0.9.
...
Trial 147 finished with value: 0.9 and parameters: {'
    learning_rate': 0.07537894328903638, 'depth': 5, 'max_bin':
    358, 'min_data_in_leaf': 248, 'l2_leaf_reg':
    1.2327052318936288e-07, 'random_strength':
    0.49381984936103734, 'bagging_temperature': 7.0491042648346,
     'od_type': 'IncToDec', 'od_wait': 43}. Best is trial 147
    with value: 0.9.
CATBoost - Optimization using optuna
auc:0.9 , log_loss:0.9651209210448068 , ks:0.7142857142857143

LGBM_model = train(X_train, y_train, X_test, y_test, balanced='
    balanced', method='LGBM')
Trial 4753 finished with value: 0.8214285714285714 and
    parameters: {'learning_rate': 0.04666777939260228, '
    num_leaves': 150, 'lambda_l1': 2.0584202047356643e-06, '
    lambda_l2': 2.6949988611136263e-08, 'min_data_in_leaf': 7, '
    max_depth': 62, 'feature_fraction': 0.7016619725081281, '
    bagging_fraction': 0.6273888361729247, 'bagging_freq': 7}.
    Best is trial 426 with value: 0.8857142857142858.
```

```
18: ...
19: Trial 426 finished with value: 0.8857142857142858 and
      parameters: {'learning_rate': 0.08959873811059711, '
      num_leaves': 186, 'lambda_l1': 3.78113915153367e-08, '
      lambda_l2': 5.286213387951746e-08, 'min_data_in_leaf': 11, '
      max_depth': 23, 'feature_fraction': 0.5167806798611404, '
      bagging_fraction': 0.6306577516857322, 'bagging_freq': 7}.
      Best is trial 426 with value: 0.8857142857142858.
20: LGBM - Optimization using optuna
21: auc:0.8857142857142858 , log_loss:0.43692623324540775 , ks
      :0.7285714285714286
```

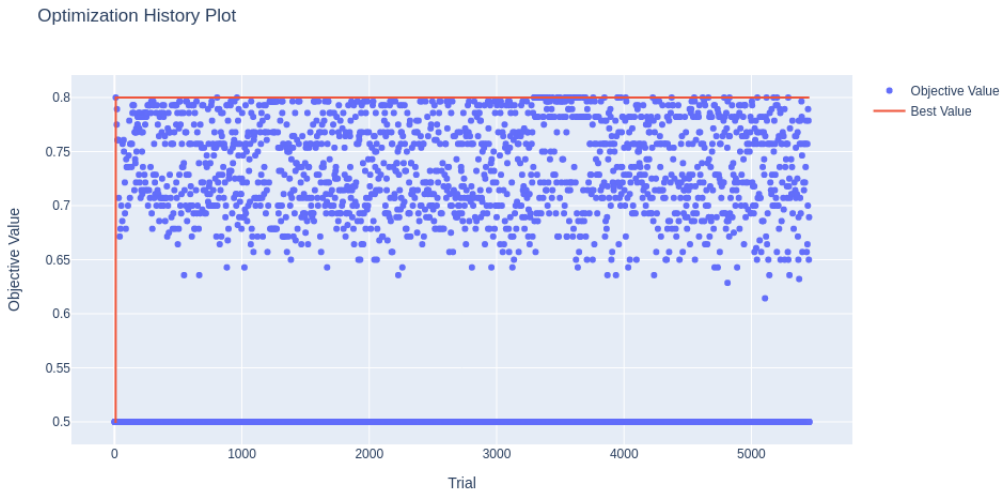
| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.80000 | 0.90000 | 0.88571 |
| **logloss** | 0.57024 | 0.96512 | 0.43693 |
| **KS** | 0.51429 | 0.71429 | 0.72857 |

Tabela 37 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros otimizados pelo Optuna no conjunto de dados de Insuficiência Renal.

### 5.3.2 Resultados do estudo do Optuna no conjunto de dados de Insuficiência Renal utilizando o XGBoost.

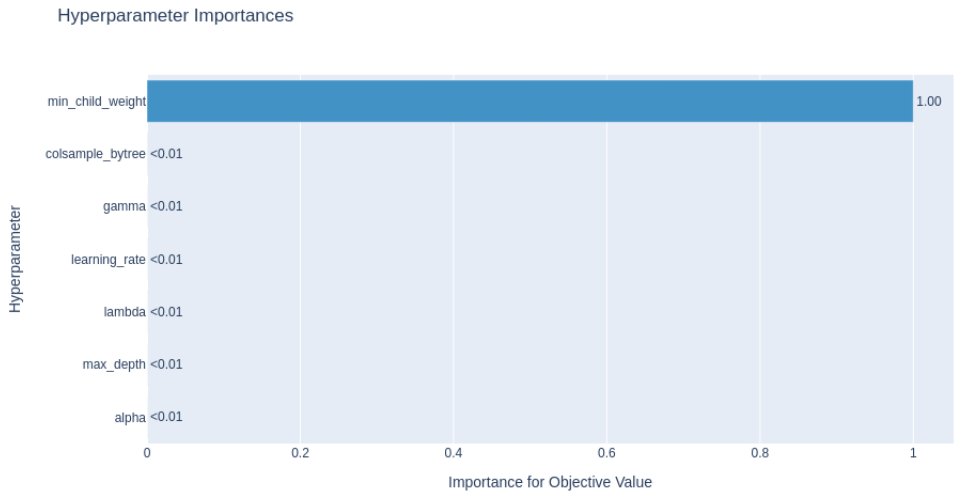
Novamente, vamos analisar os resultados do estudo do Optuna.

Figura 43 – Valores do estudo do XGBoost no conjunto de dados de Insuficiência Renal pelo Optuna.




A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

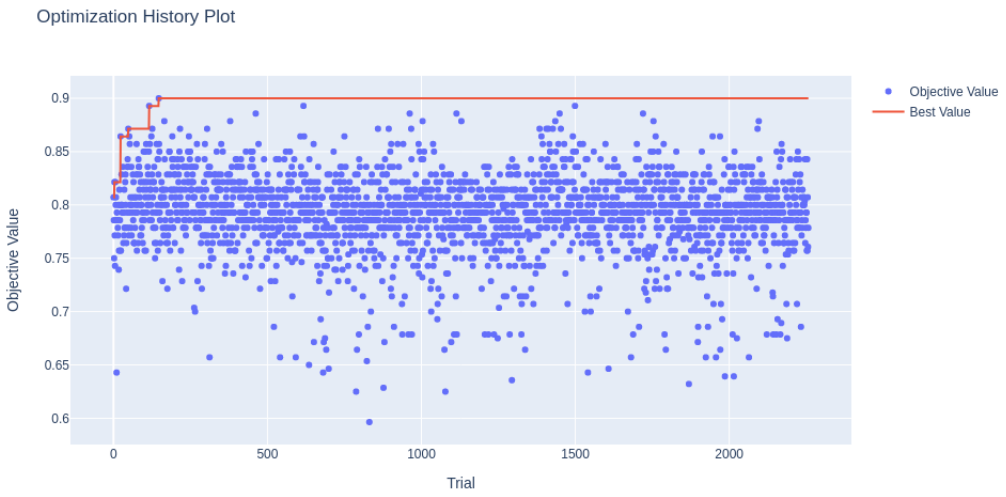
Figura 44 – Hiperparâmetros do XGBoost com maior importância no Optuna no dados de Insuficiência Renal.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foi o *min_child_weight*.

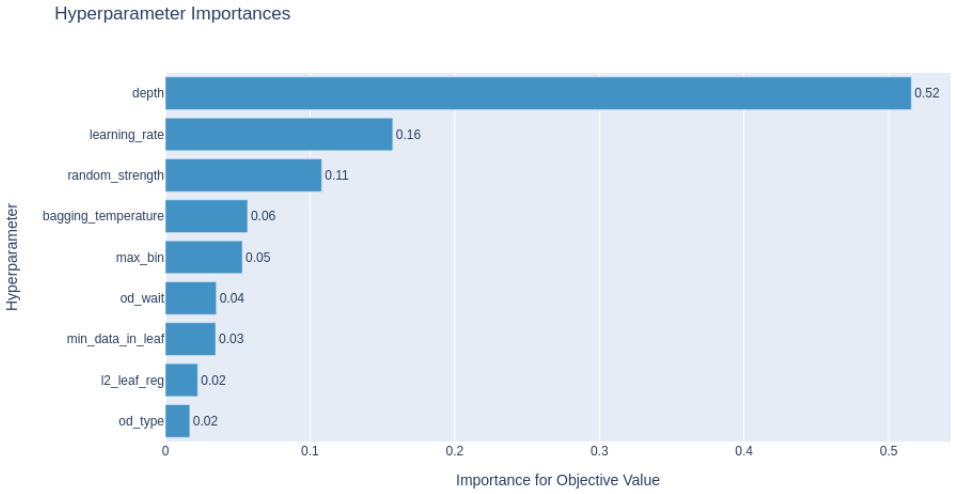
### 5.3.3 Resultados do estudo do Optuna no conjunto de dados de Insuficiência Renal utilizando o CatBoost.

Figura 45 – Valores do estudo do CatBoost no conjunto de dados de Insuficiência Renal pelo Optuna.




A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

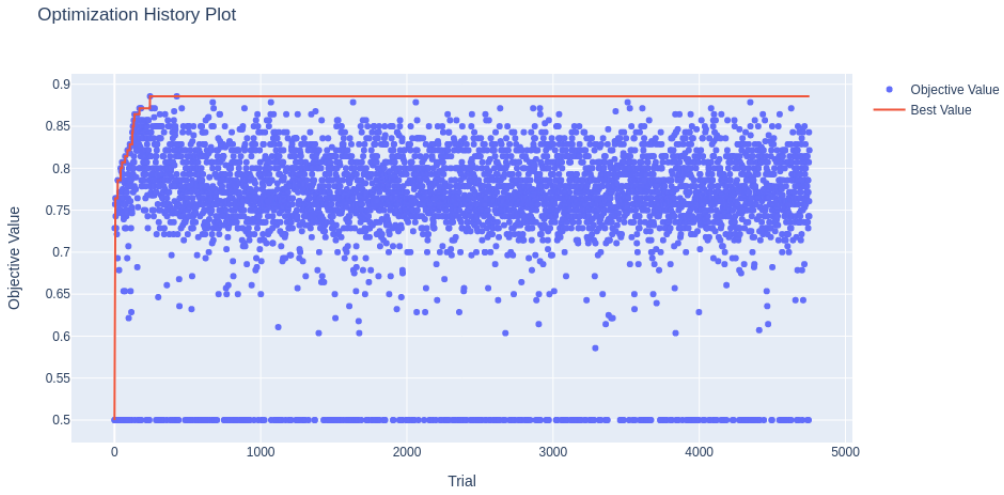
Figura 46 – Hiperparâmetros do XGBoost com maior importância no Optuna no dados de Insuficiência Renal.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foram o *depth* e o *learning_rate*.

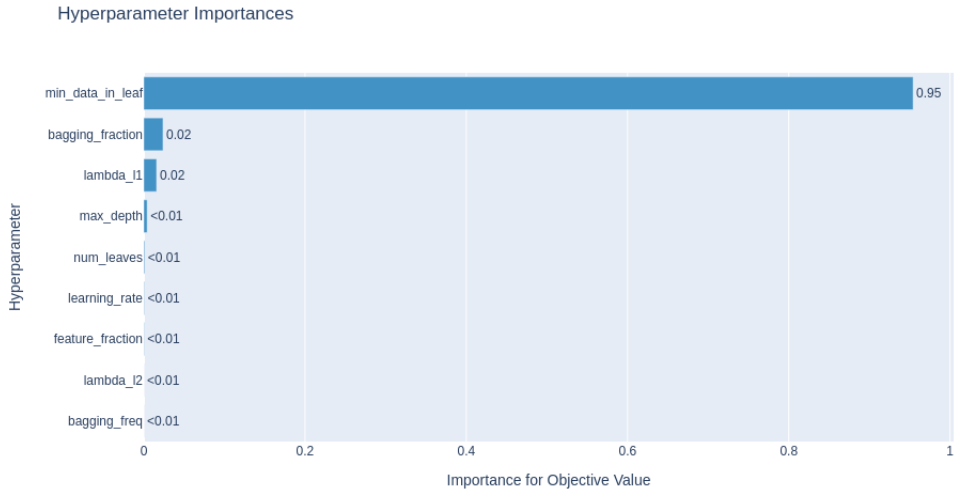
### 5.3.4 Resultados do estudo do Optuna no conjunto de dados de Insuficiência Renal utilizando o LightGBM.

Figura 47 – Valores do estudo do LightGBM no conjunto de dados de Insuficiência Renal pelo Optuna.




A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

Figura 48 – Hiperparâmetros do LightGBM com maior importância no Optuna no dados de Insuficiência Renal.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foi o *min_data_in_leaf*.

## 5.4 Resultados Conjunto de dados de Carcinoma de Mama

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| **AUC** | 0.97950 | 0.97156 | 0.94511 |
| **logloss** | 0.60595 | 0.80793 | 1.81785 |
| **KS** | 0.95899 | 0.94312 | 0.89021 |
| **tempo (s)** | 0.14904 | 2.72823 | 0.08682 |

Tabela 38 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros *Default* no conjunto de dados de Carcinoma de Mama.

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| **AUC** | 0.96693 | 0.95899 | 0.95106 |
| **logloss** | 1.00992 | 1.21190 | 1.41388 |
| **KS** | 0.93386 | 0.91799 | 0.90212 |
| **tempo (s)** | 0.08013 | 0.81609 | 0.10254 |

Tabela 39 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de Carcinoma de Mama.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.96693 | 0.95899 | 0.95106 |
| **logloss** | 1.00992 | 1.21190 | 1.41388 |
| **KS** | 0.93386 | 0.91799 | 0.90212 |
| **tempo (s)** | 0.08013 | 0.81609 | 0.10254 |

Tabela 40 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de Carcinoma de Mama.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.97487 | 0.95437 | 0.95106 |
| **logloss** | 0.80793 | 1.41388 | 1.41388 |
| **KS** | 0.94974 | 0.90873 | 0.90212 |
| **tempo (s)** | 0.10812 | 0.78369 | 0.07552 |

Tabela 41 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 3$, $Reg_L = 0.01$ no conjunto de dados de Carcinoma de Mama.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.96362 | 0.97156 | 0.95899 |
| **logloss** | 1.00991 | 0.80793 | 1.21190 |
| **KS** | 0.92725 | 0.94312 | 0.91799 |
| **tempo (s)** | 0.09410 | 3.08160 | 0.05303 |

Tabela 42 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 6$, $Reg_L = 0.01$ no conjunto de dados de Carcinoma de Mama.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.95899 | 0.96362 | 0.95106 |
| **logloss** | 1.21190 | 1.00991 | 1.41388 |
| **KS** | 0.91799 | 0.92725 | 0.90212 |
| **tempo (s)** | 0.09263 | 0.77307 | 0.08819 |

Tabela 43 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.1$, $MD = 3$, $Reg_L = 0.05$ no conjunto de dados de Carcinoma de Mama.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.97156 | 0.95899 | 0.95437 |
| **logloss** | 0.80793 | 1.21190 | 1.41388 |
| **KS** | 0.94312 | 0.91799 | 0.90873 |
| **tempo (s)** | 0.08900 | 2.68093 | 0.07608 |

Tabela 44 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.01$ no conjunto de dados de Carcinoma de Mama.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.97156 | 0.97156 | 0.06191 |
| **logloss** | 0.80793 | 0.80793 | 0.95899 |
| **KS** | 0.94312 | 0.94312 | 1.21190 |
| **tempo (s)** | 0.07007 | 2.72953 | 0.91799 |

Tabela 45 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 6$, $Reg_L = 0.05$ no conjunto de dados de Carcinoma de Mama.

| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.96693 | 0.95437 | 0.95106 |
| **logloss** | 1.00992 | 1.41388 | 1.41388 |
| **KS** | 0.93386 | 0.90873 | 0.90212 |
| **tempo (s)** | 0.05206 | 0.83468 | 0.09816 |

Tabela 46 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros $LR = 0.3$, $MD = 3$, $Reg_L = 0.05$ no conjunto de dados de Carcinoma de Mama.

### 5.4.1 Optuna no Conjunto de dados de Carcinoma de Mama

No final da execução o Optuna nos retorna a quantidade de *trials* e qual obteve a melhor performance. Abaixo temos os códigos com as melhores saídas do Optuna de cada modelo.

**Código-fonte 10** – Resultado do Optuna no conjunto de dados de Carcinoma de Mama.

```
1: XGBoost_model = train(X_train, y_train, X_test, y_test,
     balanced='balanced', method='XGBoost')
2: Trial 4948 finished with value: 0.9972075249853027 and
     parameters: {'learning_rate': 0.05731935932209517, '
     max_depth': 11, 'min_child_weight': 6, 'gamma':
     3.4052496218231224e-05, 'alpha': 3.790159562548499e-08, '
     lambda': 0.37064484949542936, 'colsample_bytree':
     0.1368272730789507}. Best is trial 615 with value:
     0.9988242210464433.
3: ...
4: Trial 615 finished with value: 0.9988242210464433 and
     parameters: {'learning_rate': 0.0436689491255323, 'max_depth
     ': 14, 'min_child_weight': 15, 'gamma': 3.027611637978531e
     -07, 'alpha': 1.2966964481647062e-07, 'lambda':
     1.125613997027746, 'colsample_bytree': 0.13844700517507155}.
      Best is trial 615 with value: 0.9988242210464433.
5: XGBoost - Optimization using optuna
6: auc:0.9988242210464433 , log_loss:0.1163896211145217 , ks
     :0.9841269841269841
```

```
 7:
 8: CatBoost_model = train(X_train_cat, y_train_cat, X_test_cat,
      y_test_cat, balanced='balanced', method='CATBoost')
 9: Trial 350 finished with value: 0.9975014697236919 and
      parameters: {'learning_rate': 0.014693012954604871, 'depth':
       6, 'max_bin': 371, 'min_data_in_leaf': 128, 'l2_leaf_reg':
      1.6337878324887535e-05, 'random_strength':
      5.267279748486068, 'bagging_temperature': 8.30501182209448,
      'od_type': 'Iter', 'od_wait': 31}. Best is trial 71 with
      value: 0.9992651381540271.
10: ...
11: Trial 71 finished with value: 0.9992651381540271 and parameters
      : {'learning_rate': 0.006920317042180218, 'depth': 6, '
      max_bin': 375, 'min_data_in_leaf': 18, 'l2_leaf_reg':
      5.356718213907343e-06, 'random_strength': 7.0698333503761, '
      bagging_temperature': 8.330048758136687, 'od_type': 'Iter',
      'od_wait': 30}. Best is trial 71 with value:
      0.9992651381540271.
12: CATBoost - Optimization using optuna
13: auc:0.9992651381540271 , log_loss:0.044536185929194914 , ks
      :0.9748677248677248
14:
15: LGBM_model = train(X_train, y_train, X_test, y_test, balanced='
      balanced', method='LGBM')
16: Trial 4658 finished with value: 0.9961787184009405 and
      parameters: {'learning_rate': 0.0739594630318519, '
      num_leaves': 142, 'lambda_l1': 6.1038128339002564e-06, '
      lambda_l2': 1.0045566613521662e-08, 'min_data_in_leaf': 61,
      'max_depth': 9, 'feature_fraction': 0.41696597857129664, '
      bagging_fraction': 0.8138570308199493, 'bagging_freq': 4}.
      Best is trial 2418 with value: 0.9994121105232218.
17: ...
18: Trial 2418 finished with value: 0.9994121105232218 and
      parameters: {'learning_rate': 0.07615521372640538, '
      num_leaves': 219, 'lambda_l1': 2.74036247131309e-08, '
      lambda_l2': 1.7948089596435946e-07, 'min_data_in_leaf': 66,
      'max_depth': 18, 'feature_fraction': 0.4971540914007164, '
      bagging_fraction': 0.43488273051341403, 'bagging_freq': 4}.
      Best is trial 2418 with value: 0.9994121105232218.
19: LGBM - Optimization using optuna
20: auc:0.9994121105232218 , log_loss:0.06707615388690602 , ks
      :0.9841269841269841
```

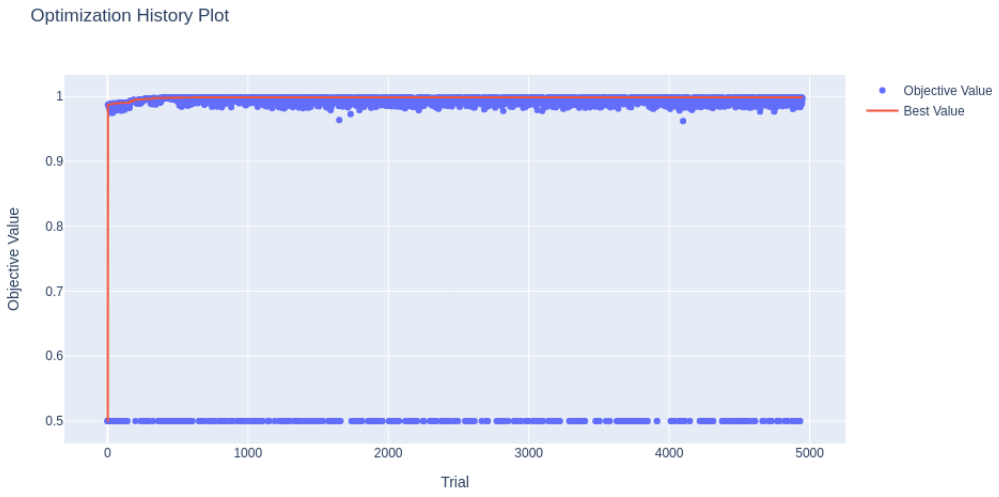
| | XGBoost | CatBoost | LightGBM |
|---|---|---|---|
| **AUC** | 0.99882 | 0.99927 | 0.99941 |
| **logloss** | 0.11639 | 0.04454 | 0.06708 |
| **KS** | 0.98413 | 0.97487 | 0.98413 |

Tabela 47 – Resultado do XGBoost, CatBoost e LGBM com os hiperparâmetros otimizados pelo Optuna no conjunto de dados de Carcinoma de Mama.

### 5.4.2 Resultados do estudo do Optuna no conjunto de dados de Carcinoma de Mama utilizando o XGBoost.

Novamente iremos analisar o estudo do Optuna.

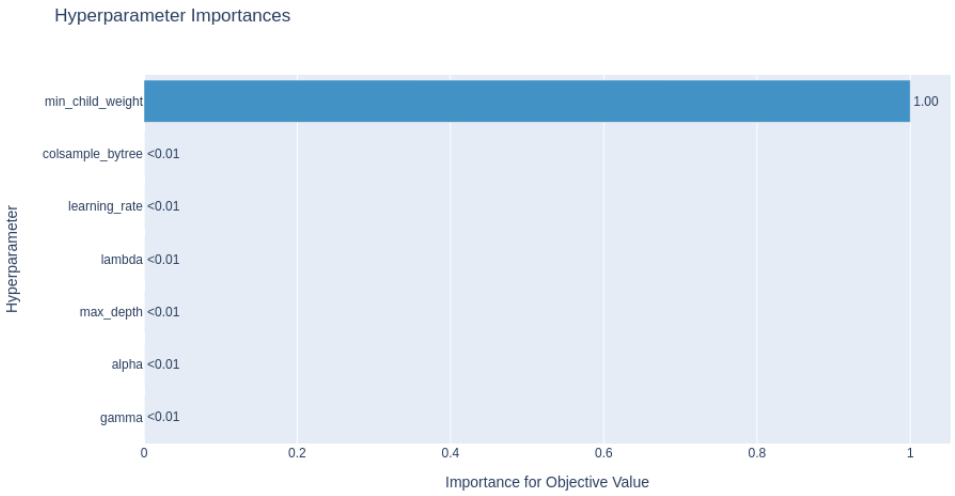
Figura 49 – Valores do estudo do XGBoost no conjunto de dados de Carcinoma de Mama pelo Optuna.




.

A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

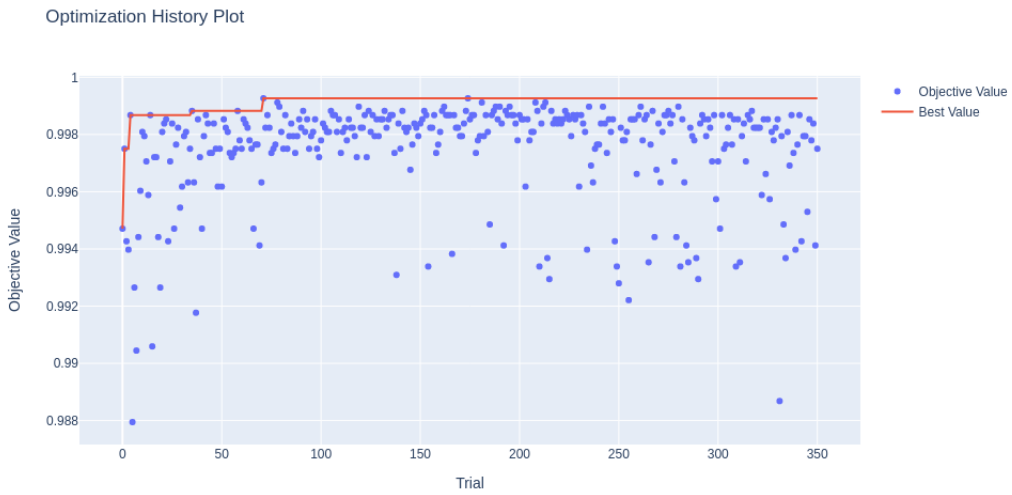
Figura 50 – Hiperparâmetros do XGBoost com maior importância no Optuna no dados de Carcinoma de Mama.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foi o *min_child_weight*.

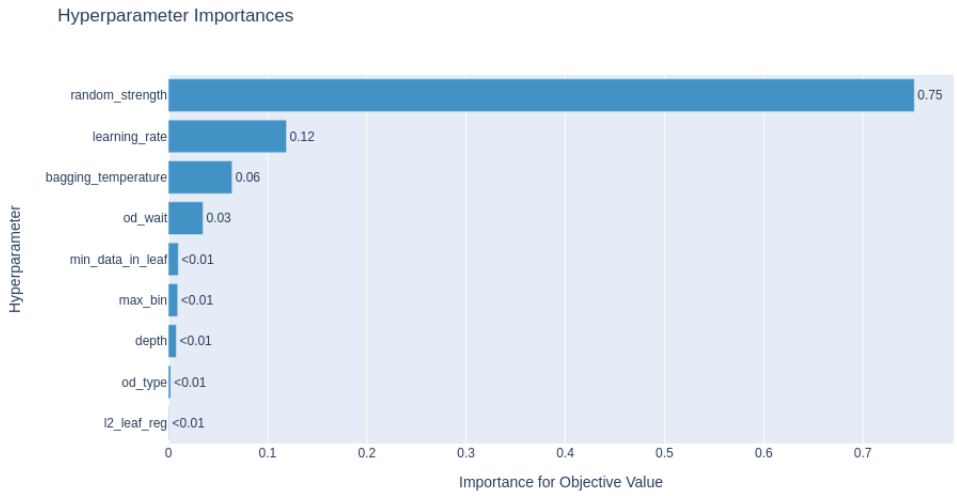
### 5.4.3 Resultados do estudo do Optuna no conjunto de dados de Carcinoma de Mama utilizando o CatBoost.

Figura 51 – Valores do estudo do CatBoost no conjunto de dados de Carcinoma de Mama pelo Optuna.




A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

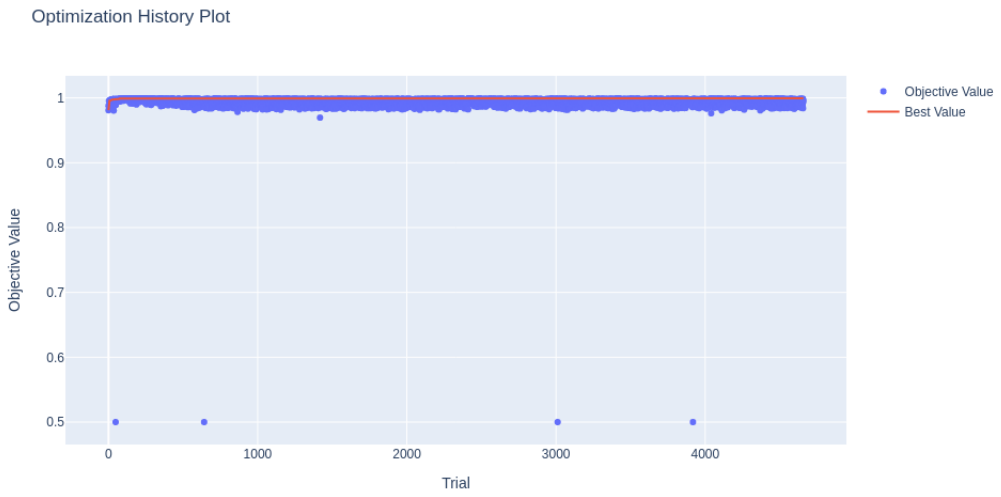
Figura 52 – Hiperparâmetros do CatBoost com maior importância no Optuna no dados de Carcinoma de Mama.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foram o *random_strength* e o *learning_rate*.

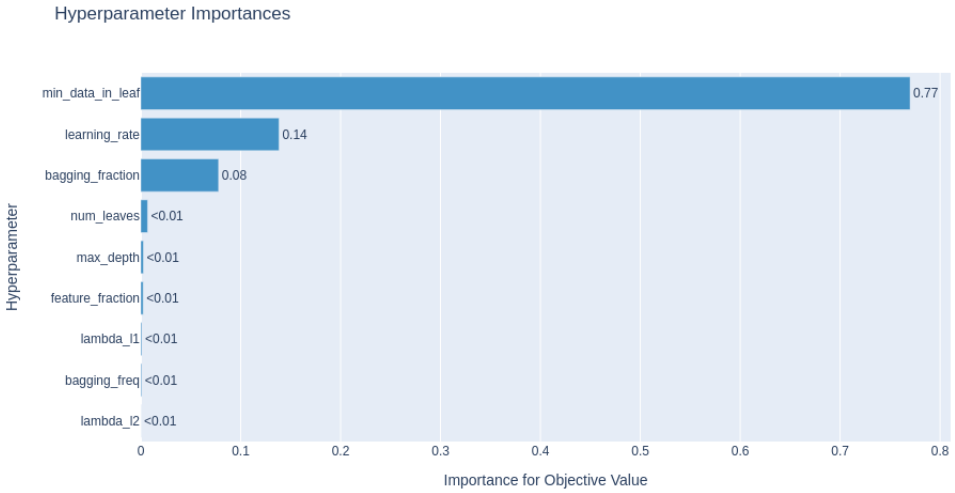
### 5.4.4 Resultados do estudo do Optuna no conjunto de dados de Carcinoma de Mama utilizando o LightGBM.

Figura 53 – Valores do estudo do LightGBM no conjunto de dados de Carcinoma de Mama pelo Optuna.




A partir do estudo de otimização do Optuna foi possível identificar quais os hiperparâmetros que possuem o maior impacto no tuning do Optuna.

Figura 54 – Hiperparâmetros do LightGBM com maior importância no Optuna no dados de Carcinoma de Mama.




Ou seja, podemos concluir que ao longo do estudo do Optuna o hiperparâmetro com maior importância no resultado final foram o *min_data_in_leaf* e o *learning_rate*.

CAPÍTULO

# 6

# RESULTADOS E DISCUSSÕES

Nesta secção iremos discutir os resultados obtidos pelos modelos no capítulo 5. O principal foco é consolidar um resumo de cada teste nos conjuntos de dados e analisar se o modelo final otimizado pelo Optuna teve um aumento de performance em relação aos modelos com os hiperparâmetros *Default*.

Nos testes iniciais a performance dos modelos foi variável, em alguns testes o XGBoost se destacou, outros o CatBoost e outros o LightGBM. Em termos de custo computacional, o CatBoost é o mais custoso enquanto que o LightGBM foi o teve a melhor performance do ponto de vista de execução computacional.

Esses resultados eram de fato esperados, conforme foi abordado no capítulo 3. O fato do CatBoost precisar lidar com as variáveis categóricas e a simetria da árvore o torna mais lento, enquanto o LightGBM foi criado justamente para ser mais rápido.

Nos primeiros testes não foi possível identificar o impacto do *LR* e do $Reg_L$ diretamente na performance dos modelos, o hiperparâmetro que ficou mais em evidência foi o *MD*. Na maioria dos casos, o seu aumento ocasionou um maior tempo de execução, o que de fato era esperado pois esse hiperparâmetro representa o crescimento máximo da profundidade da árvore.

Os principais resultados desse estudo são sobre o modelo final otimizado pelo Optuna, com as diversas tentativas do Optuna foi possível entender melhor como os hiperparâmetros funcionam no modelo e encontrar os hiperparâmetros com maior importancia. Conseguimos concluir a influencia desses hiperparâmetros e analisar se houve mudança deles ao longo dos conjunto de dados. Os resultados finais de cada modelo serão mostrados a seguir. A comparação será em cima dos modelos *Default* e dos finais.

Podemos perceber que, para todos os conjuntos de dados, tivemos aumento de performance nas principais métricas de validação. É importante mencionar que o conjunto de dados de Insuficiência Renal foi onde a otimização do Optuna se destacou e nos trouxe a maior perfor-

mance. Após isso, foi aplicado o SHAP para o melhor modelo de cada conjunto de dados, para que possamos interpretar algumas das principais variáveis e tornar o modelo menos 'caixa-preta'.

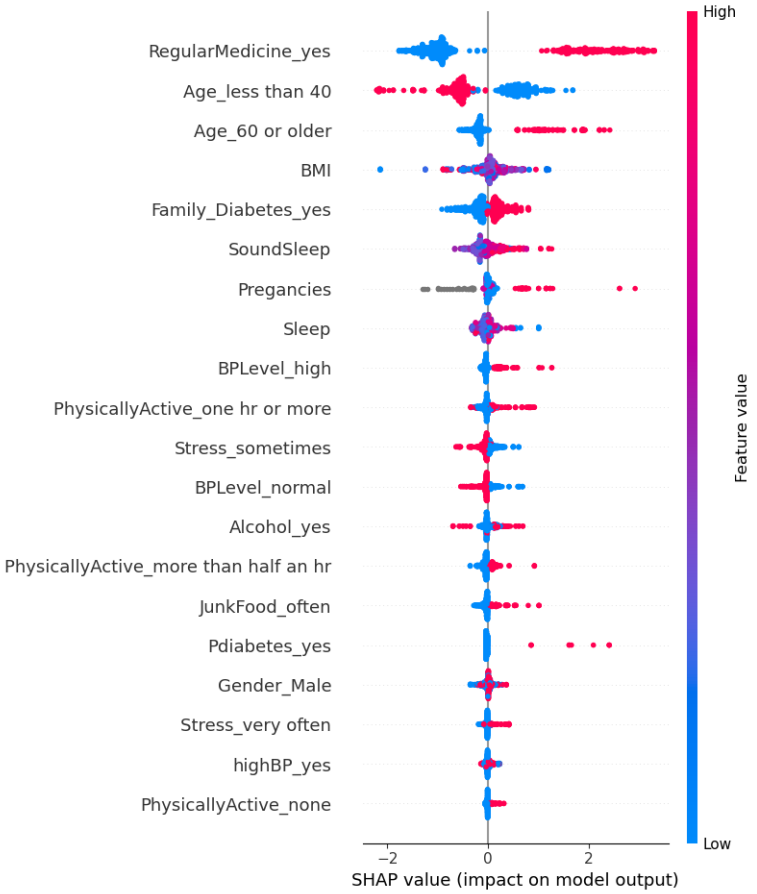
## 6.1 Resultados Finais Diabetes

A partir das tabelas 11 e 19 podemos calcular a diferença de performance em percentual.

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| $\delta_{AUC}$ | 4.22% | 9.43% | 6.52% |
| $\delta_{KS}$ | 0.62 | 10.70% | 4.98% |

Tabela 48 – Aumento final da performance do modelo otimizado pelo Optuna para o conjunto de dados de Diabetes.

Figura 55 – SHAP Values para o modelo LightGBM do conjunto de dados de Diabetes



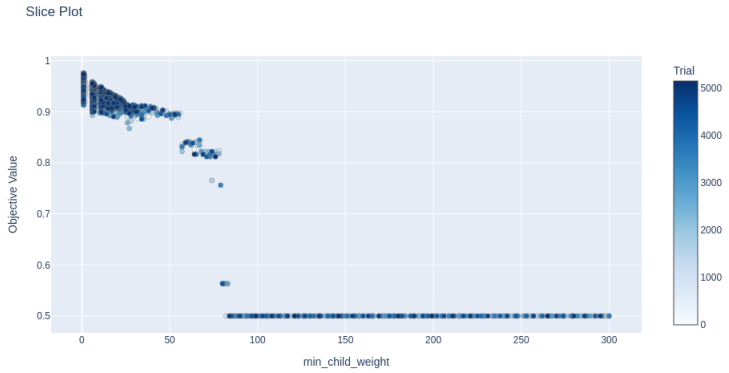

E no final, podemos identificar quais foram os hiperparâmetros com maior impacto em cada um dos modelos e quais foram seus valores.

| Modelo | Hiperparâmetros | |
|---|---|---|
| **XGBoost** | 'min_child_weight': 1 | |
| **CatBoost** | 'depth': 10 | 'learning_rate': 0.005585552379158199 |
| **LightGBM** | 'lambda_l1': 2.7481689793447196e-06 | learning_rate': 0.08425779644832665 |

Tabela 49 – Valores finais dos hiperparâmetros com maiores impactos em cada modelo no conjunto de dado de Diabetes.

Para o XGBoost, o modelo otimizado pelo optuna tem o hiperparâmetro 'min_child_-weight': 1, e como esse hiperparâmetro é o com maior importância, podemos analisar o seu comportamento ao longo do estudo.

Figura 56 – Hiperparâmetros *min_child_weight* do XGBoost no estudo do Optuna no conjunto de dados de Diabetes.

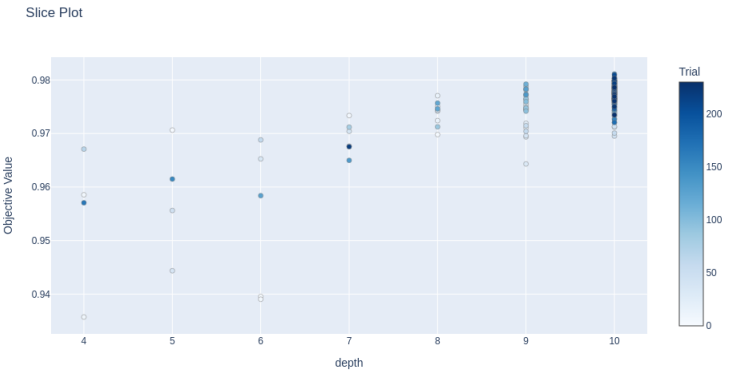



.

Claramente a figura 56 nos mostra que existe uma relação de performance com *min_-child_weight*: para valores altos do hiperparâmetro a performance cai drasticamente e os melhores valores estão próximos do 0.

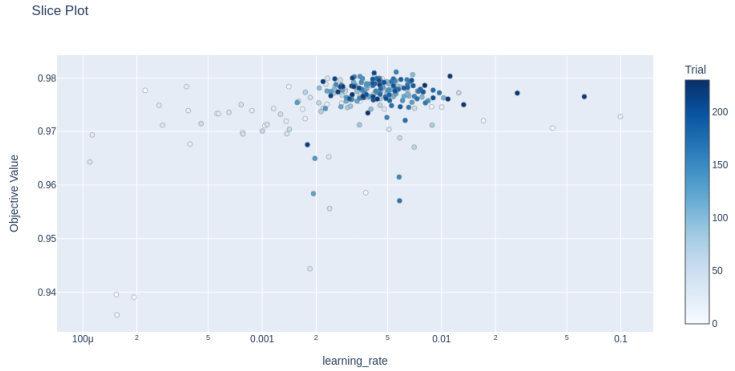
Para o CatBoost, o modelo otimizado pelo optuna tem os hiperparâmetros 'depth': 10 e o 'learning_rate': 0.005585552379158199, e como esses hiperparâmetros são os que possuem as maiores importância no estudo, podemos analisar os seus impactos.

Figura 57 – Hiperparâmetros *depth* do CatBoost no estudo do Optuna no conjunto de dados de Diabetes.




.

Figura 58 – Hiperparâmetros *learning_rate* do CatBoost no estudo do Optuna no conjunto de dados de Diabetes.

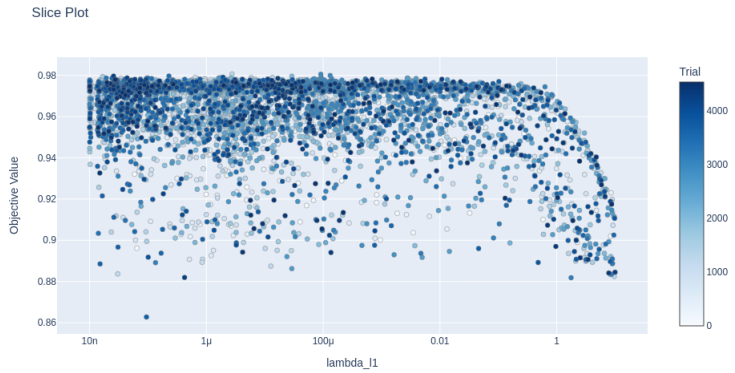



.

A figura 57 nos mostra que existe uma relação de performance com o crescimento do *depth*, com os melhores valores e concentranto no 10. Enquanto que a figura 58 o *learning_rate* tem uma melhor performance na região de 0.001 até 0.01.

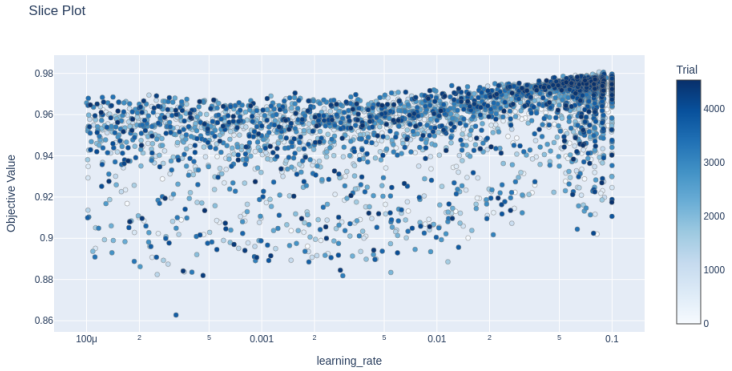
No LightGBM, o modelo otimizado pelo optuna tem os hiperparâmetros 'lambda_l1': 2.7481689793447196e-06 e o learning_rate': 0.08425779644832665, e como esses hiperparâmetros são os mais importantes no estudo, podemos analisar os seus impactos.

Figura 59 – Hiperparâmetros *lambda_l1* do LightGBM no estudo do Optuna no conjunto de dados de Diabetes.




.

Figura 60 – Hiperparâmetros *learning_rate* do LightGBM no estudo do Optuna no conjunto de dados de Diabetes.




.

Na figura 59, podemos perceber que começa uma leve queda na performance a partir de valores *lambda_l1* maiores que 0.01, ficando mais evidente com valores próximo de 1. E na figura 60 existe uma relação de crescimento de performance conforme os valores de *learning_rate* vão aumentando, ficando claro que a região com maior performance está perto de *learning_rate* com o valor de 0.1.

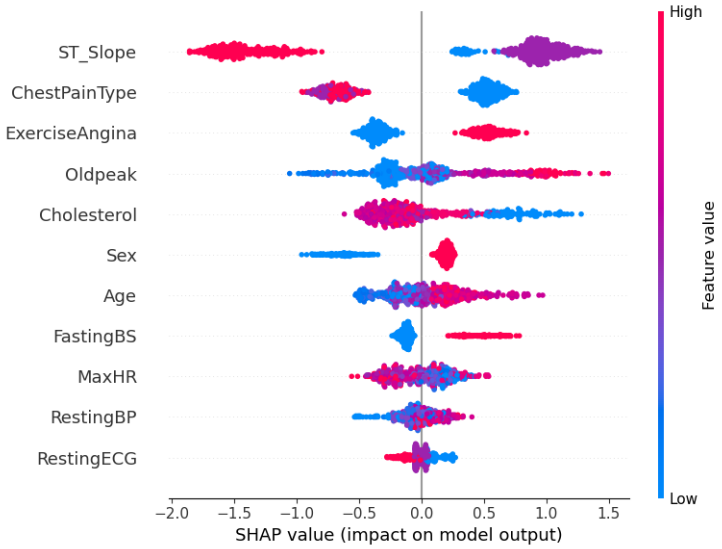
## 6.2 Resultados Finais Insuficiência Cardíaca

A partir das tabelas 20 e 28 podemos calcular a diferença de performance em percentual.

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| $\delta_{AUC}$ | 12.51% | 5.52% | 11.06% |
| $\delta_{KS}$ | 16.80% | -1.77% | 11.52% |

Tabela 50 – Aumento final da performance do modelo otimizado pelo Optuna para o conjunto de dados de Insuficiência Cardíaca.

Figura 61 – SHAP Values para o modelo XGBoost do conjunto de dados de Insuficiência Cardíaca



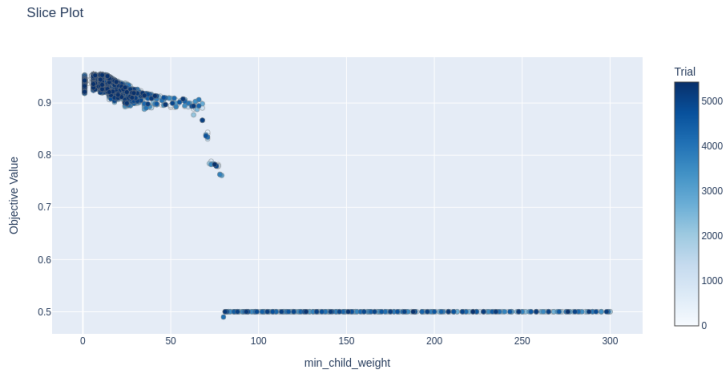

E no final, podemos identificar quais foram os hiperparâmetros com maior impacto em cada um dos modelos e quais foram seus valores.

| **Modelo** | **Hiperparâmetros** | |
|---|---|---|
| **XGBoost** | 'min_child_weight': 6 | |
| **CatBoost** | 'learning_rate': 0.006239961585898258 | 'l2_leaf_reg': 9.62802219566606 |
| **LightGBM** | 'learning_rate': 0.0765705960307223 | 'min_data_in_leaf': 26 |

Tabela 51 – Valores finais dos hiperparâmetros com maiores impactos em cada modelo no conjunto de dado de Insuficiência Cardíaca.

Para o XGBoost novamente o *min_child_weight* se destacou, mostrando que para valores menores que 50 é onde estão as melhores performances do modelo.

Figura 62 – Hiperparâmetros *min_child_weight* do XGBoost no estudo do Optuna no conjunto de dados de Insuficiência Cardíaca.



Para o CatBoost temos *learning_rate* e o *l2_leaf_reg* com as maiores importâncias no estudo.

Figura 63 – Hiperparâmetros *learning_rate* do CatBoost no estudo do Optuna no conjunto de dados de Insuficiência Cardíaca.

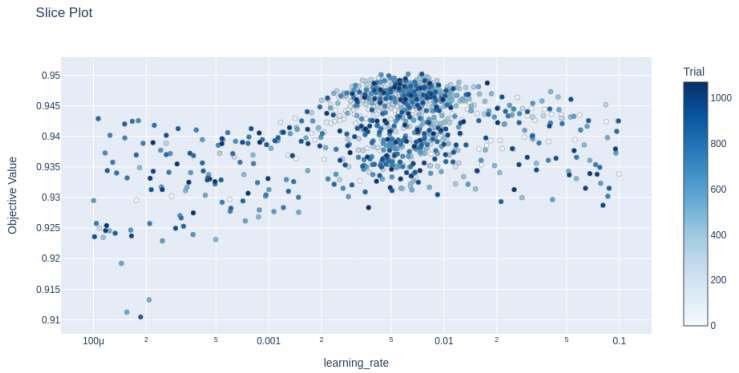
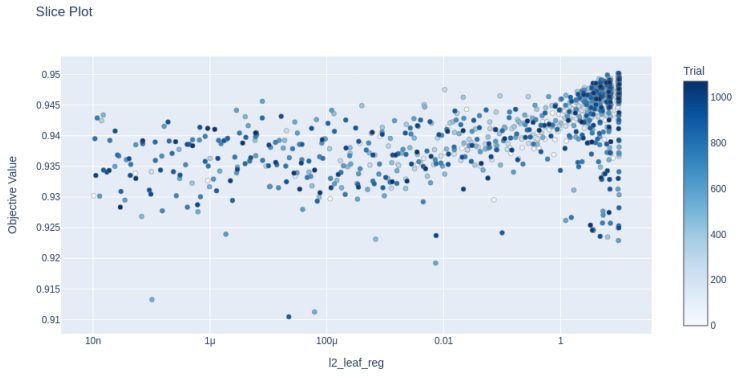



Figura 64 – Hiperparâmetros *l2_leaf_reg* do CatBoost no estudo do Optuna no conjunto de dados de Insuficiência Cardíaca.




Podemos concluir que, para o *learning_rate*, a região com a melhor performance está entre 0.005 e 0.01 e para *l2_leaf_reg* existe uma relação de aumento de performance com o aumento desse hiperparâmetro, sendo a região com maior performance com *l2_leaf_reg* acima de 1.

Para o LightLGBM temos o *learning_rate* e o *min_data_in_leaf* que somados representam 70% da importancia do estudo.

Figura 65 – Hiperparâmetros *learning_rate* do LightGBM no estudo do Optuna no conjunto de dados de Insuficiência Cardíaca.

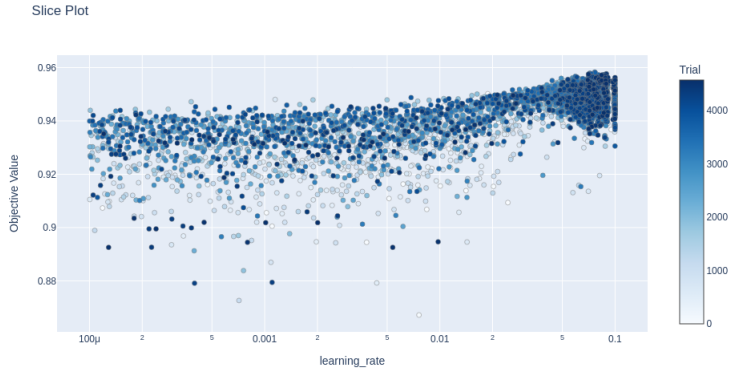
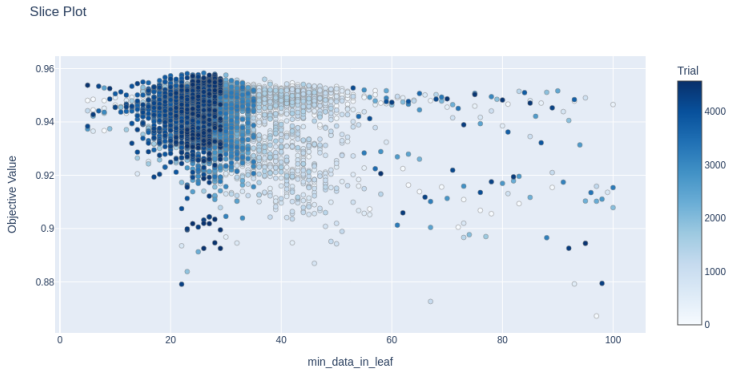



Figura 66 – Hiperparâmetros *min_data_in_leaf* do LightGBM no estudo do Optuna no conjunto de dados de Insuficiência Cardíaca.




Conforme o *learning_rate* vai aumentado, a perfomance do estudo também aumenta sendo que o melhor valor está próximo de *learning_rate* com 0.1. E para *min_data_in_leaf* conseguimos perceber que valores abaixo de 80 já temos uma boa performance.

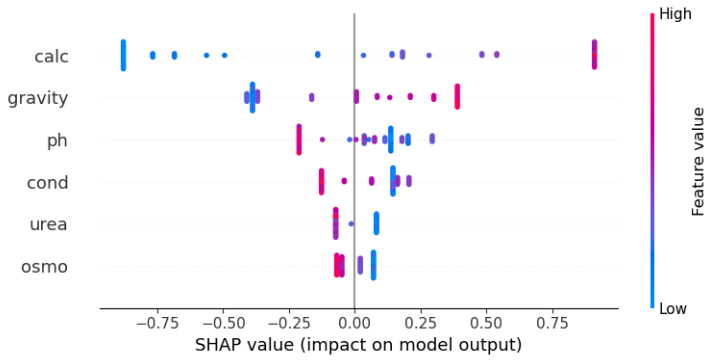
## 6.3 Resultados Finais Insuficiência Renal

A partir das tabelas 29 e 37 podemos calcular a diferença de performance em percentual.

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| $\delta_{AUC}$ | 31.76% | 23.53% | 21.57% |
| $\delta_{KS}$ | 140.00% | 56.26% | 59.37% |

Tabela 52 – Aumento final da performance do modelo otimizado pelo Optuna para o conjunto de dados de Insuficiência Renal.

Figura 67 – SHAP Values para o modelo CatBoost do conjunto de dados de Insuficiência Renal



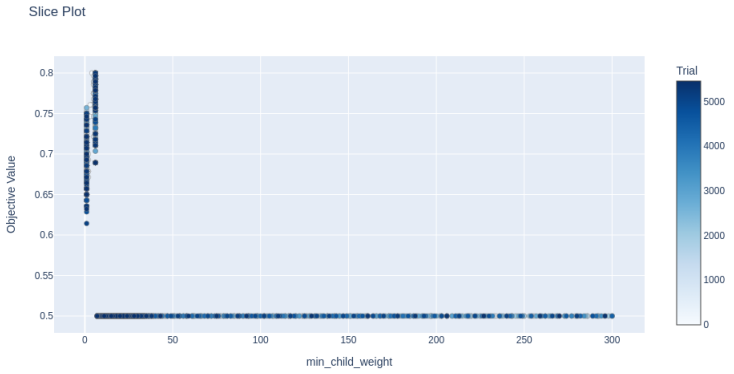

E no final, podemos identificar quais foram os hiperparâmetros com maior impacto em cada um dos modelos e quais foram seus valores.

| Modelo | Hiperparâmetros | |
|---|---|---|
| **XGBoost** | 'min_child_weight': 4 | |
| **CatBoost** | 'depth': 5 | 'learning_rate': 0.07537894328903638 |
| **LightGBM** | 'min_data_in_leaf': 7 | |

Tabela 53 – Valores finais dos hiperparâmetros com maiores impactos em cada modelo no conjunto de dado de Insuficiência Renal.

Para o XGBoost o hiperparâmetros com maior impacto foi novamente o *min_child_weight*

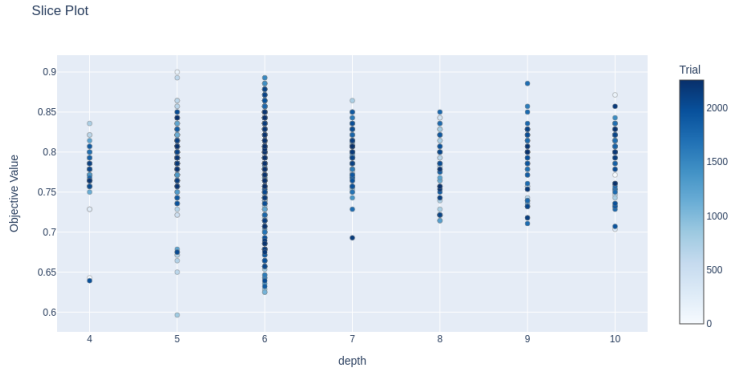
Figura 68 – Hiperparâmetros *min_child_weight* do XGBoost no estudo do Optuna no conjunto de dados de Insuficiência Renal.




E podemos perceber que a performance fica satisfatória para valores de *min_child_weight* abaixo de 10.

Para o CatBoost os hiperparâmetros que representaram quase 80% da importância foram o *depth* e o *learning_rate*.

Figura 69 – Hiperparâmetros *depth* do CatBoost no estudo do Optuna no conjunto de dados de Insuficiência Renal.



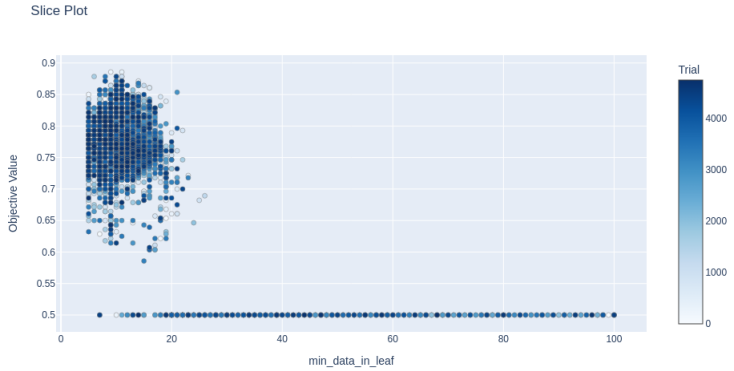

.

Figura 70 – Hiperparâmetros *learning_rate* do CatBoost no estudo do Optuna no conjunto de dados de Insuficiência Renal.


Pela figura 69 percebe-se que os melhores valores se concentram em 5, 6 e 9 para *depth*. E na figura 70 os maiores valores de *learning_rate*, próximo a 0.1, são os que apresentaram a melhor performance.

Para o LightGBM o *min_data_in_leaf* apresentou 95% da importância.

Figura 71 – Hiperparâmetros do LightGBM no estudo do Optuna no conjunto de dados de Insuficiência Renal.




Sendo que valores abaixo de 20 obtiverem as melhores performances.

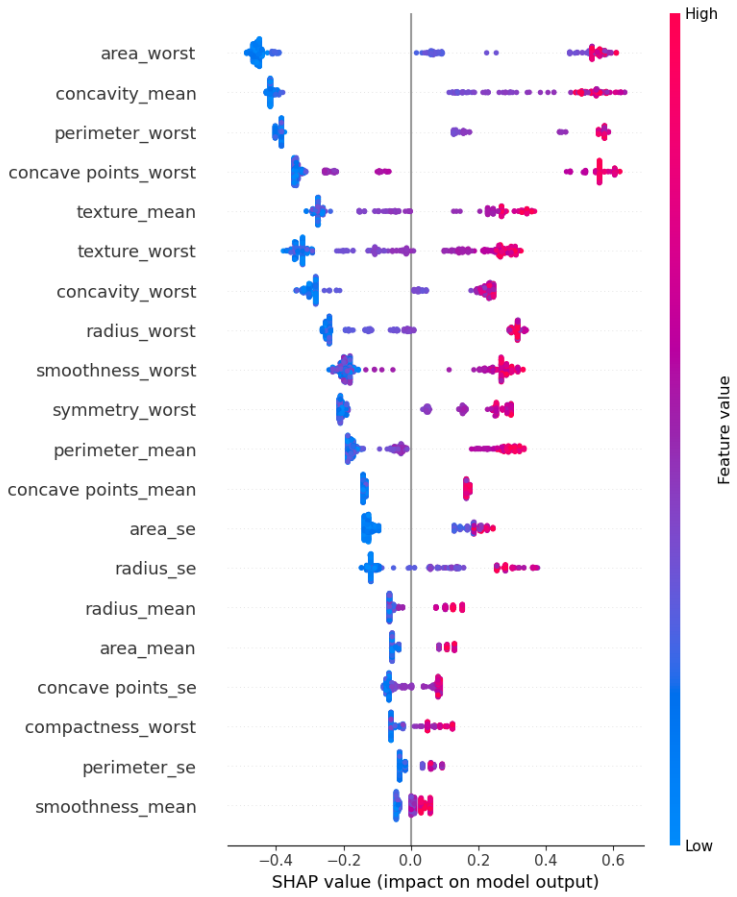
## 6.4 Resultados Finais Carcinoma de Mama

A partir das tabelas 38 e 47, podemos calcular a diferença de performance em percentual.

| | **XGBoost** | **CatBoost** | **LightGBM** |
|---|---|---|---|
| $\delta_{AUC}$ | 1.97% | 2.85% | 5.75% |
| $\delta_{KS}$ | 2.62% | 3.37% | 10.55% |

Tabela 54 – Aumento final da performance do modelo otimizado pelo Optuna para o conjunto de dados de Carcinoma de Mama.

Figura 72 – SHAP Values para o modelo LightGBM do conjunto de dados de Carcinoma de Mama

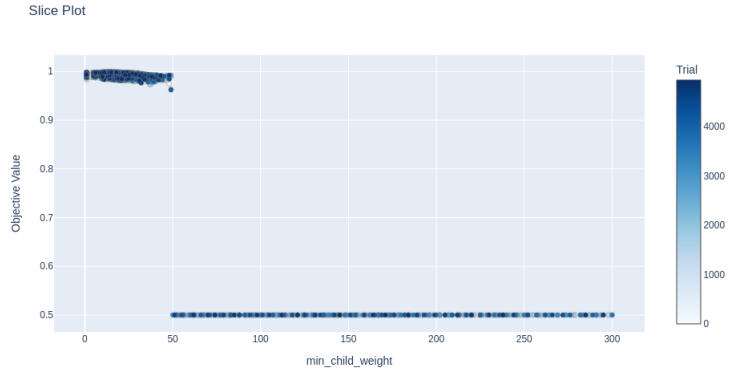



| **Modelo** | **Hiperparâmetros** | |
|---|---|---|
| **XGBoost** | 'min_child_weight': 15 | |
| **CatBoost** | 'random_strength': 5.267279748486068 | 'learning_rate': 0.014693012954604871 |
| **LightGBM** | 'min_data_in_leaf': 66 | 'learning_rate': 0.07615521372640538 |

Tabela 55 – Valores finais dos hiperparâmetros com maiores impactos em cada modelo no conjunto de dados de Carcinoma de Mama.

Para o XGBoost o hiperparâmetro *min_child_weight* teve a maior importância.

Figura 73 – Hiperparâmetro *min_child_weight* do XGBoost no estudo do Optuna no conjunto de dados de Carcinoma de Mama.




Então para valores abaixo de 50 o modelo teve as melhores performances.

Para o CatBoost os hiperparâmetros *random_strength* e o *learning_rate* representam 87% da importância.

Figura 74 – Hiperparâmetros *random_strength* do CatBoost no estudo do Optuna no conjunto de dados de Carcinoma de Mama.

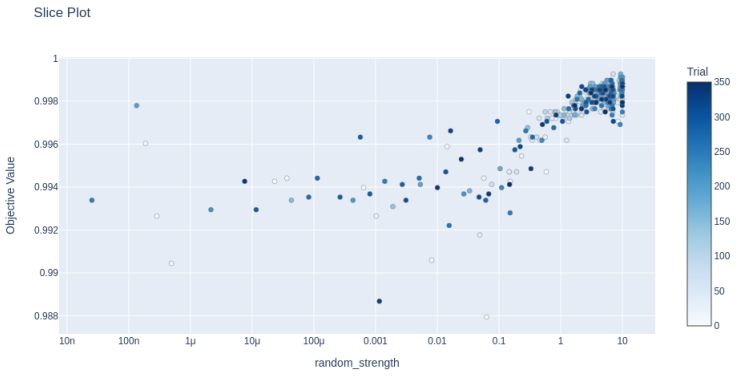
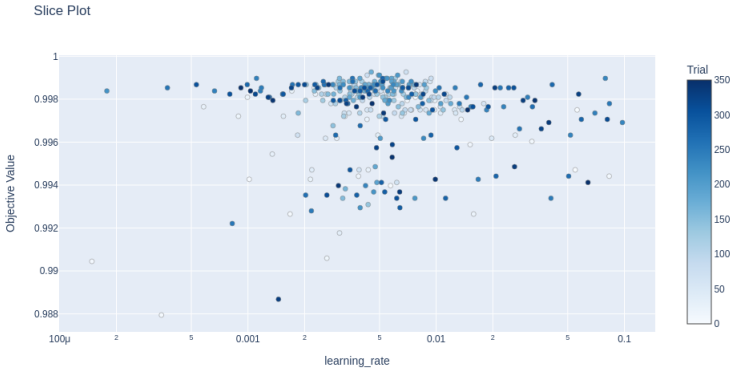



Figura 75 – Hiperparâmetros *learning_rate* do CatBoost no estudo do Optuna no conjunto de dados de Carcinoma de Mama.



Pela figura 74, quanto maior o *random_strength* melhor a perfomance do modelo, sendo a região entre 5 e 10 ideal. E na figura 75 podemos perceber que o *learning_rate* entre 0.001 até 0.1 teve bons resultados, sendo essa uma boa região para escolher.

Para o LightGBM os hiperparâmetros *min_data_in_leaf* e *learning_rate* representaram 90% da importância.

Figura 76 – Hiperparâmetros *min_data_in_leaf* do LightGBM no estudo do Optuna no conjunto de dados de Carcinoma de Mama.

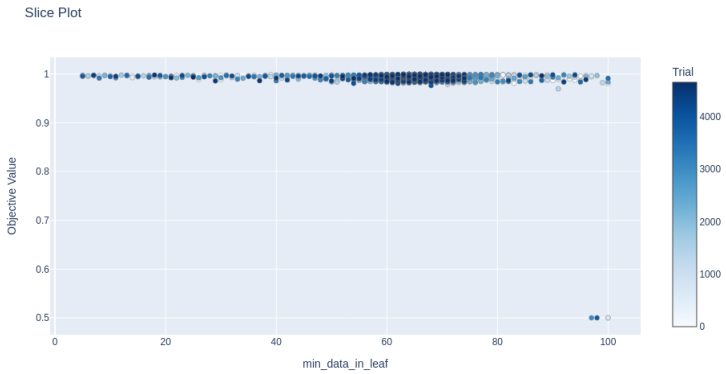
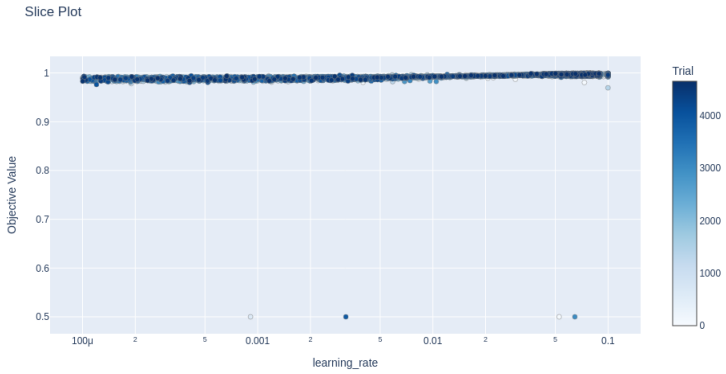



Figura 77 – Hiperparâmetros *learning_rate* do LightGBM no estudo do Optuna no conjunto de dados de Carcinoma de Mama.




Na figura 76 podemos perceber que não é possível concluir uma região ótima para o *min_data_in_leaf* nem para o *learning_rate*. Isso aconteceu pelo fato do modelo já ter conseguido ótimas performances iniciais e talvez o espaço dos Hiperparâmetros do LightGBM devesse ter sido divido em mais intervalos.

CAPÍTULO

# 7

# CONCLUSÃO

Neste trabalho foi possível entender os fundamentos de aprendizado supervisionado, as etapas da construção de um modelo de aprendizado não supervisionado, o problema de classificação, as métricas de validação de performance, os hiperparâmetros e a interpretabilidade do modelo.

Conseguimos entender a diferença entre os algoritmos XGBoost, CatBoost e LightGBM e aplicar suas implementações em conjuntos de dados reais da medicina. Foi feita uma analise descritiva dos conjuntos de dados e aplicado as transformações para correção dos dados e/ou transformação das variáveis categóricas para numéricas para os modelos de XGBoost e LightGBM.

No final, foi possível entender o impacto dos hiperparâmetros no modelo através do Optuna. Foi possível encontrar quais hiperparâmetros tiveram a maior importância no estudo nos diferentes conjunto de dados. Foi criada uma função sólida em *.py* para o treino, otimização e estudo do Optuna, sendo possível utilizar nos três algoritmos sem nenhum problema. Analisando a principal métrica de validação, **AUC**, tivemos um valor de ganho de performance para os modelos tunados pelo Optuna em relação aos modelos com os hiperparâmetros *Default* entre 5% e 10% para os conjunto de dados de Diabetes, Insuficiência Cardíaca e Carcinoma de Mama. Vale ressaltar que, para o modelo mais complexo de prever, Insuficiência Renal, tivemos ganhos de 20-30%. E no final, conseguimos interpretar como cada variável funcionou no modelo pelo SHAP, o que é extremamente importante para conseguir explicar o modelo.

## 7.1 Limitações e Trabalhos Futuros

Uma das principais limitações deste trabalho foi o tempo de execução. Treinar os modelos pelo Optuna e para um grande espaço de hiperparâmetros levou tempo. Foi possível utilizar uma escala logarítmica para alguns hiperparâmetros para gerar resultados com maiores diferenças de

magnitude, então para futuros trabalhos a sugestão é aumentar a escala dos hiperparâmetros e estudar mais casos que o Optuna nos retornou com um número maior de conjunto de dados.

# REFERÊNCIAS

AKIBA, T.; SANO, S.; YANASE, T.; OHTA, T.; KOYAMA, M. Optuna: A next-generation hyperparameter optimization framework. In: **Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2019. Citado nas páginas 27 e 71.

ASHWIN, L. Data streaming algorithms for the kolmogorov-smirnov test. p. 95–104, 2015. Citado nas páginas 17 e 53.

BROWN, C. D.; DAVIS, H. T. Receiver operating characteristics curves and related decision measures: A tutorial. **Chemometrics and Intelligent Laboratory Systems**, v. 80, n. 1, p. 24–38, 2006. ISSN 0169-7439. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169743905000766>. Citado nas páginas 17 e 48.

BUSSAB, W. de O.; MORETTIN, P. A. **Estatística Básica**. 9. ed. [S.l.]: Saraiva, 2013. Citado na página 53.

CHEN, T.; GUESTRIN, C. **XGBoost documentation**. Disponível em: <https://xgboost.readthedocs.io/en/stable/>. Acesso em: 05/12/2022. Citado na página 66.

____. Xgboost: A scalable tree boosting system. **CoRR**, abs/1603.02754, 2016. Disponível em: <http://arxiv.org/abs/1603.02754>. Citado nas páginas 17 e 66.

DOROGUSH, A. V.; GULIN, A.; GUSEV, G.; KAZEEV, N.; PROKHORENKOVA, L. O.; VOROBEV, A. **CatBoost documentation**. Disponível em: <https://catboost.ai/en/docs/>. Acesso em: 05/12/2022. Citado na página 66.

____. Fighting biases with dynamic boosting. **CoRR**, abs/1706.09516, 2017. Disponível em: <http://arxiv.org/abs/1706.09516>. Citado na página 66.

E., R. D.; MCCLELLAND, J. L.; GROUP, P. R. **Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations**. The MIT Press, 1986. ISBN 9780262291408. Disponível em: <https://doi.org/10.7551/mitpress/5236.001.0001>. Citado na página 62.

E., S. R.; YOAV, F. A brief introduction to boosting. In: **Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. (IJCAI'99), p. 1401–1406. Citado na página 63.

ELSHAWI, R.; MAHER, M.; SAKR, S. **Automated Machine Learning: State-of-The-Art and Open Challenges**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1906.02287>. Citado na página 56.

FAWCETT, T. An introduction to roc analysis. **Pattern Recognition Letters**, v. 27, n. 8, p. 861–874, 2006. ISSN 0167-8655. ROC Analysis in Pattern Recognition. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016786550500303X>. Citado nas páginas 48 e 49.

FRIEDMAN, J. H. Greedy function approximation: A gradient boosting machine. **The Annals of Statistics**, Institute of Mathematical Statistics, v. 29, n. 5, p. 1189 – 1232, 2001. Disponível em: <https://doi.org/10.1214/aos/1013203451>. Citado nas páginas 61, 64 e 65.

FRIEDMAN, J. H.; STUETZLE, W. Projection pursuit regression. **Journal of the American Statistical Association**, [American Statistical Association, Taylor Francis, Ltd.], v. 76, n. 376, p. 817–823, 1981. ISSN 01621459. Disponível em: <http://www.jstor.org/stable/2287576>. Citado na página 61.

GLOSSARY, M. **State of Data Science and Machine Learning 2022**. 2022. Disponível em: <https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html>. Acesso em: 05/12/2022. Citado nas páginas 17 e 52.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. 1. ed. [S.l.]: The Mit Press, 2016. Citado nas páginas 27, 54, 59, 60 e 62.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. 2. ed. [S.l.]: Springer New York, NY, 2009. ISBN 978-0-387-84857-0. Citado nas páginas 17, 38, 39, 44, 45, 46, 47, 62, 63 e 65.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. **Automated Machine Learning:Methods, Systems, Challenges**. 1. ed. [S.l.]: Springer, 2019. Citado nas páginas 54, 55, 56, 57 e 71.

JAMES, E. G. Fixed-point drift and hysteresis in frequency-scaled unimanual coordination. **Journal of Motor Behavior**, Routledge, v. 44, n. 4, p. 281–288, 2012. PMID: 22857616. Disponível em: <https://doi.org/10.1080/00222895.2012.702141>. Citado na página 56.

KAGGLE. **State of Data Science and Machine Learning 2022**. 2022. Disponível em: <https://www.kaggle.com/kaggle-survey-2022>. Acesso em: 05/12/2022. Citado na página 37.

KE, G.; MENG, Q.; FINLEY, T.; WANG, T.; CHEN, W.; MA, W.; YE, Q.; LIU, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2017. v. 30. Disponível em: <https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf>. Citado na página 66.

KOLMOGOROV, A. **Sulla determinazione empirica di una legge di distribuzione». Giornale dell'istituto italiano degli attuari.** 1933. Citado na página 52.

KUHN, M.; JOHNSON, K. **Applied Predictive Modeling**. 1. ed. [S.l.]: Springer New York, NY, 2013. ISBN 978-1-4614-6849-3. Citado nas páginas 17, 44, 48, 49 e 54.

LUNDBERG, S. M.; ERION, G.; CHEN, H.; DEGRAVE, A.; PRUTKIN, J. M.; NAIR, B.; KATZ, R.; HIMMELFARB, J.; BANSAL, N.; LEE, S.-I. From local explanations to global understanding with explainable ai for trees. **Nature Machine Intelligence**, Nature Publishing Group, v. 2, n. 1, p. 2522–5839, 2020. Citado na página 73.

LUNDBERG, S. M.; LEE, S. A unified approach to interpreting model predictions. **CoRR**, abs/1705.07874, 2017. Disponível em: <http://arxiv.org/abs/1705.07874>. Citado nas páginas 18, 73 e 74.

LUNDBERG, S. M.; LEE, S.-I. **SHAP documentation**. Disponível em: <https://shap.readthedocs.io/en/latest/>. Acesso em: 05/12/2022. Citado nas páginas 17 e 73.

LUXBURG, U. von; SCHOELKOPF, B. **Statistical Learning Theory: Models, Concepts, and Results**. arXiv, 2008. Disponível em: <https://arxiv.org/abs/0810.4752>. Citado nas páginas 38 e 39.

MANTOVANI, R. G.; HORVÁTH, T.; CERRI, R.; JUNIOR, S. B.; VANSCHOREN, J.; CARVALHO, A. C. P. de Leon Ferreira de. An empirical study on hyperparameter tuning of decision trees. **CoRR**, abs/1812.02207, 2018. Disponível em: <http://arxiv.org/abs/1812.02207>. Citado nas páginas 55 e 59.

MARSLAND, S. **Machine Learning: An Algorithmic Perspective**. 2. ed. [S.l.]: Chapman and Hall/CRC, 2014. ISBN 1466583282. Citado nas páginas 38 e 39.

MENGNAN, D.; NINGHAO, L.; XIA, H. **Techniques for Interpretable Machine Learning**. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1808.00033>. Citado na página 72.

MICROSOFT. **LGBM documentation**. Disponível em: <https://lightgbm.readthedocs.io/en/v3.3.2/>. Acesso em: 05/12/2022. Citado na página 66.

MITCHELL, T. M. **Machine Learning**. 1. ed. [S.l.]: McGraw-Hill, 1997. ISBN 0070428077. Citado na página 38.

NESTEROV, Y. **Introductory Lectures on Convex Optimization**. 1. ed. [S.l.]: Springer, 2004. Citado na página 62.

OPTUNA. **Optuna documentation**. Disponível em: <https://optuna.readthedocs.io/en/stable/index.html>. Acesso em: 05/12/2022. Citado na página 71.

PROBST, P.; BISCHL, B.; BOULESTEIX, A.-L. **Tunability: Importance of Hyperparameters of Machine Learning Algorithms**. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1802.09596>. Citado nas páginas 27, 54 e 60.

RUDER, S. An overview of gradient descent optimization algorithms. **CoRR**, abs/1609.04747, 2016. Disponível em: <http://arxiv.org/abs/1609.04747>. Citado na página 62.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach, Global Edition**. 4. ed. [S.l.]: Pearson, 2021. ISBN 9780134610993; 0134610997; 9781292401133; 1292401133; 9781292401171; 1292401176. Citado nas páginas 17, 43, 58 e 62.

SCHAPIRE, R. E. A brief introduction to boosting. In: **Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. (IJCAI'99), p. 1401–1406. Citado na página 62.

SHAPLEY, L. S. **A Value for N-Person Games**. Santa Monica, CA: RAND Corporation, 1952. Citado na página 73.

SIGEL, S.; CASTELLAN, N. J. **Nonparametric Statistics for the Behavioral Sciences**. 2. ed. [S.l.]: Mcgraw-Hill, 1988. ISBN 1466583282. Citado na página 52.

SMIRNOV, N. Table for estimating the goodness of fit of empirical distributions. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 19, n. 2, p. 279 – 281, 1948. Disponível em: <https://doi.org/10.1214/aoms/1177730256>. Citado na página 52.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. **Practical Bayesian Optimization of Machine Learning Algorithms**. arXiv, 2012. Disponível em: <https://arxiv.org/abs/1206.2944>. Citado na página 57.

VOVK, V. The fundamental nature of the log loss function. **CoRR**, abs/1502.06254, 2015. Disponível em: <http://arxiv.org/abs/1502.06254>. Citado na página 51.

WANG, M.; ZHENG, K.; YANG, Y.; WANG, X. An explainable machine learning framework for intrusion detection systems. **IEEE Access**, v. 8, p. 73127–73141, 2020. Citado nas páginas 17, 73 e 74.

WATANABE, S.; HUTTER, F. **c-TPE: Generalizing Tree-structured Parzen Estimator with Inequality Constraints for Continuous and Categorical Hyperparameter Optimization**. arXiv, 2022. Disponível em: <https://arxiv.org/abs/2211.14411>. Citado na página 71.

YANG, L.; SHAMI, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. **CoRR**, abs/2007.15745, 2020. Disponível em: <https://arxiv.org/abs/2007.15745>. Citado nas páginas 54, 55 e 71.

ZHANGA, Y.; CHIA, G.; ZHANGA, Z. Decision tree for credit scoring and discovery of significant features: An empirical analysis based on chinese microfinance for farmers. 2018. Disponível em: <https://doi.org/10.2298/FIL1805513Z>. Citado na página 59.

ZHENG, M. **Spatially Explicit Hyperparameter Optimization for Neural Networks**. 1. ed. [S.l.]: Springer, 2022. Citado nas páginas 54, 55 e 71.

ZÖLLER, M.; HUBER, M. F. Survey on automated machine learning. **CoRR**, abs/1904.12054, 2019. Disponível em: <http://arxiv.org/abs/1904.12054>. Citado na página 56.

APÊNDICE

A

# CÓDIGOS IMPLEMENTADOS

O código final se encontra no repositório no GitHub.

ANEXO

# A

# PÁGINAS DE DOCUMENTAÇÃO

**<https://www.kaggle.com/kaggle-survey-2022>** Página com a pesquisa State of Data Science and Machine Learning 2022, plataforma Kaggle é utilizada para competições de Machine Learning e divulgação de datasets;

**<https://xgboost.readthedocs.io/en/stable/>** Página de Documentação do XGBoost;

**<https://catboost.ai/en/docs/>** Página de Documentação do CatBoost;

**<https://lightgbm.readthedocs.io/en/>** Página de Documentação do LightGBM;

**<https://optuna.readthedocs.io/en/stable/index.html>** Página de Documentação do Optuna;

**<https://shap.readthedocs.io/en/latest/>** Página de Documentação do SHAP;

**<https://www.kaggle.com/datasets/tigganeha4/diabetes-dataset-2019>** Conjunto de dados de Diabetes;

**<https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction>** Conjunto de dados de Insuficiência Cardíaca;

**<https://www.kaggle.com/datasets/vuppalaadithyasairam/kidney-stone-prediction-based-on-urine-a** Conjunto de dados do Cálculo Renal;

**<https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data>** Conjunto de dados do Carcinoma de Mama;

**<https://github.com/joaomh/study_boosting_optuna_USP_undergraduate_thesis>** Repositório final deste trabalho com as análises e os códigos implementados;