Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Estatística

Prevendo a popularidade de um *post* no Instagram via métodos de *Machine Learning*

Marcos Costa da Silva

Trabalho de Conclusão de Curso

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Estatística

Prevendo a popularidade de um *post* no Instagram via métodos de *Machine Learning*

Marcos Costa da Silva
Orientador: Prof. Dr. Rafael Izbicki

Trabalho de Conclusão de Curso apresentado ao Departamento de Estatística da Universidade Federal de São Carlos - DEs-UFSCar, como parte dos requisitos para obtenção do título de Bacharel em Estatística.

São Carlos
Fevereiro de 2024

Federal University of São Carlos
Exact and Technology Sciences Center
Department of Statistics

Predicting the popularity of an Instagram post using Machine Learning methods

Marcos Costa da Silva
Advisor: Prof. Dr. Rafael Izbicki

Bachelors dissertation submitted to the Department of Statistics, Federal University of São Carlos - DEs-UFSCar, in partial fulfillment of the requirements for the degree of Bachelor in Statistics.

São Carlos
February 2024

Marcos Costa da Silva

Prevendo a popularidade de um *post* no Instagram via métodos de *Machine Learning*

Este exemplar corresponde à redação final do trabalho de conclusão de curso devidamente corrigido e defendido por Marcos Costa da Silva e aprovado pela banca examinadora.

Aprovado em 24 de janeiro de 2024

Banca Examinadora:

- Rafael Izbicki (Orientador)
- Michel Helcias Montoril
- Ricardo Felipe Ferreira

Dedico este trabalho aos meus pais, que são os mentores da minha vida e a base de todas as minhas conquistas.

Agradecimentos

Agradeço primordialmente a Deus e aos meus pais, Marcos Rogério da Silva e Rose Natalina Costa, que me concederam o dom da vida e, desde então, sempre me proporcionaram muito amor, carinho e orientação. Expresso minha gratidão especial ao meu tio Aroldo por me introduzir ao fascinante mundo dos dados, e aos meus irmãos, Mateus e Erika e Raiany, que estiveram sempre ao meu lado. Lembro-me saudosamente das minhas avós, Matilde e Deolinda, que, sem dúvida, estariam muito orgulhosas ao verem as conquistas de seu neto. Estendo minha enorme gratidão aos demais familiares que continuamente apoiaram toda a minha jornada.

Agradeço a todos os meus amigos de infância, em especial a João Maurício, Matheus e Wesley, por todas as trocas de ideias e compartilhamento de sonhos, assim como a todos os amigos que conheci através da universidade. Quero expressar minha gratidão a cada um dos professores que me guiaram ao longo dessa jornada de graduação, transmitindo o que há de mais valioso nessa etapa da vida: o conhecimento.

Agradeço imensamente ao meu orientador, Rafael Izbicki, referência mundial em Machine Learning e a maior inspiração da minha carreira, por ter aceitado compartilhar sua experiência na orientação do meu trabalho de conclusão de curso. Não posso deixar de agradecer aos meus líderes de trabalho, Afonso e Rafael, que têm me apoiado muito nesta etapa final do curso, bem como a toda a equipe de trabalho que diariamente transmite ensinamentos valiosos para a vida.

"A gente procura fazer o melhor que a gente pode no tempo que a gente tem."

(Chorão)

Resumo

Este trabalho foi motivado pelo cenário tecnológico atual em que vivemos, no qual as pessoas estão cada vez mais conectadas nas redes sociais, gerando diariamente uma vasta quantidade de dados. Se utilizados de maneira adequada, esses dados podem proporcionar *insights* valiosos no mercado digital. Ao combinar essa fonte de informação com métodos modernos de inteligência artificial, mais especificamente, algoritmos de *machine learning*, conseguimos estudar as principais características que impulsionam a popularidade de um *post* no Instagram.

Em nossa pesquisa, utilizamos uma base de dados real com informações de perfil de 1887 usuários do Instagram. Inicialmente, exploramos os dados por meio de métodos estatísticos para compreender sua disposição e composição. Em seguida, selecionamos as variáveis de interesse e de maior relevância no para predizer a popularidade de uma publicação, utilizando técnicas como *features importance* por meio do algoritmo LightGBM na base de treinamento. Posteriormente, normalizamos as variáveis contínuas selecionadas e modelamos os dados usando tanto o algoritmo LightGBM quanto a rede neural do tipo *Perceptron* Multicamadas.

Como resultado final, concluímos que os indicadores desenvolvidos durante o processo de *feature engineering*, juntamente com as variáveis da base original, como o número de visualizações, comentários, seguidores, e as variáveis geradas no Processamento de Linguagem Natural (PLN), como o sentimento da legenda, mostraram-se as mais relevantes para prever a popularidade de um *post* no Instagram. Entre todos os testes de ajuste realizados, o modelo que demonstrou o melhor desempenho na manipulação das informações foi o LightGBM, com parâmetros otimizados, alcançando um **RMSE** de 0.04 e um $\mathbf{R}^2$ de 0.99.

Palavras-chave: *Instagram, LightGBM, Perceptron Multicamadas, Popularidade.*

Abstract

This work was motivated by the current technological scenario in which we live, where people are increasingly connected on social networks, generating a vast amount of data daily. If used appropriately, these data can provide valuable insights in the digital market. By combining this source of information with modern methods of artificial intelligence, more specifically, machine learning algorithms, we were able to study the main characteristics that drive the popularity of a post on Instagram.

In our research, we used a real database with profile information from 1887 Instagram users. Initially, we explored the data through statistical methods to understand its disposition and composition. Then, we selected the variables of interest and greater relevance to predict the popularity of a publication, using techniques such as feature importance through the LightGBM algorithm on the training set. Subsequently, we normalized the selected continuous variables and modeled the data using both the LightGBM algorithm and the Multilayer Perceptron neural network.

As a final result, we concluded that the indicators developed during the feature engineering process, along with the variables from the original database such as the number of views, comments, followers, and the variables generated in Natural Language Processing (NLP) such as the sentiment of the caption, proved to be the most relevant for predicting the popularity of a post on Instagram. Among all the fitting tests performed, the model that demonstrated the best performance in handling the information was LightGBM, with optimized parameters, achieving an **RMSE** of 0.04 and an $\mathbf{R^2}$ of 0.99.

Keywords: *Instagram, LightGBM, Multi Layer Perceptron, Popularity.*

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

O Instagram, uma das principais redes sociais da atualidade, passou por diversas transformações desde seu surgimento em 2010. Inicialmente, concebido como uma plataforma para compartilhamento de imagens com filtros e legendas, o aplicativo conquistou rapidamente milhões de usuários, que utilizavam a plataforma para registrar momentos de viagens, rotina diária, festas e momentos em família (Canaltech, 2023).

Com o passar do tempo, o Instagram foi se aprimorando e introduziu novas funcionalidades para diversificar o conteúdo compartilhado. Em 2016, surgiram os *Stories*, que são publicações curtas com tempo limitado de exibição, e em 2018, os *Reels*, vídeos curtos com diversos efeitos, para competir com a ascensão do TikTok (Nerdweb, 2022).

Segundo dados do VOLPATO (2023), o Instagram conta com aproximadamente 2 bilhões de usuários ativos em todo o mundo, sendo 113 milhões apenas no Brasil. Essa ampla base de usuários tornou a plataforma um canal estratégico para influenciadores digitais e agências de *marketing*, que o utilizam como meio de divulgação de conteúdos estratégicos para atrair a atenção dos usuários e incentivar a compra de produtos. O alcance proporcionado pela rede social amplia a possibilidade de conquistar novos clientes e aumentar a receita das empresas, que antes não seriam atingidos organicamente.

No contexto de publicidade, o uso do Instagram como ferramenta de divulgação é realizado tanto através de estratégias de *marketing* de influência quanto por meio de anúncios direcionados para melhorar o tráfego dos sites das empresas, o *marketing* digital. Enquanto o primeiro se baseia na credibilidade de pessoas para promover produtos por meio de postagens, o segundo método utiliza anúncios segmentados para direcionar o tráfego para as empresas (ZANE, 2022).

O investimento no *marketing* de influência vem crescendo significativamente, alcançando

aproximadamente US13, 8 bilhões em 2021. No Brasil, em 2022, cerca de 67% dos usuários do Instagram acompanhavam algum *influencer* e 40% das compras realizadas na *internet* eram influenciadas por eles. Além disso, as empresas que investem corretamente na contratação de influenciadores podem obter um retorno sobre o investimento de até cinco vezes o valor investido (Exame, 2022).

A popularidade de uma postagem influencia diretamente o sucesso de uma propaganda e a conversão de vendas gerada por um influenciador. Por isso, compreender as principais características que tornam uma publicação relevante no Instagram é fundamental para aprimorar o conteúdo divulgado e direcioná-lo ao público adequado, aumentando o engajamento da postagem e a credibilidade da conta divulgadora (Carta *et al.*, 2020).

Nesse sentido, a análise de dados e a identificação das características que tornam uma postagem popular no Instagram podem contribuir significativamente para o aprimoramento dos conteúdos a serem divulgados e para uma possível segmentação mais efetiva do público-alvo. A aplicação de técnicas estatísticas, como análise exploratória de dados e testes específicos, combinada com a utilização de métodos de aprendizado de máquina, como o LightGBM e redes neurais, pode proporcionar *insights* valiosos sobre a popularidade de uma postagem e direcionar estratégias de *marketing* de forma mais eficiente. Assim, essa pesquisa tem por objetivo geral investigar as principais características que tornam uma postagem popular, utilizando técnicas estatísticas e prever a popularidade dos *posts* através de *machine learning*.

Nos próximos capítulos, apresentamos a metodologia, na qual descrevemos as principais técnicas desenvolvidas no trabalho, desde a obtenção dos dados até a aplicação dos métodos de análise descritiva, pré-processamento e modelagem. Especificamos detalhadamente cada etapa para a aplicação de um problema de aprendizado de máquina supervisionado com foco em predição, começando pela divisão dos dados em conjuntos de treino e teste, até a avaliação de desempenho dos modelos selecionados para o estudo, que incluem o LightGBM e o Perceptron Multicamadas. Após a conclusão dessas etapas, apresentamos o capítulo referente à aplicação das técnicas mencionadas no banco de dados selecionado. Por fim, apresentamos as conclusões deste trabalho.

Capítulo 2

Metodologia

A metodologia adotada abrange diversas etapas, desde a análise descritiva e o pré-processamento dos dados até a implementação de modelos preditivos. Nas duas primeiras fases, buscamos entender melhor as informações disponíveis, tratar e criar variáveis que possam contribuir no ajuste dos modelos. Para características textuais, como a descrição e legenda das postagens, são empregadas técnicas de Processamento de Linguagem Natural (PLN), que visam extrair *insights* semânticos a partir dos textos. Já para a criação de novas *features*, utilizamos o método de engenharia de recursos, o qual busca capturar nuances de relacionamento entre as variáveis, podendo ser representado por meio de um novo indicador.

Na sequência, será apresentada uma análise detalhada dos métodos utilizados neste estudo, o *LightGBM*, um algoritmo de *boosting* que vem sendo amplamente utilizado tanto em competições de *Machine Learning* como em empresas *Data Driven*. Além disso, serão discutidas as redes neurais, mais especificamente a arquitetura do *Perceptron* Multicamadas. Também serão destacadas diversas abordagens para aprimorar a capacidade preditiva do modelo, que vão desde a seleção de variáveis até o *tuning* de parâmetros.

Este estudo busca explicar a popularidade de uma postagem no Instagram, bem como contribuir para a compreensão mais profunda dos fatores que impulsionam a interação dos usuários nessa plataforma. Ao integrar abordagens de *Machine Learning* com dados específicos do Instagram, este trabalho visa oferecer uma perspectiva valiosa para usuários, profissionais de *marketing* digital e influenciadores, sobre como compreender e otimizar a presença *online*.

2.1 Elementos para o caso de predição

Quando nos referimos às técnicas de aprendizado supervisionado, estamos essencialmente buscando empregar essas técnicas para fazer previsões o mais precisas possível sobre novas observações que compartilham características semelhantes àquelas presentes na base de dados usada para treinar o modelo, mas que não possuem rótulos definidos antecipadamente.

Tabela 2.1: Formato usual da base de dados para predição.

Covariáveis			Rótulo
$X_{1,1}$	$\dots$	$X_{1,d}$	Y_1
$\vdots$	$\ddots$	$\vdots$	$\vdots$
$X_{n,1}$	$\dots$	$X_{n,d}$	Y_n

Em notação de vetor, temos: $(\mathbf{X}_i, \mathrm{Y}_i) \in \mathbb{R}^d \times \mathbb{R}$, $i = 1, \dots n$.

Matematicamente, no estudo de predição, buscamos encontrar uma relação entre uma observação $\mathbf{x} \in \mathbb{R}^d$ das covariáveis $(\mathbf{X}_1, ..., \mathbf{X}_d)$ e a variável resposta $\mathbf{Y}$ por meio de uma função do tipo:

$$r : \mathbb{R}^d \to \mathbb{R}$$
$$\mathbf{x} \to r(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$$

Para elucidar ainda mais este tipo de aprendizado, podemos subdividir esta classe de modelos com base no tipo de tarefa a ser realizada e no formato da variável resposta. Em essência, quando a variável resposta Y é quantitativa, nos deparamos com um problema de regressão, enquanto que se Y for qualitativa, enfrentamos um problema de classificação. Com isso em mente, cada situação apresenta características específicas que serão exploradas em detalhes mais adiante. No entanto, antes de nos aprofundarmos nesses aspectos, é crucial compreendermos o objetivo do método preditivo. (Izbicki e dos Santos, 2020).

2.1.1 Objetivo do aprendizado supervisionado

Como já mencionado anteriormente, o foco desse tipo de aprendizado está direcionado a realizar boas predições para novas amostras de dados que possuam as mesmas características daquela utilizada na fase de ajuste, o que só é possível quando se tem uma

função g que captura a relação existente entre as covariáveis e o rótulo. Dessa maneira, supõe-se que os novos *inputs* de dados, $(\mathbf{X}_{n+1},Y_{n+1})$, ..., $(\mathbf{X}_{n+m},Y_{n+m})$ sejam i.i.d. de forma que:

$$g(\mathbf{x}_{n+1}) \approx y_{n+1}, \ldots, g(\mathbf{x}_{n+m}) \approx y_{n+m},$$

em que m é um número inteiro positivo.

Diferentemente da abordagem inferencial, na qual são feitas diversas suposições sobre o modelo e de que o mesmo é correto, na abordagem preditiva a única suposição que temos é de que as novas observações sejam i.i.d., já que não precisamos necessariamente de um modelo que atenda aos pressupostos de normalidade, homocedasticidade e independência dos erros, mas sim de um ajuste que apresente as melhores estimativas. A seguir, detalharemos melhor cada uma das tarefas desse tipo de aprendizado (Izbicki e dos Santos, 2020).

2.1.2 Regressão

Cada modelo destinado à tarefa de regressão possui em sua estrutura uma função de predição do tipo: $g : \mathbb{R}^d \longrightarrow \mathbb{R}$. Uma maneira de avaliar a eficácia dessa função é por meio de uma medida de risco, que visa quantificar o desvio entre a predição e o valor observado. Essa métrica pode levar em consideração o desvio ao quadrado (utilizaremos como referência), em módulo, entre outras abordagens (Izbicki e dos Santos, 2020):

$$R(g) = \mathbb{E}\big[(Y - g(\mathbf{X}))^2\big].$$

2.1.3 Classificação

No caso de classificação, temos que Y pode assumir valores em um conjunto de classes $\mathcal{C}$, ou seja, $g : \mathbb{R}^d \longrightarrow \mathcal{C}$. Para medir sua eficiência, podemos tomar como base a função de risco (Izbicki e dos Santos, 2020):

$$R(g) = \mathbb{E}[\mathbb{I}(Y \neq g(\mathbf{X}))] = \mathbb{P}(Y \neq g(\mathbf{X})).$$

2.1.4 Particionamento dos dados

Selecionado o tipo de problema a se tratar, o passo seguinte é determinar como será feito o *input* de dados para o aprendizado do modelo. Como primeira opção, podemos utilizar a técnica de *Data Splitting*, em que separamos de forma aleatória uma parte dos dados para a etapa de treinamento e deixamos uma outra parte para o teste, onde avaliamos o ajuste pela função de erro. Considerando $s < n$, geralmente é adotado o seguinte critério:

Tabela 2.2: Dados Treino (70%).

$\mathbf{X}$	$\mathbf{Y}$
$\mathbf{X}_1$	Y_1
$\vdots$	$\vdots$
$\vdots$	$\vdots$
$\mathbf{X}_s$	Y_s

Tabela 2.3: Dados Teste (30%).

$\mathbf{X}$	$\mathbf{Y}$
$\mathbf{X}_{s+1}$	Y_{s+1}
$\vdots$	$\vdots$
$\mathbf{X}_n$	Y_n

Outra abordagem seria aplicar a técnica de validação cruzada, na qual se utiliza toda a amostra. Inicialmente, os dados são particionados aleatoriamente em k lotes de tamanhos semelhantes. Um desses lotes é retirado para teste, enquanto os demais são utilizados no treinamento. Esse processo é repetido de maneira que cada um dos k lotes seja testado uma vez. O resultado final é uma média da função de erro de todos os lotes de testes (Izbicki e dos Santos, 2020).

Entendido os elementos que compõem um problema de predição, seguimos para etapa de modelagem. Neste trabalho, optamos por aplicar especificamente dois algoritmos de aprendizado de máquina supervisionado: o LightGBM e o *Perceptron* Multicamadas, ambos voltados para a tarefa de regressão. O que caracteriza a escolha dessa classe de modelos é principalmente o formato tabular dos dados, no qual temos uma base composta por d variáveis preditoras e uma variável resposta, ou seja, cada possível combinação dessas variáveis explicativas recebem um rótulo pré-determinado.

Posteriormente, realizamos a divisão do conjunto de dados em treino e teste, empregando validação cruzada durante a etapa de treinamento. Esse procedimento visa avaliar o desempenho do modelo em diferentes subconjuntos de dados, assegurando replicabilidade para novas observações desconhecidas durante o teste dos modelos estudados. Na fase de avaliação, examinamos a eficácia do algoritmo em aprender sobre os dados. Para isso, utilizamos as principais métricas de regressão (MAE, MSE, RMSE e R^2) a fim de

verificar a capacidade do ajuste em generalizar para novas observações. Além disso, aplicamos métodos de otimização de hiperparâmetros (*Random Search)*, visando realizar uma comparação entre os dois algoritmos mencionados e selecionar como representante final aquele que apresentar melhor desempenho nesta tarefa.

2.2 Processamento de linguagem natural

O processamento de linguagem natural tem como objetivo processar dados no formato textual, de forma a gerar resultados sobre os conteúdos analisados. Essa aplicação é bastante usual ao lidar com dados de redes sociais como, por exemplo, na análise de *tweets*.

Os métodos de PLN envolvem uma variedade de técnicas, como *"tokenização"* (divisão do texto em unidades significativas), análise morfológica (estudo da estrutura das palavras), análise sintática (análise da estrutura gramatical das frases) e processamento semântico (compreensão do significado). Além disso, técnicas de aprendizado de máquina, como redes neurais, são frequentemente aplicadas para melhorar a capacidade de um sistema em lidar com a complexidade da linguagem natural.

No contexto deste trabalho, são adotadas técnicas específicas de PLN para representação de documentos e análise de sentimentos. Essas técnicas são aplicadas em informações como legenda e conteúdo da imagem referente ao *post*, e as variáveis resultantes são incluídas no modelo com intuito de melhorar a precisão do ajuste.

2.2.1 Representação de documentos por meio de *Bag of Words*

A técnica *Bag of Words* (BoW) é uma abordagem fundamental em processamento de linguagem natural (PLN) e recuperação de informações. Sua essência reside na representação simplificada de documentos de texto, onde a ordem das palavras é ignorada, e apenas a frequência de ocorrência das palavras é considerada.

No *Bag of Words*, um documento é tratado como um "saco"ou coleção não ordenada de palavras, e a informação sobre a estrutura gramatical é descartada. Cada palavra única no vocabulário do *corpus* é representada como uma dimensão, e a presença ou ausência de cada palavra em um documento é codificada. Esse método resulta em uma matriz de termos, onde cada linha corresponde a um documento referente à publicação e cada coluna a uma palavra única que não se repete.

A importância do *Bag of Words* reside na simplicidade da representação e na sua

aplicabilidade em diversas tarefas de PLN, como classificação de texto, *clustering* e análise de sentimentos. Essa abordagem é particularmente eficaz em situações onde a estrutura gramatical é menos relevante, mas a frequência das palavras é informativa, como em análises de grandes conjuntos de documentos.

Em resumo, o *Bag of Words* é uma técnica valiosa para simplificar a representação de documentos de texto, proporcionando uma base sólida para análises semânticas e extração de informações em diversas aplicações de processamento de linguagem natural.

Para realização da etapa de codificação das palavras, utilizaremos a função *CountVectorizer* da biblioteca *scikit-learn* do Python, que converte uma coleção de documentos de texto em uma matriz de contagem de *tokens*. Ela transforma o texto em uma representação numérica, onde cada linha representa um documento e cada coluna representa a contagem de uma palavra específica nesse documento (Izbicki e dos Santos, 2020).

Isso será essencial para destacarmos os termos que em geral mais aparecem nos textos analisados e consequentemente para construção de um indicador de sentimentos, que será melhor descrito a seguir.

2.2.2 Análise de sentimentos

A análise de sentimentos é uma técnica de Processamento de Linguagem Natural (PLN) voltada para a compreensão das emoções expressas em textos. O objetivo é avaliar e identificar a polaridade emocional presente em determinados conjuntos de dados textuais.

O uso desta técnica envolve a aplicação de algoritmos e modelos de aprendizado de máquina para classificar o tom emocional de um texto como positivo, negativo ou neutro. Nesse contexto, será adotada uma abordagem de classificadores baseados em aprendizado supervisionado, em especial utilizaremos a biblioteca *TextBlob* do Python.

Essa biblioteca utiliza o classificador de Naive Bayes, um algoritmo de previsão que se baseia no Teorema de Bayes. Ele assume independência condicional entre os recursos, que, no caso do processamento de texto, são as palavras. Essa suposição simplifica os cálculos e torna o algoritmo eficiente e fácil de implementar, especialmente para tarefas de análise de sentimentos.

Durante a fase de treinamento no *TextBlob*, o modelo é exposto a exemplos de textos com rótulos de sentimentos correspondentes. Isso permite que o modelo aprenda a reconhecer padrões linguísticos associados a diferentes sentimentos. Depois de treinado, o ajuste pode ser aplicado a novos textos para prever o sentimento associado a cada um

deles. Dessa forma, como a biblioteca *TextBlob* já contém uma função com o algoritmo treinado, é suficiente aplicá-la aos dados do estudo e criar a variável de sentimento, que contribuirá para complementar o *book* de variáveis a serem testadas nos modelos. Neste estudo, os textos a serem analisados referem-se às legendas das postagens e às descrições dos conteúdos das imagens (Silva, 2021).

2.3 Engenharia de atributos

A engenharia de atributos, também popularmente conhecida como *feature engineering*, refere-se ao processo de tratar e criar novas variáveis a partir das já existentes em um conjunto de dados. O objetivo é aprimorar o desempenho e a eficácia de modelos de *machine learning*. Essas novas *features* são desenvolvidas para capturar padrões importantes e informações relevantes contidas nos dados originais.

Na modelagem de dados, a qualidade das *features* desempenha um papel crucial no sucesso de um modelo. Nesse contexto, toda criatividade é válida ao aplicar o processo de engenharia de atributos, especialmente quando lidamos com variáveis que não contribuem significativamente em uma aplicação. Qualquer transformação aplicada nas variáveis existentes, com o intuito de torná-las mais compreensíveis e informativas é importante para aprimorar a capacidade dos algoritmos de aprendizado de máquina. Essa técnica abrange desde a criação de novas *features* até a combinação de variáveis existentes, a normalização de dados, a manipulação de formatos, entre outras estratégias.

Uma aplicação bastante clara para exemplificar a importância desse método seria a criação de um novo atributo com base em duas variáveis altamente correlacionadas. Dessa forma, evitamos problemas de multicolinearidade, permitindo que o novo atributo represente de maneira mais eficaz as informações combinadas das duas variáveis.

Assim, o processo de *feature engineering* contribui para aprimorar a capacidade do modelo em identificar padrões, realizar previsões mais precisas e generalizar bem para dados não observados. Em resumo, trata-se de um procedimento crucial no desenvolvimento de modelos de *machine learning* eficazes, permitindo extrair o máximo de informação dos dados disponíveis (Corrêa, 2021).

2.4 Otimização de parâmetros

A otimização de parâmetros é uma abordagem que envolve testar diferentes valores para os hiperparâmetros de um modelo e escolher um subconjunto que resulte no melhor desempenho em um conjunto de dados específico.

Neste estudo, utilizamos o método de ajuste de parâmetros, conhecido como *Random Search*. Trata-se de uma abordagem de otimização amplamente empregada em algoritmos de aprendizado de máquina para encontrar a combinação mais eficiente de hiperparâmetros de um modelo. Hiperparâmetros são configurações externas ao algoritmo que não são estimadas durante o treinamento, e encontrar valores ideais para esses parâmetros é crucial para otimizar o desempenho do modelo.

Ao contrário da busca de grade (*grid search*), que avalia todas as combinações possíveis de valores para encontrar a melhor combinação de parâmetros em uma grade pré-definida, o *Random Search* seleciona aleatoriamente conjuntos específicos no espaço de possibilidade para avaliação. Essa abordagem aleatória tem a vantagem de explorar uma ampla gama de valores de forma mais eficiente, especialmente quando o espaço de busca é extenso.

O método de *Random Search* foi escolhido devido à sua eficácia em situações em que o número de hiperparâmetros é grande e a busca exaustiva seria computacionalmente custosa. A abordagem aleatória possibilita uma exploração abrangente desses espaços, frequentemente encontrando soluções satisfatórias em um período de tempo mais curto. Embora não garanta a descoberta da melhor combinação possível, é uma estratégia eficiente para encontrar soluções aceitáveis em cenários práticos de aplicação (Brownlee, 2020).

2.5 LightGBM

O *Light Gradient Boosting Machine* (LightGBM), desenvolvido pela Microsoft em 2016 com o propósito de otimizar o uso de memória e tempo de processamento, é um algoritmo de aprendizado de máquina do tipo *ensemble*, ou seja, é um composto de vários modelos que, juntos, desempenham uma performance melhor.

Ensembles, em geral, possuem diversas características, podendo ser do tipo *Bagging*, no qual cada algoritmo é ajustado em paralelo para realizar previsões, e o resultado final é determinado de acordo com a previsão mais frequente. Um exemplo são as *Random Forests*. No formato de *Boosting*, os modelos são usados em sequência, tendo como ob-

jetivo o ajuste atual prever o resíduo do anterior, sem muitas restrições na escolha da função de perda. Além desses métodos, ainda temos o *Gradient Boosting*, semelhante ao segundo caso, porém utiliza uma métrica com erro diferenciável para o uso do gradiente descendente, onde buscamos um valor de previsão de forma que a função de perda seja mínima, contribuindo precisamente no processo de otimização do resíduo.

No caso do LightGBM, temos em sua estrutura um *ensemble* do tipo *Gradiente Boosting Decision Tree* (GBDT), ou seja, os ajustes simples são implementados com árvores de decisão que utilizam gradiente para correção dos resíduos, muito utilizado devido à sua alta eficiência, precisão e interpretabilidade. Uma de suas vantagens computacionais está na forma como essas árvores são construídas, já que seu crescimento não ocorre por linha em mais de uma folha. Em vez disso, a cada nível, é escolhida uma única folha a ser expandida, geralmente aquela que mais tende a reduzir a perda (Microsoft, 2023).

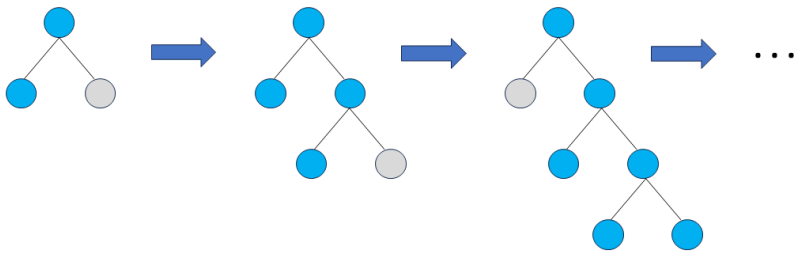

Figura 2.1: Representação: *Life-wise tree growth.* Fonte: Autor.

Outra característica desse modelo é que não utiliza diretamente as árvores de decisão, cujo objetivo é encontrar o melhor corte dentre todos os possíveis, mas sim utiliza de um tipo de aprendizado que tem como base o método de histograma altamente otimizado, garantindo maior eficiência com relação ao tempo e consumo de memória, já que reduz-se todos os possíveis pontos de corte em *bins* que agrupam pontos próximos.

De modo geral, a proposta do LightGBM é solucionar o problema de tempo e custo com memória durante seu treinamento. Para isso, as principais técnicas adotadas visam encontrar os melhores pontos de divisão nas árvores que compõem a estrutura do método, sem ter que testar todos os possíveis casos (histograma altamente otimizado).

Além disso, utilizam do *exclusive feature bundling* (EFB), ou seja, agrupamento de recursos exclusivos , que permite reduzir a dimensão das variáveis através da combinação de recursos apropriados, como é o caso das variáveis *dummies*, que assumem 0 ou 1 indicando a ausência ou presença de determinada classe em uma característica.

Por fim, para facilitar a convergência do modelo em um resultado preciso sem a necessidade de explorar todas as entradas do banco de dados de uma única vez, no qual é feito um ajuste inicial e a partir dos gradientes (erros residuais) de cada observação prevista, uma amostragem é realizada, selecionando deterministicamente os casos em que se obtiveram os maiores erros, em torno de 20%, e dos 80% com menores gradientes, seleciona-se de forma aleatória uma porcentagem pré-definida, em torno de 10%, para finalizar a composição dessa amostra.

A seguir, descrevemos as características gerais do método de *Gradient Boosting Machine*, do GBDT, um caso específico dessa classe de algoritmos, em que utiliza árvores de decisão em sua estrutura e nessa mesma seção, mencionaremos sobre o histograma altamente otimizado que é utilizado no LightGBM, assim como entraremos no detalhe sobre a amostragem e empacotamento de variáveis do modelo mencionado, **GOSS** e **EFB**, respectivamente, finalizando com uma ideia geral para implementação desse tipo de algoritmo.

2.5.1 Gradient Boosting Machine

A estrutura estatística do Gradient Boosting Machine (GBM), foi desenvolvida por Jerome H. Friedman (Friedman, 2001), com intuito de solucionar uma otimização numérica, cujo objetivo era minimizar a perda do modelo a medida em que seria incluído ajustes simples (fracos), assim como ocorre no processo de gradiente descendente, sem alterar os termos já inclusos anteriormente. Segundo (Saini, 2021) para realização desse método é necessário a execução de três etapas: otimizar uma função de perda, um modelo inicial fraco e um modelo aditivo para adicionar mais ajustes simples, a fim de minimizar o conteúdo do primeiro passo.

- Função de Perda: sua principal característica é ser diferenciável, já que o gradiente *boosting* garante que um novo algoritmo de *boosting* não precise ser derivado para cada possível função escolhida, devido sua estrutura genérica. Comumente, a função

de perda utilizada é dada por:

$$L(g(\mathbf{X}), Y) = (Y - g(\mathbf{X}))^2,$$

já que garante a propriedade de ser diferenciável, como exigida nessa estrutura, para aplicação do Gradiente descendente, cujo objetivo é minimizar a função de perda.

- Modelo Fraco: em geral, é composto por árvores de decisão exaustivas, ou seja, que escolhem os melhores pontos de divisão com base em medidas de pureza, como Gini, ou que minimizam a perda.

- Modelo Aditivo: consiste em adicionar uma árvore por vez, não alterando os casos já pertencentes ao modelo e, como forma de minimizar a perda nesse processo, é utilizado o gradiente descendente. No geral, esse método é usado para reduzir um conjunto de parâmetros, tal como os coeficientes de uma regressão ou pesos em uma rede neural e com base no erro ou perda de cada iteração, os pesos são atualizados, diminuindo o resíduo final. Seguindo a mesma lógica, para o caso do modelo aditivo, calculamos a perda dos submodelos que compõem a estrutura geral e utilizamos o gradiente descendente, adicionando uma nova árvore que reduza ainda mais a perda (segue o gradiente). O processo se repete até que não haja melhoramento na perda, finalizando o treinamento.

2.5.2 GBDT

Gradient Boosting Decision Tree (GBDT) é um tipo de *ensemble* que constitui em sua estrutura árvores de decisão que são treinadas sequencialmente e corrigidas a cada iteração através de gradientes negativos (erros residuais).

Diferentemente dos demais algoritmos de *boosting*, o LightGBM, ao invés de usar a técnica de pré-classificação, na qual busca o melhor ponto de divisão dentre todos possíveis, em baixa velocidade e alto custo de memória, é utilizado o algoritmo baseado em histograma altamente otimizado, no qual agrupa os valores das variáveis contínuas em intervalos discretos, formando as barras do histograma de recursos, garantindo maior eficiência no treinamento, já que reduz consideravelmente as possibilidades de corte.

Todavia, apesar dessa técnica melhorar a complexidade do modelo, ainda temos um problema quanto à aplicação em grandes bancos de dados, já que a quantidade de pontos

de divisão dependerá do número de instâncias e recursos.

Para aprender uma função com base no conjunto de dados de entrada, com dimensão $\mathcal{X}^s$, ao espaço gradiente, $\mathcal{G}$, como composição de sua estrutura, o GBDT usa árvores de decisão. Então, tendo um conjunto de treinamento de tamanho n com as observações i.i.d, a cada iteração do gradiente, aqueles que assumem valores negativos na função de perda em relação à saída do modelo são denotados como $\{g_1, ..., g_n\}$. Dessa forma, a árvore de decisão divide cada nó onde se apresenta o maior ganho de informação, que no GBDT, geralmente é medido pela variância após a divisão (Ke *et al.*, 2017).

2.5.3 GOSS

Os tipos de amostragem variam de acordo com o *ensemble* e tendem a ser comuns, por simplesmente se basearem na seleção de subconjuntos das instâncias de dados. Algumas dessas metodologias tem como base o AdaBoost, o qual usa técnicas em que as observações são filtradas de acordo com seu peso comparado a um limite pré-fixado, ou ainda, usam métodos em que a taxa de amostragem é atualizada a cada passo do treinamento. Ademais, temos um tipo específico utilizado no *Stochastic Gradient Boosting* (SGB), onde em cada iteração é usada uma amostra aleatória para treinar os modelos simples.

Embora essas técnicas sejam amplamente utilizadas, aquelas baseadas no AdaBoost não podem ser aplicadas diretamente no GBDT, pois não possuem pesos nativos para as observações em sua estrutura, e o SGB tende a comprometer a precisão do ajuste.

A fim de utilizar um método eficiente de amostragem no LightGBM, foi sugerido o uso do *Gradient-based One-Side Sampling* (GOSS) em sua estrutura, uma forma de reduzir a magnitude dos dados, sem que haja perda de informações relevantes. Apesar do GBDT não possuir pesos nativos, o gradiente aplicado a cada observação gera uma pontuação. Isso permite que o GOSS mantenha todas as entradas com gradientes altos e faça uma amostragem nas demais, composta por pequenos gradientes e que são multiplicados por uma constante a fim de manter a característica da distribuição dos dados.

De forma geral, no GOSS, primeiro é realizada a ordenação decrescente das instâncias utilizadas no treino de acordo com seus valores resultantes do gradiente, em módulo. Em seguida, de acordo com um valor pré-estabelecido, a, é mantido na amostra, seguindo a ordem estabelecida anteriormente, as $a \times 100\%$ observações com maiores gradientes, formando o subconjunto A. Assim, para o restante dos dados, com menores gradientes,

dado por A^c e representando $(1-a)\times 100\%$ do conjunto de dados, retiramos uma amostra aleatória, formando o subconjunto B, com tamanho $b\times|A^c|$. Por fim, realizamos a divisão entre as observações de acordo com o ganho de informação (variância) estimada, $\tilde{V}_j(d)$, aplicada ao conjunto $A\cup B$.

Pelo fato de utilizarmos o ganho de informação em um subconjunto menor de instâncias, ao invés do conjunto todo, para determinar o melhor ponto de divisão, é que reduzimos o custo computacional e temos uma garantia de que a precisão não será fortemente impactada quando o número de dados é grande. Vale destacar que, quando $a = 0$, temos que a amostragem é a aleatória simples. (Ke *et al.*, 2017).

2.5.4 EFB

Quando se trata da redução de recursos, é comum que seja feita uma filtragem das características não relevantes e, caso ainda permaneça alta dimensão (esparsidade) na base de dados, sob hipótese de que os atributos possuam redundância significativa, métodos como análise de componentes principais (PCA) são aplicados, porém nem sempre essa suposição é valida. Em geral, projetamos trabalhar com um conjunto de dados cujas variáveis contribuam de forma exclusiva e não considerar alguma delas pode impactar na precisão do modelo.

De forma a não comprometer a eficiência do ajuste, o método de agrupamento de recursos exclusivos (EFB), foi proposto para garantir a alta eficácia do LighGBM. Dessa maneira, sabendo que em um espaço de atributos esparsos, muitas dessas informações são mutuamente exclusivas, isto é, nunca assumem valores diferentes de zero de forma simultânea, podemos agrupar tais recursos em um único, chamado de pacote de recursos exclusivos.

Para tal aplicação, podemos utilizar um algoritmo de varredura de variáveis, o qual terá como função, construir pacotes de atributos como são feitos nas *features* individuais, reduzindo a complexidade computacional de $O(n\times d)$ para $O(n\times p)$, desde que $p < d$, sendo n o tamanho da base de dados, d quantidade de variáveis originais e p a quantidade de variáveis após empacotamento. Para isso precisamos determinar quais recursos devem ser agrupados e como construir o empacotamento (Ke *et al.*, 2017).

2.5.5 Implementação Básica - GBDT

Para implementação específica do LightGBM, basta seguir o passo a passo genérico a seguir, especificando o método de amostragem dos dados como sendo o GOSS e a técnica de seleção de recursos EFB.

Formulação genérica para Regressão

1. Inicialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$, sendo $\gamma = -\frac{\partial L(y,\bar{y})}{\partial \bar{y}}$, em que y é o valor verdadeiro e $\bar{y}$ é a média dos valores verdadeiros. Em muitas implementações, $f_0(x)$ é, na verdade, a média dos valores verdadeiros. Isso é conhecido como a solução de mínimos quadrados para a inicialização do modelo em problemas de regressão.

2. Para $m = 1, ...M$:

   (a) Para $i = 1, ..., N$, calcule:

   $$r_{im} = -\Big[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\Big]_{f=f_{m-1}}.$$

   (b) Ajuste uma árvore de regressão nos valores de r_{im} para obter as regiões $R_{jm}, j = 1, ...J_m$.

   (c) Para $j = 1, ..., J_m$, calcule:

   $$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1(x_i)} + \gamma).$$

   (d) Atualizar $f_m(x)$:

   $$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(x \in R_{jm}).$$

3. Estimativa: $\hat{f}(x) = f_M(x)$.

Formulação genérica para Classificação

Para este caso, a cada iteração m, repetimos K vezes o passo 2, ou seja, uma vez para cada classe usamos:

$$-g_{ikm} = \Big[\frac{\partial L(y_i, f_1(x_i), ..., f_K(x_i)}{\partial f_k(xi)}\Big]_{f(x_i)=f_{m-1(x_i)}}$$
$$= \mathbb{I}(y_i = \mathcal{G}_k) - p_k(x_i).$$

A estimativa final será dada pela classe mais votada ou através das probabilidades resultantes, dentre as respostas de K expansões de diferentes árvores (acopladas) $f_{kM}(x), k = 1, ..., K$ (Hastie *et al.*, 2009).

2.6 Redes Neurais Artificiais - PMC

As redes neurais artificias representam uma classe de algoritmos que possuem em sua estrutura um modelo matemático inspirado no funcionamento do cérebro humano. Caracterizada como um tipo de aprendizado profundo, ou seja, capaz de extrair características complexas e abstratas dos dados, baseia-se no uso de neurônios artificiais (nós) que se conectam através de camadas por meio de pesos, criando um sistema de atualização interna que aprende com os próprios erros, ou seja, adquirem conhecimento através de experiência. Devido a esse fato, essa classe de modelos por padrão necessita de uma quantidade relevante de dados (sinais) para o treinamento. A arquitetura de uma rede neural artificial é composta por neurônios interconectados em três tipos de camadas:

- Camada de entrada: *input* dos dados a serem processados;
- Camadas ocultas: analisa e processa o resultado da camada que a antecede; no caso mais básico, são os dados da camada de entrada, em casos mais complexos, são os resultados da camada oculta anterior;
- Camada de saída: fornece o resultado final de todo o processamento após determinada iteração exceder um certo limite pré-definido (*threshold*), podendo ter um ou mais nós, a depender do problema em questão.

Os principais algoritmos dessa classe são definidos de acordo com o comportamento dos dados ao serem inseridos no neurônio de entrada até o fim do processo, nó de saída.

Elas são divididas em três tipos: redes neurais *feedforward*, algoritmo de *backpropagation* e redes neurais convolucionais (Amazon, 2023).

Podemos determinar uma rede neural específica, definindo sua composição estrutural através da quantidade de neurônios em cada camada (topologia), característica dos nós e a regra de treinamento (padrão definido para o ajuste dos pesos). Essa correção nos pesos ocorrem em ciclos, ou seja, em uma apresentação completa de todos os N pares (entrada e saída) dos dados de treinamento e pode ser definida de dois modos:

- Modo Padrão: os pesos são corrigidos a cada nova observação da base de dados na rede, baseando-se apenas no erro atual, ou seja, ocorrem N correções a cada ciclo;
- Modo *Batch*: em cada ciclo tem-se apenas uma correção feita com base no erro médio de todas observações naquela iteração de treinamento.

Em se tratando do processo de aprendizagem, segue que a propriedade mais importante desse tipo de algoritmo é aprender com o ambiente, e isso depende diretamente da estrutura da rede. Essa estrutura construída com base no tipo de aprendizado, sendo ele supervisionado, não supervisionado ou por reforço (PONCE, 2023).

2.6.1 Perceptron Multicamadas

Seguindo (Silva *et al.*, 2010), as redes neurais do tipo *Perceptron* Multicamadas com o algoritmo de retropropagação do erro (*backpropagation*) no treinamento, são comumente as mais utilizadas e pertencem à classe de aprendizado supervisionado, sendo consideradas as arquiteturas mais versáteis, possuindo em sua estrutura, no mínimo duas camadas ocultas.

O algoritmo *backpropagation*, também conhecido como regra *delta generalizada*, utilizado no treinamento dessas redes, tem como base principal duas regras bem definidas que são empregadas em sequência, de forma a ajustar os pesos automaticamente a cada iteração, a fim de reduzir gradativamente os erros.

- Propagação adiante (*forward*): os dados $\{\boldsymbol{x_1},...,\boldsymbol{x_n}\}$ são passados para a camada de entrada e processados até a camada de saída, sem que haja atualização dos pesos sinápticos e limiares dos neurônios. Com as estimativas geradas, uma comparação é feita com as saídas esperadas, e, através dos erros gerados, é feito um ajuste nos pesos e limiares dos nós;

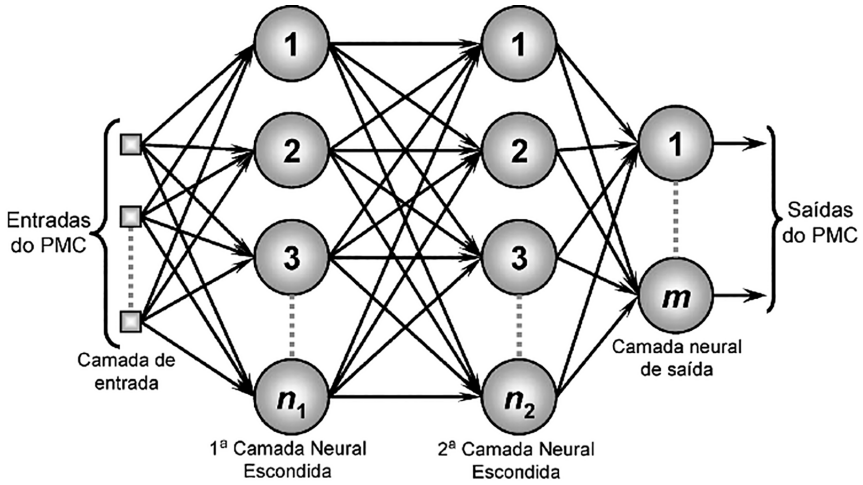

Figura 2.2: Representação de uma rede do tipo *PMC*. Fonte: Silva et al. (2010, p. 92).

- Propagação reversa (*backward*): aqui os ajustes dos pesos e limiares dos neurônios são feitos durante a execução da fase.

Backpropagtion

Para o uso do algoritmo *backpropagation* é preciso definirmos inicialmente algumas variáveis e parâmetros que configuram cada neurônio $\{j\}$ das $\{L\}$ camadas que alimentam uma função de ativação $g(.)$, contínua e diferenciável em todo domínio, tal como a logística ou tangente hiperbólica. Sintetizando a terminologia, temos:

- $W_{ji}^{(L)}$: matriz de pesos sinápticos que conecta o nerônio $\{j\}$ da camada $\{L\}$ com o neurônio $\{i\}$ da camada $\{(L-1)\}$;

- $l_j^{(L)}$: vetor de entradas ponderadas do neurônio $\{j\}$ da camada $\{L\}$;

- Y_j^L: vetor de saída do j-ésimo neurônio da camada $\{L\}$.

Após as especificações iniciais, é preciso definir uma função de erro para medir o desvio entre as saídas geradas e as respostas esperadas. Em geral, é utilizado o erro quadrático médio.

Com intuito de facilitar o entendimento dos passos realizados pelo algoritmo, dividimos sua descrição em duas etapas: ajuste da matriz de pesos sinápticos da camada de saída e o ajuste para as camadas intermediárias, a fim de minimizar o erro de predição. Para exemplificar tal fato, consideremos uma PMC com duas camadas ocultas, ou seja, com três matrizes de pesos, sendo W_{ji}^1 e W_{ji}^2 referentes às camadas intermediárias e W_{ji}^3 à

camada de saída. Dessa forma, temos que a primeira etapa acontece através do seguinte processo iterativo:

$$W_{ji}^{(3)}(t+1) = W_{ji}^{(3)}(t) + \eta \cdot \delta_j^{(3)} \cdot Y_i^{(2)},$$

onde η é a taxa de aprendizagem. E a segunda parte, para este caso em específico é dada pelas seguintes atualizações:

$$W_{ji}^{(2)}(t+1) = W_{ji}^{(2)}(t) + \eta \cdot \delta_j^{(2)} \cdot Y_i^{(1)}$$
$$W_{ji}^{(1)}(t+1) = W_{ji}^{(1)}(t) + \eta \cdot \delta_j^{(1)} \cdot x_i.$$

Em resumo, o algoritmo *backpropagation*, empregado para ajustar os pesos de uma rede neural com base no erro calculado durante a propagação reversa, utiliza a derivada do erro em relação aos parâmetros da rede para efetuar as atualizações, visando assim a redução do erro na próxima iteração. Adicionalmente, a técnica de otimização mais comumente empregada para realizar essas atualizações em um modelo que representa uma composição de funções é o gradiente descendente, indicando a taxa de variação em relação a uma variável. No contexto da rede neural, ao calcularmos o gradiente da função de custo em relação aos pesos, obtemos uma taxa que representa a mudança da função de custo para cada peso, orientando, dessa forma, a atualização dos pesos a fim de minimizar o erro.

2.6.2 Implementação Básica - PMC

1. Passar as amostras de treinamento $\{x_1, .., x_d\}$;
2. Passar os rótulos esperado dessas entradas $\{y\}$;
3. Inicializar com pequenos valores as matrizes de pesos para cada camada, $W_{ji}^{(L)}$;
4. Especificar a taxa de aprendizagem η e precisão ϵ;
5. Iniciar o contador de épocas;
6. Iterar até que $|E_m^{atual} - E_m^{anterior}| \leq \epsilon$:
   (a) $E_m^{anterior} \leftarrow E_m$;

(b) Para todas as amostras de treinamento:

i. Obter $l_j^{(L)}$ e $Y_j^{(L)}$ (passo forward);

ii. Determinar $\delta_j^{(L)}$ e Ajustar $W_{ji}^{(L)}$.

(c) Obter $Y_j^{(L)}$;

(d) $E_m^{atual} \leftarrow E_m$;

(e) época $\leftarrow$ época+1.

2.7 Avaliação dos modelos

Uma das partes mais importantes da modelagem de dados é avaliar o quão preciso foi o modelo em generalizar para novas observações. Para isso, utilizamos métricas de avaliação e podemos até empregar uma validação por estimativa intervalar. A seguir, descreveremos cada um dos métodos.

2.7.1 Métricas de avaliação

As métricas de regressão são utilizadas para avaliar o desempenho de modelos que buscam prever valores contínuos. Para este trabalho, consideramos quatro métricas distintas, de modo que cada uma delas captura diferentes nuances do resultado geral (James *et al.*, 2013).

1. **MAE (Erro Absoluto Médio):**
   - Fórmula: $MAE = \frac{1}{n}\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid$.
   - Vantagem: Fácil interpretação, pois representa a média absoluta dos erros, mantendo a escala original dos dados.
2. **MSE (Erro Quadrático Médio):**
   - Fórmula: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.
   - Vantagem: Penaliza com mais intensidade os maiores erros, ampliando a atenção de grandes discrepâncias.
3. **RMSE (Raiz do Erro Quadrático Médio):**
   - Fórmula: $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$.

- Vantagem: Fornece uma medida do erro na mesma escala que a variável resposta, facilitando a interpretação. Quanto maior a distância em relação ao MAE, temos um indicativo de erros muito discrepantes.

4. **R^2 (Coeficiente de Determinação):**

- Fórmula: $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$.
- Vantagem: Mede a proporção da variabilidade da variável dependente que é explicada pelo modelo. Varia de 0 a 1, sendo 1 indicativo de um ajuste perfeito.

2.7.2 Intervalo de confiança para o risco

Considerando os dados do conjunto de teste, de tamanho m, que não foram utilizados no treinamento do modelo, podemos determinar um intervalo de confiança para o risco da seguinte maneira:

$$\hat{R}(g) = \frac{1}{m}\sum_{k=1}^{m}\underbrace{(Y_k - g(\boldsymbol{X}_k))^2}_{W_k}.$$

Por meio do Teorema do Limite Central, para m suficientemente grande e sabendo que $\hat{R}(g)$ é uma média de variáveis i.i.d,

$$\hat{R}(g) \approx N\left(R(g), \frac{1}{m}\mathbb{V}[W_1]\right).$$

Desse modo, temos que $W_1, ..., W_m$ são i.i.d's e considerando a média $\bar{W} = \frac{1}{m}\sum_{k=1}^{m} W_k$, podemos estimar $\mathbb{V}[W_1]$ por:

$$\hat{\sigma^2} = \frac{1}{m}\sum_{k=1}^{m}(W_k - \bar{W})^2.$$

Portanto, o intervalo de confiança aproximado de 95% para o risco é:

$$\hat{R}(g) \pm 2\sqrt{\frac{1}{m}\hat{\sigma^2}}.$$

Além de termos uma estimativa intervalar para o risco, a equação anterior permite determinar o tamanho ideal para a divisão dos dados originais em conjunto de treino e conjunto de teste. O menor valor de m pode ser escolhido de forma que o intervalo de confiança para o risco seja tão pequeno quanto desejemos. Este método de escolha é ideal quando a amostra é grande (Izbicki e dos Santos, 2020).

Capítulo 3

Aplicação

3.1 Banco de dados

O banco de dados selecionado para este estudo foi utilizado em um projeto publicado no *Towards Data Science* e disponibilizado abertamente no **Github** do autor. Essas observações foram coletadas por meio de *Web Scraping* no ano de 2019 e trazem características de alguns usuários do Instagram que possuíam perfil público na plataforma durante o período de coleta das informações.

Entrando no detalhe dos dados, observamos que a base contém 20.668 observações e 73 atributos diretos, que trazem informações referentes à conta do usuário e das postagens feitas, tais como: área comercial, número de seguidores e seguidos, se a conta é verificada, se o *post* é um vídeo, número de curtidas, data e hora da publicação, entre outras.

Na etapa de entendimento do banco de dados, foi perceptível que diversas variáveis do conjunto de dados se dá pela criação de novos atributos com base em alguns já existentes, como por exemplo **intervalos do dia**, derivado da variável **hora do dia**. Além disso, há variáveis categóricas que foram tratadas de forma que cada classe se tornasse uma coluna, e sua ocorrência fosse indicada pelo valor 1; 0 caso contrário, como no caso do uso da função **get_dummies** da biblioteca **pandas** do python, o que tende aumentar significativamente a *esparsidade* da matriz de covariáveis, dependendo da quantidade de classes existentes.

As informações apresentadas estão em formato tabular e abrangem a maioria dos tipos de dados. Ou seja, incluem variáveis categóricas, numéricas e textuais, como no caso da legenda e descrição do conteúdo da imagem associada ao *post*. Essas variáveis foram tratadas por meio de técnicas de processamento de linguagem natural, com o objetivo de

gerar novas *features* para compor o modelo.

3.2 Análise Descritiva

Inicialmente, importamos todas as bibliotecas necessárias para trabalhar com a exploração dos dados, bem como a própria base disposta no *drive* e carregada para o ambiente *Google Colab*, onde foi desenvolvida todas etapas do processo.

Foi mencionado anteriormente que diversas variáveis do conjunto de dados se dá principalmente pelo tratamento que foi realizado em algumas variáveis categóricas, como no caso do dia da semana, intervalos de horas do dia e na categoria de negócio da conta, representando 7, 6 e 18 novos atributos gerados, respectivamente.

A partir desse entendimento dos dados e com base no formato de cada variável, particionamos os atributos em 7 principais grupos, que podem ser observados na tabela 3.1 com a respectiva quantidade em cada um deles.

Tabela 3.1: Grupo de variáveis.

Características	**Quantidade**
Post	12
Conta	11
Tempo	20
Área comercial	19
Identificadores	3
Não informativas	7
Resposta	1

Através do agrupamento acima e de acordo com o objetivo deste trabalho, determinamos quais variáveis seriam escolhidas para fazer parte deste estudo e que iriam passar por uma investigação mais detalhada.

Dessa maneira, rearranjamos a nova base de dados composta pelas 10 variáveis categóricas ou numéricas com informações das contas dos usuários, todas temporais, todas da área comercial, 2 referentes aos *posts* e a variável resposta, totalizando em 52 atributos.

Das 52 variáveis finais, temos que aquelas que são referentes às variáveis categóricas que já foram tratadas, trazem as mesmas informações que seu atributo original. Dessa forma, temos que 31 atributos podem ser analisados por meio de sua origem.

A coluna que se refere à data do *post*, pelo fato de não estarmos realizando um estudo que investiga características ao longo do tempo, consideramos essa variável apenas para

criar um indicador do quão ativa é a conta, ou seja, qual o tempo médio em dias que conta apresenta entre a publicação de um *post* e outro, restando explorar de maneira univariada, 20 *features*.

3.2.1 Estudando o comportamento das variáveis

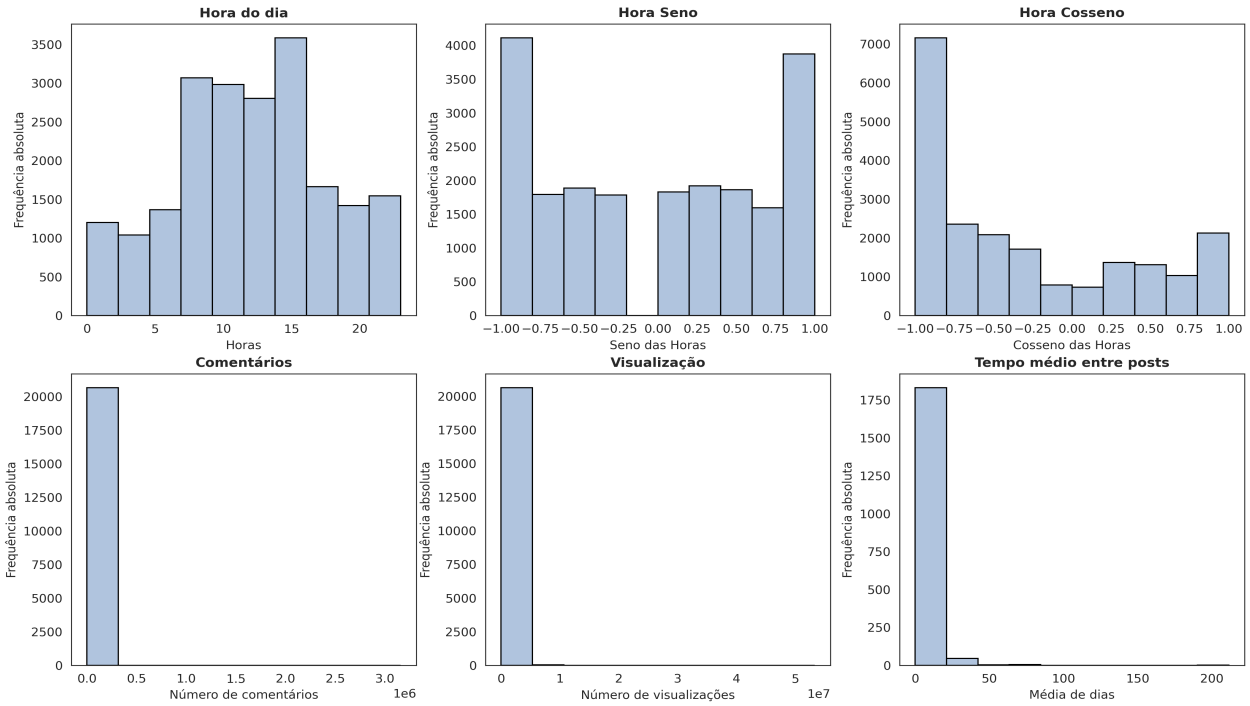

Figura 3.1: Histogramas das variáveis relacionadas aos *posts*.

Como podemos observar na primeira linha da Figura 3.1, temos informações relacionadas às horas em que um *post* foi feito. Os intervalos de classe com maiores frequências se concentram desde um pouco antes das 10 horas até por volta das 15 horas. Para as outras duas variáveis, referentes ao Seno e Cosseno das horas (que codificam a natureza cíclica do tempo, ou seja, 11h55 está próximo de 12h05, mas 23h está longe de 00h), temos que a primeira traz uma configuração em que concentra as maiores faixas de valores nos limites inferiores e superiores do intervalo, ou seja, em -1 e 1, enquanto a segunda possui uma certa assimetria à direita, concentrando a maior frequência na faixa próxima de -1.

Analisando a segunda linha da Figura 3.1, percebemos que as duas primeiras *features* que trazem as principais informações de um *post*, possuem um comportamento muito semelhante, concentrando seus valores medianos em 2.650 e 374.459, respectivamente.

Todavia, é perceptível uma distorção na distribuição, fazendo parecer com que todos os valores se concentrem em uma única faixa com valores baixos comparado ao *range* total. Isso ocorre devido à presença de alguns *posts* muito influentes, ou seja, que tiveram uma repercussão bem fora da realidade das demais postagens da base, podendo ser considerado como *outliers*.

Já o último gráfico da Figura 3.1 representa a variável criada com base no intervalo de dias entre *posts*. Dessa maneira, conseguimos mensurar o quão ativa é uma conta e podemos observar que, de fato, grande parte do público atua com regularidade na plataforma, havendo casos de tempo médio entre *posts* que não chegam nem a um dia. Em geral 75% das contas levam, em média, até 4 dias, e apenas 62 perfis dos 1887 têm, em média, um intervalo maior que 20 dias.

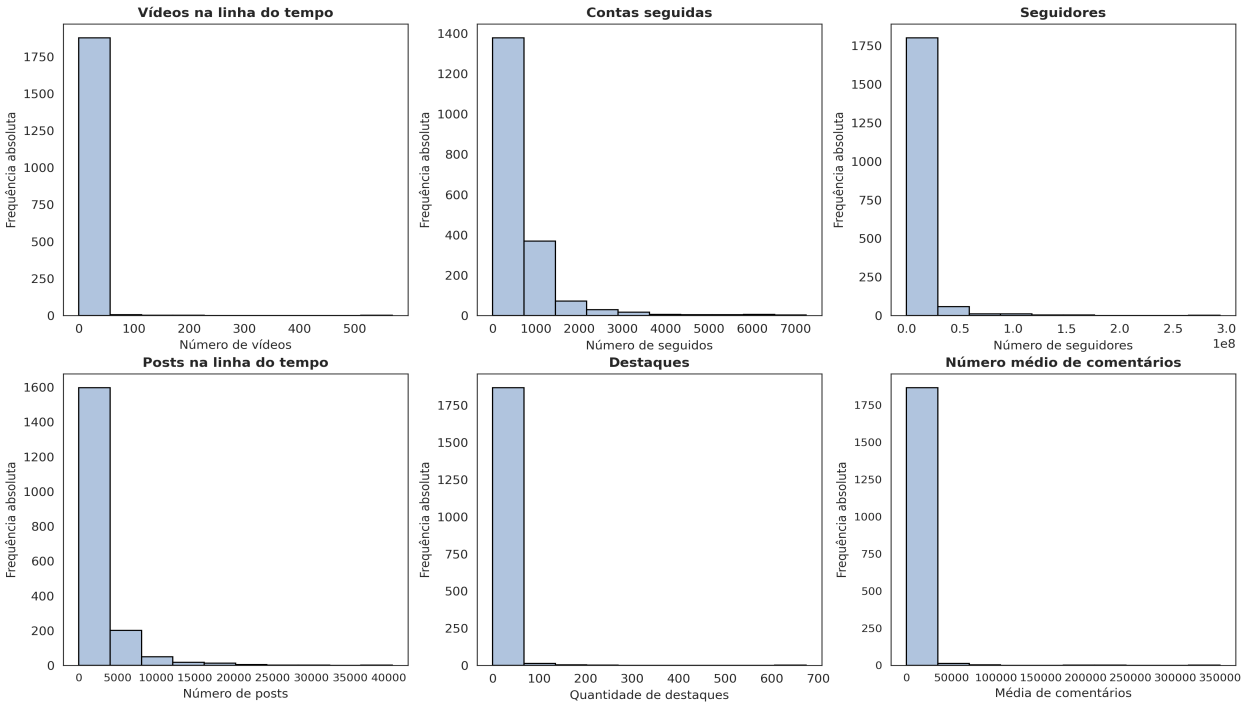

Figura 3.2: Histogramas das variáveis relacionadas à conta do usuário.

Explicando cada uma dessas variáveis relacionadas à conta do usuário e apresentando conjuntamente estatísticas obtidas durante a análise dessas informações, conforme ilustrado na Figura 3.2, observa-se que a quantidade de vídeos na linha do tempo se concentra próximo de zero. Além disso, constatamos que 75% dos usuários da base não seguem mais que 769 contas, atingindo no máximo 7.246, enquanto mantêm uma média de 8.350.909

seguidores. Em relação às postagens ao longo do tempo, há uma frequência significativa na classe até 500, com uma quantidade mediana de 1376 *posts*. No penúltimo gráfico, notamos que a classe com a maior ocorrência de destaques vai até 100 por perfil. Concluindo, o número médio de comentários por conta tem uma frequência predominante na classe que vai até 50.000, com algumas outras representações em torno de 200.000, indicando, muito provavelmente, contas de pessoas famosas e relevantes na plataforma.

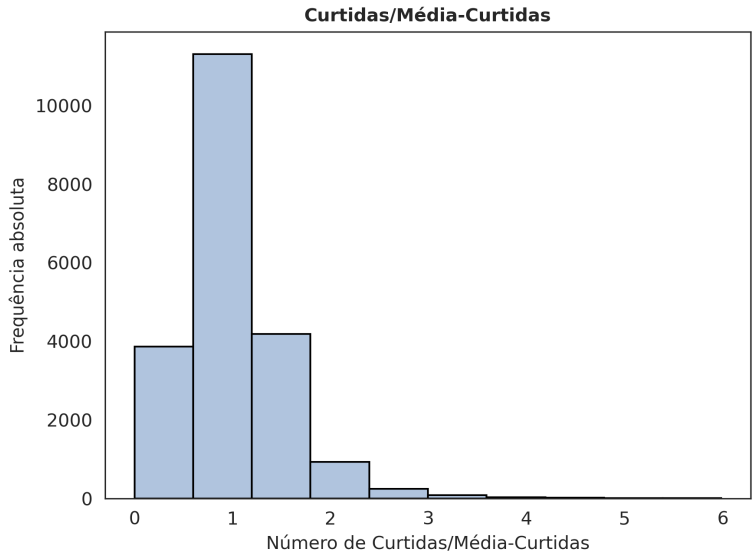

Figura 3.3: Histograma da variável resposta.

A partir da Figura 3.3, apresentamos individualmente a distribuição da variável alvo do estudo, na qual se destaca uma concentração mais significativa de **curtidas/média-curtidas** em torno de 1. Isso indica que, de maneira geral, os *posts* desses usuários costumam obter uma quantidade consistente de *likes* (engajamento uniforme).

Adentrando mais nos detalhes, apresentamos as principais métricas que descrevem o comportamento dessa *feature*:

Tabela 3.2: Estatísticas resumo da variável resposta.

Métrica	Mín	DP	Média	Q1	Mediana	Q3	Máx	Assimetria	Curtose
Valor	0.00	0.50	1.00	0.68	0.93	1.21	5.98	1.60	6.19

Sabemos que existem 20.668 valores preenchidos para essa variável, e, a partir disso, podemos observar que o valor mínimo, igual a zero, indica a presença de *posts* sem curtidas. Por outro lado, o valor máximo de 5.98 sugere que certas postagens ultrapassaram a média de curtidas da conta. Além disso, temos a média igual a 1 e uma mediana de 0.93,

apesar de serem próximas, podemos verificar uma assimetria positiva (à direita), como condiz com a própria métrica referente a isso, com valor de 1.60. Como característica adicional, temos a curtose de 6.19, o que implica no formato de curva leptocúrtica.

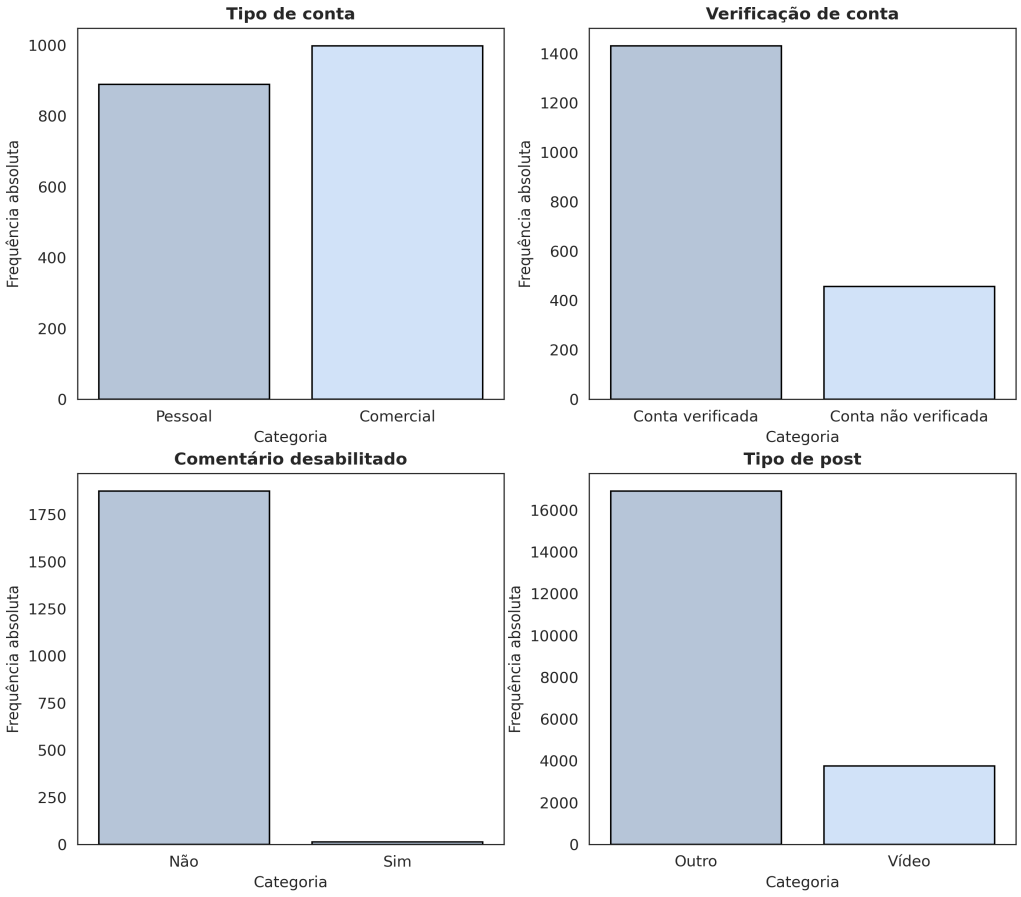

Figura 3.4: Gráficos de barras - variáveis relacionadas à conta e aos *posts*.

Os gráficos da Figura 3.4 destacam algumas características importantes sobre as postagens na plataforma. Como podemos observar, há praticamente uma distribuição equitativa entre contas pessoais e comerciais. No que diz respeito a perfis verificados, observa-se uma predominância desta classe, enquanto contas com os comentários desabilitados são praticamente inexistentes. Concluindo com outra informação sobre *posts*, podemos inferir que a maioria não são vídeos, havendo aproximadamente 4000 exemplares destes.

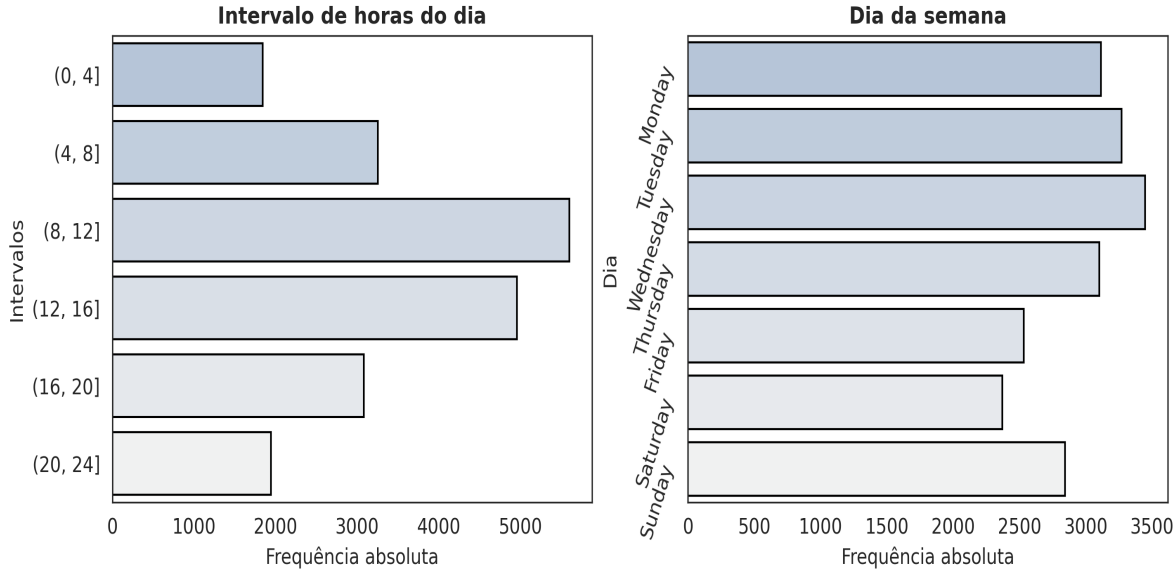

Figura 3.5: Gráficos de barras referentes ao tempo do *post*.

Na figura 3.5, os intervalos de horas do dia em que os *posts* são realizados mostram uma maior frequência na classe que abrange as horas entre 8h00 e 12h00, seguida pelo período das 12h00 às 16h00. Os intervalos antes e depois apresentam comportamentos semelhante nas caudas. Quanto aos dias com maior quantidade de *posts*, observa-se que o início da semana tem mais publicações do que o próprio fim de semana. Uma justificativa plausível seria um menor engajamento durante a semana, exigindo assim uma quantidade maior de postagens.

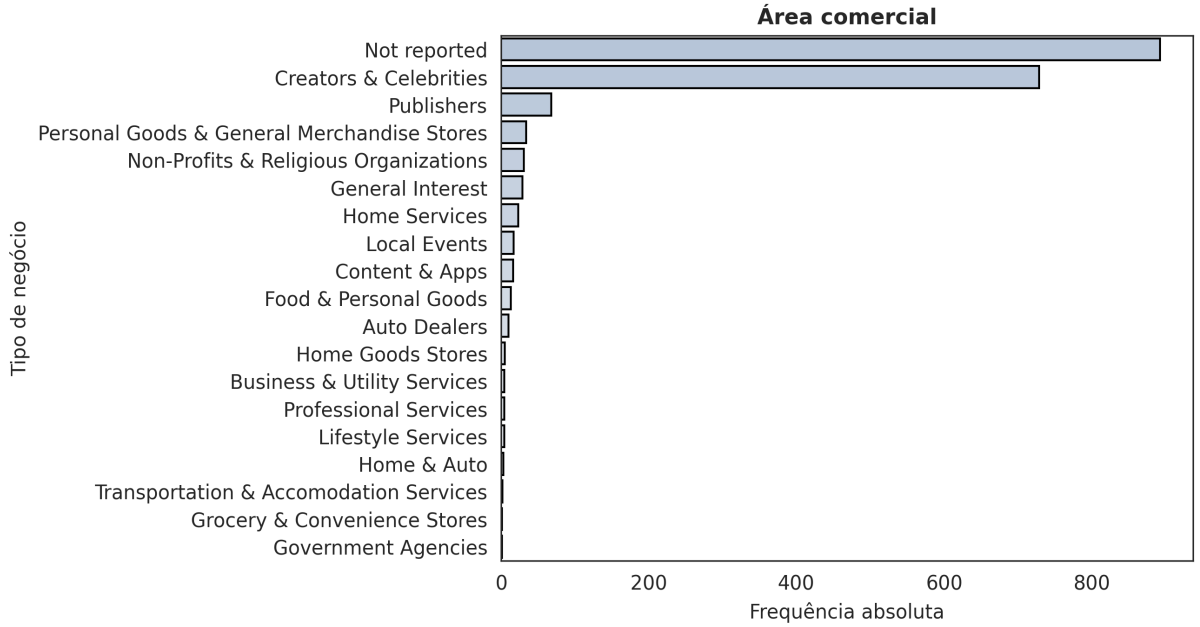

Figura 3.6: Gráfico de barras referente à área de negócio da conta.

Ao analisar o tipo de negócio ao qual a conta se encaixa, conforme mostrado na Figura 3.6, observamos que, na maioria dos casos, não foi relatado, indicando a possibilidade de atuação em diversas áreas simultaneamente. Em segundo lugar, notamos que uma grande parte dos usuários são criadores de conteúdo ou celebridades. Do mais, temos uma baixa frequência para os demais nichos.

3.2.2 Análise de correlação das variáveis

Na etapa final da análise exploratória, verificar a correlação entre as *features* é crucial em projetos de dados nos quais se espera uma relação linear das covariáveis com a variável resposta. No entanto, não podemos garantir tal relação neste caso específico, por isso, conduziremos apenas uma avaliação de correlação entre as variáveis contínuas independentes do conjunto de dados, com o objetivo de identificar covariáveis que apresentem maior similaridade entre si. Essa abordagem visa destacar casos que merecem atenção especial, proporcionando uma base sólida para a fase subsequente de engenharia de características (*feature engineering*). Vale ressaltar que, quando duas variáveis estão altamente correlacionadas linearmente, consideramos a criação de uma nova variável que capture de forma mais eficaz a informação subjacente. Este procedimento visa otimizar a representação dos dados e aprimorar o desempenho do modelo.

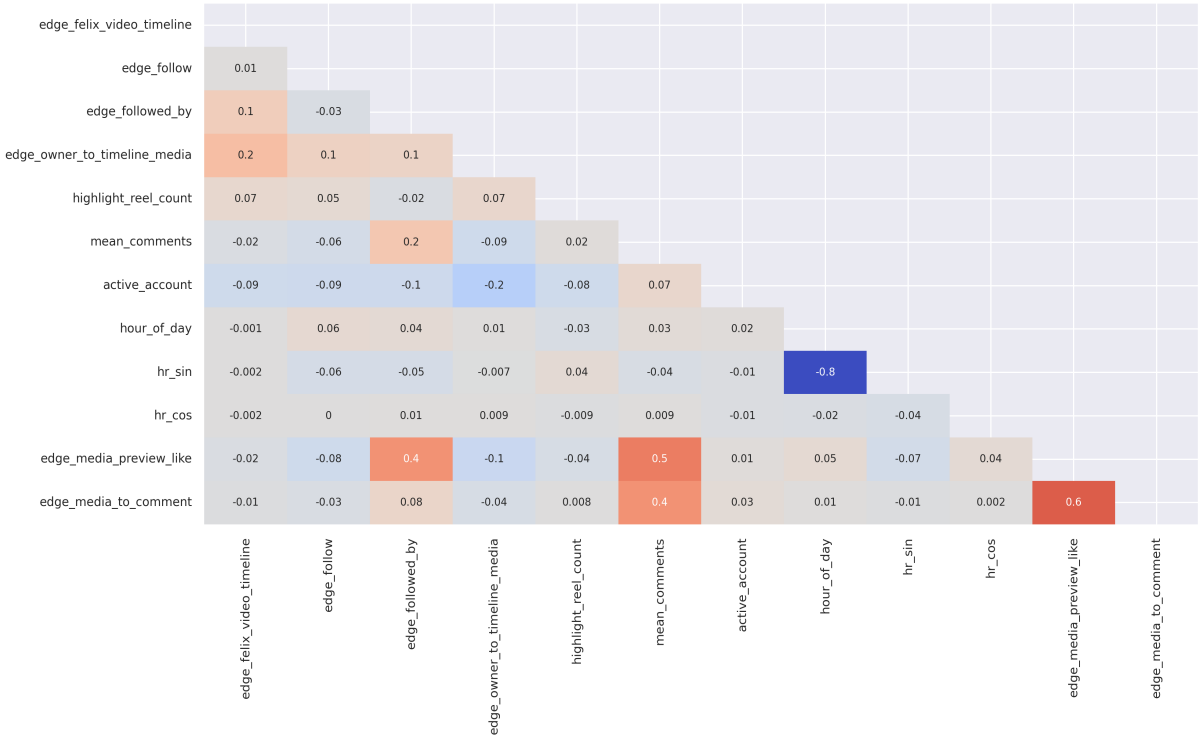

Figura 3.7: Mapa de calor das correlações para as *features* independentes.

Através da matriz de correlação apresentada na Figura 3.7, percebemos que há grupos específicos de covariáveis que se relacionam linearmente entre si, como por exemplo, os atributos relacionados ao tempo da postagem e informações do número de visualizações com o número de comentários.

Considerando que utilizaremos modelos robustos, eventuais pontos destacados na relação linear dessas covariáveis serão tratados internamente em suas estruturas. Embora esta investigação preliminar não proporcione um direcionamento completo sobre a contribuição das variáveis na explicação do rótulo, a fim de realizar uma seleção de variáveis, existem diversos métodos que podemos considerar para executar essa tarefa. Tais técnicas serão detalhadas mais amplamente na seção de pré-processamento.

3.3 Pré-processamento

3.3.1 Processamento de Linguagem Natural

Após realizar toda a análise, compreensão e correção de alguns valores nas variáveis do conjunto de dados, avançamos para a fase de pré-processamento. Nesta etapa, dedicamo-nos ao tratamento das variáveis textuais que serão consideradas em nosso ajuste. Para isso, procedemos com a etapa de processamento de linguagem natural, aplicando essa técnica tanto à *feature* que se refere à legenda do *post*, quanto à *feature* que contém a descrição do conteúdo relacionado ao *post*, como exemplificado por: *"Image may contain: 2 people, selfie and closeup"*.

Seguindo com a aplicação dessa técnica, primeiramente passamos pelo processo de *Bag of Words*, onde buscamos representar de forma simples a ocorrência ou não de determinadas palavras no texto, ignorando a ordem das mesmas e considerando apenas a frequência de cada uma delas. Para que as palavras representadas possam apresentar um significado expressivo para a aplicação, removemos do texto todas as palavras consideradas "*stopwords*"(palavras de conexão ou irrelevantes no contexto, como "e", "ou", "por", etc.). O resultado final é uma matriz onde cada linha corresponde ao texto referente à variável de um determinado *post*, e as colunas são os termos. Realizamos essa aplicação para as informações mencionadas anteriormente: legenda e conteúdo da imagem, delimitando um total de 100 termos mais importantes para cada uma delas, totalizando 200 novos atributos na base de dados.

Em seguida, para as mesmas *features* acima, utilizamos a própria representação de *Bag of Words* para a aplicação da análise de sentimentos, obtendo duas novas informações que fornecem o teor emocional referente a cada *post*, tanto pelo que foi descrito na legenda quanto pelo conteúdo publicado. Além disso, para a variável que descreve o conteúdo da imagem, tínhamos a marcação da quantidade de pessoas que o *post* apresentava; capturamos essa informação, criando mais uma nova *feature*.

Essa informação complementará o estudo, assim como a variável *numfaces*, que foi extraída das postagens para representar o número de faces detectadas em cada uma das imagens. Essa variável estava disponibilizada em uma base de dados à parte, onde -1 representa a não identificação, 0 significa *sem faces*, e os demais valores pertencem aos números naturais positivos. Realizados todos esses processamentos, ficamos com uma base de dados com 20.668 linhas por 252 colunas. Todavia, fomos além e seguimos com a criação de novos atributos.

3.3.2 Engenharia de atributos

Nesta etapa, seguimos com a construção de mais alguns atributos que possam contribuir ainda mais em nossa análise. Inicialmente, obtemos alguns indicadores importantes usando as principais informações dos dados, tais como estatísticas resumo (mínimo, máximo, média, mediana, desvio padrão, contagem) das variáveis *edge_media_to_comment* e *edge_media_preview*, as quais passaram a contribuir ainda mais com o modelo. Além disso, após realizar diversos testes de combinações de variáveis e explorar diferentes métodos para efetuá-los, conseguimos identificar sete novos indicadores que maximizaram a performance do ajuste. Um exemplo disso é a diferença entre o número de visualizações e o número de comentários, dividido pelo desvio padrão de visualizações. Dessa forma, concluímos o escopo da base de dados final com 272 atributos. No entanto, realizaremos uma etapa de seleção de variáveis para remover informações que não contribuem significativamente para a explicação da variável alvo.

Com a base de dados final em mãos, para evitar influência de escala no ajuste do modelo, optamos por normalizar as variáveis preditoras contínuas, de forma que todas pertençam ao intervalo $(0, 1)$.

A fórmula utilizada foi a Min-Max, dada por:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}.$$

3.3.3 Seleção de variáveis

Após a construção do *book de variáveis* e a sua incorporação à nossa base de dados original, seguimos para a etapa de seleção de variáveis. Existem várias maneiras de realizar esse procedimento, desde o uso da matriz de correlação (quando há uma relação linear entre as covariáveis e a variável de resposta) até métodos mais específicos, como o *stepwise*, que envolve a inclusão ou exclusão sistemática de variáveis no ajuste, com base em critérios específicos, como o valor-p ou critérios de desempenho do modelo. Essa seleção também pode ser conduzida pelo método de *feature importance*, que retorna o grau de importância de cada atributo. Geralmente, florestas aleatórias são utilizadas para essa tarefa; no entanto, algoritmos que possuem árvores em sua estrutura fornecem esse *ranking* das variáveis juntamente com suas respectivas contribuições.

No nosso caso, dada a ausência de garantia quanto a uma relação linear entre as covariáveis e o rótulo, e considerando a utilização de algoritmos mais abrangentes, como ***ensembles*** e **redes neurais**, optaremos pela última técnica mencionada. Em particular, faremos a escolha de utilizar o algoritmo LightGBM com parâmetros ajustados. Essa decisão visa evitar qualquer enviesamento no ajuste. O objetivo é conduzir a seleção de variáveis com base no conjunto de treinamento, resultando em uma base de dados que inclua apenas os principais atributos, aqueles que proporcionam o maior ganho na explicação do rótulo.

Tabela 3.3: *Features* selecionadas.

Features	**Importância relativa**	**Importância acumulada**
number_of_likes_mean	v. resposta	-
dstdw_mean_preview_comment	0.16	0.16
dstdw_edge_preview_comment	0.15	0.31
sub_edge_mean_preview	0.10	0.40
sub_edge_median_preview	0.06	0.46
des_edge_mean_comment	0.05	0.51
div_mean_median_preview	0.05	0.56
mean_preview	0.04	0.60
edge_media_preview	0.03	0.63
min_preview	0.02	0.65
median_preview	0.02	0.66
std_preview	0.02	0.68
active_account	0.02	0.70
edge_follow	0.02	0.71
edge_owner_to_timeline_media	0.02	0.73
edge_media_to_comment	0.01	0.74
max_preview	0.01	0.75
edge_felix_video_timeline	0.01	0.77
div_edge_mean_comment	0.01	0.78
pontos_bow	0.01	0.79
edge_followed_by	0.01	0.80
mean_comment	0.01	0.81
std_comment	0.01	0.82
hr_cos	0.01	0.83
sentiment_legend	0.01	0.84
max_comment	0.01	0.85
highlight_reel_count	0.01	0.86
min_comment	0.01	0.87
median_comment	0.01	0.87
hour_of_day	0.01	0.88
legend_bow_the	0.00	0.89
legend_bow_of	0.00	0.89
legend_bow_we	0.00	0.89
hr_sin	0.00	0.90
num_faces	0.00	0.90
people	0.00	0.90

Como podemos observar, as duas variáveis que, de longe, mais contribuem para a explicação do rótulo foram criadas no processo de *feature engineering*. Assim, temos que a diferença entre a média de visualizações e a média de comentários, dividida pelo desvio padrão de visualizações, bem como a diferença entre o número de visualizações e o número de comentários, dividida pelo desvio padrão de visualizações, são as características mais

influentes ao tentarmos explicar a popularidade de uma postagem, segundo este estudo. No entanto, os demais indicadores construídos, assim como o quão ativa a pessoa é na plataforma, o número de seguidores do perfil, a quantidade de comentários, a variável de sentimento extraída da legenda, entre outras informações, ainda contribuem para a explicação do nosso objetivo.

Assim sendo, com base na tabela de importância para cada *feature*, foram selecionadas as 35 primeiras, abrangendo desde *dstdw_mean_preview_comment* até *people*, uma vez que consideramos as mais significativas para o estudo. Essa seleção engloba uma contribuição de 90% na explicação do rótulo.

3.4 Modelagem

Feita toda a análise descritiva e exploratória dos dados para identificar seus comportamentos, disposição das informações e possíveis erros ou ausências, etc., seguimos para a etapa de pré-processamento. Nessa fase, aplicamos Processamento de Linguagem Natural (PLN) para extrair os atributos de interesse das variáveis textuais e desenvolvemos técnicas de engenharia de atributos para criar novas *features*, aplicando padronização às variáveis com formato contínuo. Em seguida, realizamos a seleção final das variáveis mais relevantes utilizando o método de *Feature Importance* do próprio algoritmo LightGBM.

Com base nesses resultados, observamos que dois indicadores criados a partir das variáveis já existentes foram claramente os mais significativos na explicação da popularidade de uma postagem. Entretanto, decidimos considerar apenas 35 atributos de um conjunto de 272, o que representa 90% de importância na explicação da variável resposta. Após diversas manipulações e ajustes de modelos, conseguimos melhorar significativamente o resultado da predição.

Para aprofundar a análise e aprimorar ainda mais os resultados, identificamos contas de usuários com relativamente poucos seguidores, mas com alta relevância na plataforma. Por exemplo, algumas contas possuíam apenas 16 seguidores, mas apresentavam um índice de popularidade de 2.33. Esse índice indica que o respectivo *post* foi 1.33 vezes mais relevante do que a média da conta. No entanto, ao realizar uma análise direta, considerando que o usuário atingisse o maior número de *likes*, correspondente ao máximo de seguidores da conta, resultaria em apenas 16 curtidas. Multiplicando esse valor pelo índice, obteríamos aproximadamente 22 curtidas, o que não é suficiente para popularizar uma

postagem na prática.

Diante disso, optamos por considerar apenas as contas que possuem mais de 1000 seguidores, o que contribuiu significativamente para melhorar nossos resultados. Exceto pela remoção dos valores com as características anteriores, não realizamos a exclusão dos *outliers* presentes na variável resposta, pois eles representam informações relevantes no estudo, indicando casos em que um *post* obteve um engajamento acima da média da conta.

Após a conclusão de todas as etapas de preparação dos dados, dividimos a base final, alocando 80% dos dados para o conjunto de treinamento e 20% para o conjunto de teste, totalizando 16.365 e 4.092 observações, respectivamente. Na fase de treinamento, aplicamos a validação cruzada com 5 *folds* e, posteriormente, validamos o ajuste usando as informações reservadas para teste. Avaliamos cada uma dessas etapas por meio de quatro métricas de regressão: erro absoluto médio, erro quadrático médio, raiz quadrada do erro quadrático médio e o coeficiente de determinação.

3.4.1 Resultado - Algoritmo LightGBM

Ajustamos o algoritmo LightGBM com otimização de hiperparâmetros pelo método de *Random Search*, realizando 50 iterações. Aplicamos o modelo na base de treinamento, considerando a técnica de validação cruzada para $5 - folds$. Em seguida aplicamos o modelo considerado satisfatório na base separada para teste.

A seguir, apresentaremos os parâmetros que foram otimizados com seus respectivos espaços de possibilidade:

Tabela 3.4: Parâmetros otimizados do LightGBM.

Hiperparâmetro	**Espaço de busca**
min_child_samples	(1,100)
num_leaves	(2,128)
colsample_bytree	(0.1,1.0)
subsample	(0.05,1.0)
learning_rate	(1e-3,1e-1)

Na sequência, apresentamos os principais parâmetros considerados no modelo:

Tabela 3.5: Principais parâmetros do LightGBM.

Hiperparâmetro	**Valor**
boosting_type	'gbdt'
importance_type	'split'
learning_rate	0.09
subsample	0.57
colsample_bytree	0.92
max_depth	-1
min_child_samples	15
n_estimators	100
num_leaves	126
subsample_for_bin	200.000

Aplicando o modelo treinado no conjunto de teste, obtivemos os seguintes resultados:

Tabela 3.6: Resultado LightGBM.

Métrica	**Treino**	**Teste**	**IC**(95%)
MAE	0.02	0.02	(0.01, 0.02)
MSE	0.01	0.01	(0.01, 0.01)
RMSE	0.05	0.04	(0.04, 0.05)
R^2	0.99	0.99	(0.99, 0.99)

A seguir, temos uma representação da dispersão dos valores reais em comparação com os valores previstos para cada entrada na base de teste.

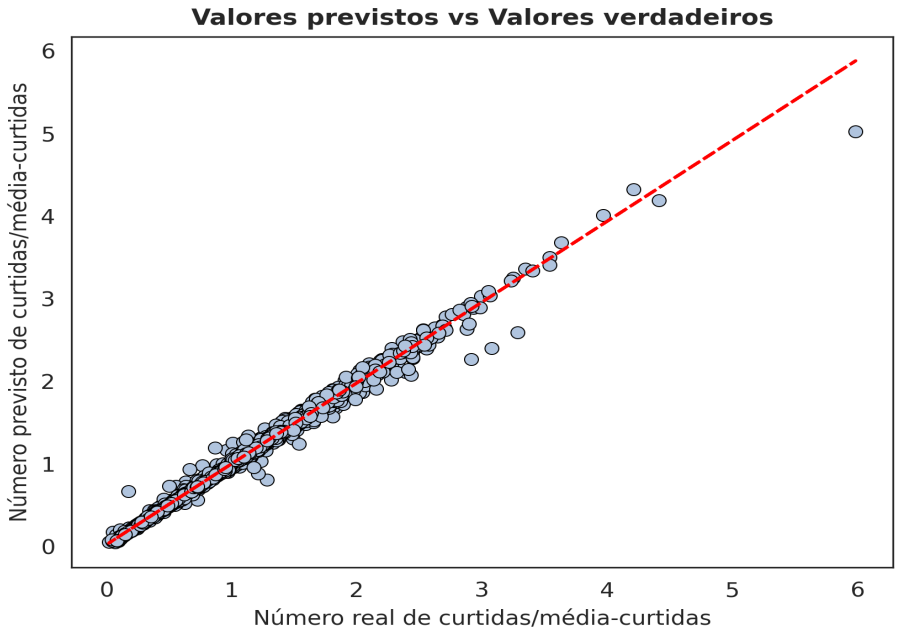

Figura 3.8: Valores reais vs Valores preditos - LightGBM.

Através da Figura 3.8, observamos que de acordo com o ajuste realizado, alcançamos uma excelente performance com o modelo. Das 4.092 observações de teste, praticamente todas foram previstas com precisão, aproximando-se da reta que representa o valor ideal de previsão. Visualmente, identificamos apenas três pontos que se encontram entre os valores-alvo de 3 e 4, os quais se afastam mais da reta, e o valor mais elevado na base, que se aproxima de 6 e foi previsto aproximadamente como 5. Exceto por esses casos mais específicos, a grande maioria dos valores foi prevista com baixo erro para esta aplicação.

3.4.2 Resultado - Algoritmo Perceptron Multicamadas

Para o algoritmo Perceptron Multicamadas, também realizamos um ajuste de parâmetros através do método *Random Search*, considerando 50 iterações. Isso se deve ao fato de buscarmos aprimorar os resultados em relação ao caso *default*, uma vez que a performance foi consideravelmente inferior quando comparada ao modelo de *Boosting*. Com os principais parâmetros definidos, ajustamos o algoritmo PMC na base de treinamento utilizando a técnica de validação cruzada com 5-*folds*. Em seguida, utilizamos o modelo já treinado para realizar a previsão no conjunto de teste. A seguir, forneceremos uma explicação mais detalhada desses passos.

Os parâmetros otimizados foram os seguintes, com os respectivos espaços de possibilidades:

Tabela 3.7: Parâmetros otimizados do PMC.

Hiperparâmetro	**Espaço de busca**
batch_size	(16,128)
n_neurons_per_layer	(100,500)
max_iter	(500,2000)
activation	(‘relu’,‘tanh’)
solver	(‘sgd’, ‘adam’)
learning_rate	(‘constant’,‘adaptive’)
tol	(1e-5, 1e-2) - log_uniform
alpha	(0.0001,0.05) - log_uniform

Dentre todos os parâmetros otimizados, determinamos que a melhor combinação obtida, juntamente com alguns parâmetros padrão, é a seguinte:

Tabela 3.8: Principais parâmetros do PMC.

Hiperparâmetro	**Valor**
activation	'tanh'
solver	'adam'
learning_rate	'constant'
n_iter_no_change	10
validation_fraction	0.1
batch_size	128
hidden_layer_sizes	500
learning_rate_init	0.001
epsilon	1e-08
tol	0.0003
max_iter	2000

A seguir, apresentamos os resultados do modelo aplicado durante o treino com validação cruzada e no conjunto de teste:

Tabela 3.9: Resultado PMC.

Métrica	**Treino**	**Teste**	**IC**(95%)
MAE	0.14	0.13	(0.13, 0.15)
MSE	0.04	0.03	(0.03, 0.05)
RMSE	0.20	0.18	(0.20, 0.21)
R^2	0.82	0.85	(0.82, 0.85)

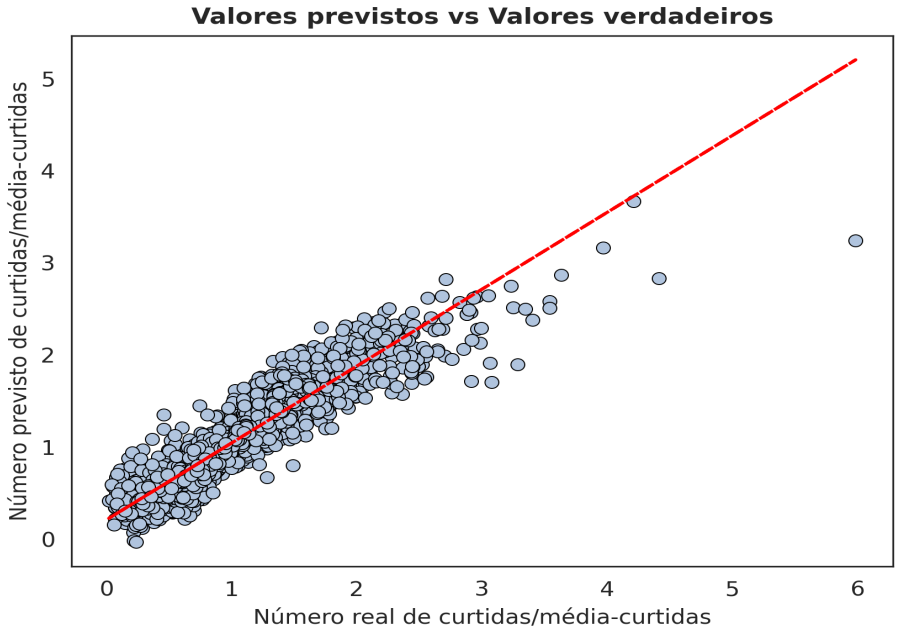

Figura 3.9: Valores reais vs Valores preditos - PMC.

Como podemos observar na Figura 3.9, a previsão realizada pela rede neural apresentou maior variabilidade, tanto nos valores mais comuns da base de dados quanto nos valores que destacariam a popularidade de uma postagem, como evidenciado pela dispersão dos pontos.

3.5 Seleção do modelo final

A partir dos resultados apresentados por cada um dos modelos anteriores, ficou evidente que a melhor performance foi obtida pelo algoritmo LightGBM, em comparação com o Perceptron Multicamadas, ambos submetidos ao processo de ajuste de parâmetros.

De forma geral, esperávamos que as redes neurais proporcionassem os melhores resultados, ou pelo menos mais próximos à efetividade que o algoritmo de *boosting* obteve, devido à sua alta complexidade e capacidade de adaptação à relação presente nos dados. No entanto, realizar testes para as inúmeras combinações dos parâmetros disponíveis a fim de encontrar a melhor estrutura para os dados em questão é computacionalmente custoso e, mesmo após realizar esse processo por longas horas, não encontramos uma combinação de hiperparâmetros que ofereça melhores previsões para superar os resultados do LightGBM.

Dessa maneira, pelo simples fato de o algoritmo de *boosting* selecionado para este estudo ter apresentado um resultado significativamente melhor em todas as métricas utilizadas, e com um menor custo computacional, o LightGBM foi escolhido como modelo final. Vale lembrar que esse resultado é específico para esse conjunto de dados.

Capítulo 4

Conclusões

Este trabalho foi motivado pelo desenvolvimento de uma aplicação de *machine learning* no contexto das redes sociais, especificamente no Instagram. Isso se deve ao fato de que esse aplicativo configura um cenário no qual as pessoas fornecem diversas informações sobre seus perfis e interesses, tornando-se uma rica fonte de dados. O objetivo dessas plataformas é gerar conteúdo assertivo para cada tipo de público, proporcionando engajamento às contas dos usuários e transformando-os em influenciadores digitais. Esses influenciadores utilizam sua popularidade para realizar divulgações de marcas e produtos em parceria com empresas.

Tomando como referência o contexto apresentado, desenvolvemos uma formulação inovadora que utiliza os métodos mais atuais de previsão em uma aplicação real. Com isso, os potenciais resultados obtidos por meio desse tipo de estudo podem gerar indicadores relevantes neste mercado. Através de um gerenciamento adequado, é possível otimizar as publicações para alcançar altos níveis de engajamento, resultando em maior rentabilidade, tanto para os *influencers* digitais como para as empresas.

Dessa maneira, com o objetivo de prever a popularidade de um *post* no Instagram e otimizar as publicações para aumentar o engajamento, estudamos o perfil público de 1887 usuários da plataforma. As informações coletadas estão relacionadas tanto à conta quanto às publicações realizadas por esses usuários. A partir da base de dados disponível, selecionamos inicialmente as variáveis categóricas e numéricas, conduzindo uma análise descritiva e exploratória. Esse processo nos permitiu compreender o comportamento de cada uma das *features*, identificar *outliers*, tratar valores faltantes na base e analisar as relações entre as covariáveis, buscando compreender possíveis relações lineares.

Feito isso, avançamos para a etapa de pré-processamento, na qual aplicamos o proces-

samento de linguagem natural nas variáveis textuais com o objetivo de obter novas informações. Em seguida, procedemos à etapa de *feature engineering*, utilizando as variáveis já existentes para criar novos indicadores, visando aprimorar a *performance* do modelo. Consideramos a padronização para as variáveis contínuas. Após essas etapas, empregamos o método de *Feature Importance* para selecionar apenas os atributos mais relevantes na justificativa do rótulo, resultando em um conjunto final de 35 covariáveis.

Assim, observamos que as principais informações para prever o engajamento de um *post* neste estudo foram principalmente provenientes dos indicadores criados na etapa de *feature engineering*, juntamente com algumas variáveis obtidas do processamento de linguagem natural. Além disso, foram consideradas as variáveis originais, que correspondem ao número de visualizações, número de comentários e número de seguidores da conta.

Após concluir a etapa de preparação dos dados, aplicamos os modelos LightGBM e Perceptron Multicamadas, ambos com os parâmetros ajustados. O resultado do primeiro ajuste foi surpreendentemente superior ao das redes neurais, alcançando um **RMSE** de 0.04 e $\mathbf{R}^2$ de 0.99, enquanto o PMC atingiu **RMSE** de 0.18 e $\mathbf{R}^2$ de 0.85. Portanto, optamos por selecionar o algoritmo de *boosting* como nosso modelo final, pois apresentou uma melhor capacidade de generalização ao ser submetido a previsões de novos dados. Ressalta-se que os novas observações precisam possuir as mesmas configurações dos dados deste estudo, o que é particularmente relevante neste caso, pois temos disponíveis previamente informações gerais sobre a publicação.

Em conclusão, tornou-se evidente que, dentre todas as fases delineadas no projeto, lidar com a seleção de variáveis e a aplicação do processo de otimização de parâmetros são contribuições fundamentais para a melhoria do modelo. Entretanto, é perceptível que o entendimento dos dados e a maneira como são manipulados – incluindo a criação de novos atributos e a remoção de valores que não estejam alinhados com o objetivo final da aplicação – pode ser considerado como a principal técnica para aprimorar os resultados de predição do modelo.

Referências Bibliográficas

Amazon (2023). O que é uma rede neural? Disponível em: https://aws.amazon.com/pt/what-is/neural-network/#:~:text=Uma%20rede%20neural%20%C3%A9%20um,camadas%2C%20semelhante%20ao%20c%C3%A9rebro%20humano. Acessado: 08-07-2023.

Brownlee, J. (2020). Hyperparameter optimization with random search and grid search. *Machine Learning Mastery*. Acessado: 14-01-2023.

Canaltech (2023). Tudo sobre instagram - história e notícias. Disponível em: https://canaltech.com.br/empresa/instagram/. Acessado: 28-05-2023.

Carta, S., Podda, A. S., Recupero, D. R., Saia, R. e Usai, G. (2020). Popularity prediction of instagram posts. *Information*, **11**(9), 453.

Corrêa, D. M. (2021). Feature engineering: preparando dados para aprendizado de máquina. *Blog Ateliware*. Acessado: 14-01-2023.

Exame (2022). Eles dominaram o mercado: a força dos influenciadores digitais. Disponível em: https://exame.com/colunistas/bora-varejo/eles-dominaram-o-mercado-a-forca-dos-influenciadores-digitais/. Acessado: 28-05-2023.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, páginas 1189–1232.

Hastie, T., Tibshirani, R., Friedman, J. H. e Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

Izbicki, R. e dos Santos, T. M. (2020). *Aprendizado de máquina: uma abordagem estatística*. Rafael Izbicki.

James, G., Witten, D., Hastie, T., Tibshirani, R. *et al.* (2013). *An introduction to statistical learning*, volume 112. Springer.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. e Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, **30**.

Microsoft (2023). Lightgbm. Disponível em: https://lightgbm.readthedocs.io/en/latest/index.html. Acessado: 22-08-2023.

Nerdweb (2022). Instagram reels: Entenda o que é esse recurso que está mudando a forma de usar o instagram. Disponível em: https://nerdweb.com.br/artigos/2022/07/o-que-e-reels-o-recurso-que-esta-mudando-a-forma-de-usar-o-instagram.html. Acessado: 28-05-2023.

PONCE, A. L. (2023). Redes neurais artificiais. Disponível em: https://sites.icmc.usp.br/andre/research/neural/. Acessado: 08-07-2023.

Saini, A. (2021). Gradient boosting algorithm: A complete guide for beginners. Disponível em: https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/. Acessado: 17-08-2023.

Silva, I. N. d., Spatti, D. H. e Flauzino, R. A. (2010). Redes neurais artificiais para engenharia e ciências aplicadas.

Silva, P. H. (2021). Textblob: Alternativa para processamento de linguagem natural. *Alura Artigos*. Acessado: 27-12-2023.

VOLPATO, B. (2023). Ranking: as redes sociais mais usadas no brasil e no mundo em 2023, com insights, ferramentas e materiais. Disponível em: https://resultadosdigitais.com.br/marketing/redes-sociais-mais-usadas-no-brasil/. Acessado: 28-05-2023.

ZANE, M. (2022). Marketing digital não é marketing de influência: entenda as diferenças. Disponível em: https://novaescolademarketing.com.br/marketing-digital-nao-e-marketing-de-influencia/. Acessado: 28-05-2023.

Apêndice A

Códigos

A.1 Importação das bibliotecas e da base

```
!pip install lightgbm -q
!pip install scikit-optimize -q

import math
import langid
import numpy as np
import pandas as pd
import datetime as dt
import seaborn as sns
import lightgbm as lgb
import matplotlib.pyplot as plt

from sklearn import metrics
from datetime import datetime
from skopt import dummy_minimize
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

from skopt.space import Real, Integer, Categorical
from skopt.utils import use_named_args
from skopt import gp_minimize
from sklearn import preprocessing
from textblob import TextBlob
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from googletrans import Translator
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
translator = Translator()
import nltk
nltk.download('stopwords')

path_df = '/content/drive/MyDrive/TGB/Dezembro-2023/Bases
/InstagramDatasetFinalVersion.csv'
df = pd.read_csv(path_df, encoding='utf-8-sig')
```

A.2 Exploração e tratamento dos dados

```
# converter formato das variáveis booleanas
df[['is_business_account','is_verified','comments_disabled','is_video']] =
df[['is_business_account','is_verified','comments_disabled','is_video']]./
apply(lambda x: x.astype(int))

# criar a variável de tempo entre posts com a unidade em dias
taken_at_timestamp = []
for i in range(len(df['taken_at_timestamp'])):
  d = pd.to_datetime(datetime.fromtimestamp(df['taken_at_timestamp'][i])./
```

```
  strftime('%Y-%m-%d, %H:%M:%S'))
  taken_at_timestamp.append(d)
df['taken_at_timestamp'] = taken_at_timestamp

conta = []
ativa = []
l_names = list(df.username.unique())
for i in range(len(l_names)):
  users = [l_names[i]]
  atv = ((pd.DataFrame(np.diff(df.query("username.isin(@users)")[['username',
  'taken_at_timestamp']].reset_index().sort_values(by=['username',
  'taken_at_timestamp'],ascending=[True,True]).taken_at_timestamp))
  /pd.Timedelta(hours=1))/24).mean()[0]
  conta.append(users)
  ativa.append(atv)

account = pd.DataFrame(conta,columns=['username'])
active = pd.DataFrame(ativa,columns=['active_account'])
active_account = pd.concat([account,active],axis=1)

df = pd.merge(df,active_account,how='left',on=['username'])
df['active_account'] = df['active_account'].fillna(0)

df['Not_reported'] = 0
df.loc[df['business_category_name'].isnull(),'Not_reported'] = 1

# Dicionário para mapear os novos nomes de coluna
novo_nome_coluna = [coluna.replace(' ', '_').replace('&', '_') for coluna in
colunas_originais]
novo_nome_coluna2 = [coluna.replace('___', '_') for coluna in novo_nome_coluna]
novo_nome_coluna3 = {colunas_originais[i]: novo_nome_coluna2[i]  for i in
range(len(colunas_originais))}
```

```
# preenchendo valores faltantes variável HourBin
df.loc[df['HourBin'].isnull(),'(20, 24]'] = 1

# base de dados final da etapa-1
df_final = df[['username',
               ### principais ###
               'edge_liked_by','edge_media_preview_like','edge_media_to_comment',
               'mean_likes','mean_comments', 'edge_follow','edge_followed_by',
               'edge_owner_to_timeline_media','edge_felix_video_timeline',
               'number_of_likes-median','highlight_reel_count',
                ### booleanas ###
               'is_business_account','is_verified','comments_disabled','is_video',
                ### tempo ###
               'active_account','Friday', 'Monday', 'Saturday',
               'Sunday', 'Thursday', 'Tuesday', 'Wednesday', 'hour_of_day',
               'hr_sin',
               'hr_cos','a0f4','a4f8','a8f12','a12f16','a16f20','a20f24',
                ### negocio ###
               'Auto_Dealers', 'Business_Utility_Services', 'Content_Apps',
               'Creators_Celebrities', 'Food_Personal_Goods', 'General_Interest',
               'Government_Agencies', 'Grocery_Convenience_Stores', 'Home_Auto',
               'Home_Goods_Stores', 'Home_Services', 'Lifestyle_Services',
               'Local_Events', 'Non-Profits_Religious_Organizations',
               'Personal_Goods_General_Merchandise_Stores', 'Professional_Services'
               'Publishers', 'Transportation_Accomodation_Services','Not_reported',
                ### resposta ###
               'number_of_likes/mean']].copy()

df_est_preview = df_limpo.groupby('username').agg({'edge_media_preview':
['mean','median','std','min','max','count']}).reset_index()
df_est_preview.columns = [df_est_preview.columns[i][0] if i == 0 else
df_est_preview.columns[i][1]+'_preview' for i in range(len(df_est_preview.columns))
df_limpo_est_p = pd.merge(df_limpo,df_est_preview,how='left',on=['username'])
```

```
df_est_preview = df_limpo.groupby('username').agg({'edge_media_preview':
['mean','median','std','min','max','count']}).reset_index()
df_est_preview.columns = [df_est_preview.columns[i][0] if i == 0
else df_est_preview.columns[i][1]+'_preview' for i in
range(len(df_est_preview.columns))]
df_limpo_est_p = pd.merge(df_limpo,df_est_preview,how='left',on=['username'])

df_est_comment = df_limpo.groupby('username').agg({'edge_media_to_comment':
['mean','median','std','min','max','count']}).reset_index()
df_est_comment.columns = [df_est_comment.columns[i][0] if i == 0
else df_est_comment.columns[i][1]+'_comment' for i in
range(len(df_est_comment.columns))]
df_limpo_est = pd.merge(df_limpo_est_p,df_est_comment,how='left',on=['username'])
```

A.3 Pré-processamento

```
# Processamento de linguagem Natural

caption1 = df[['accessibility_caption']].copy()
caption1['access_cap'] = caption1['accessibility_caption']
.str.replace('Image may contain: ', '')
caption1[['numero', 'descricao']] = caption1['access_cap']
.str.extract('(\d+)\s*([a-zA-Z\s]+)')

caption1['access_cap_clean'] = caption1['access_cap']
.str.replace('[^\w\s]', '').str.replace('\d', '')

desconsiderar_desc =list(caption1['descricao'].value_counts().index[2:])

caption1.loc[caption1['descricao'].isin(desconsiderar_desc),'numero'] = 0
```

```
caption1 = caption1.drop(columns=['descricao'])

caption1['people'] = caption1['people'].fillna(0)

sw = stopwords.words('english')

sw = stopwords.words('english')
np.array(sw)
caption1['caption'] = ''
translator = str.maketrans('', '', string.punctuation)
stemmer= PorterStemmer()

for i in range( 0, caption1['access_cap_clean'].size ):
    if( (i+1)%1000 == 0 ):
        print(str(round(i/caption1['access_cap_clean'].size*100))+"% done")
    stem_result = []
    input_str = str(caption1['access_cap_clean'][i]).lower()
    #remove emojis
    #input_str = emoji_pattern.sub(r'', input_str)
    #remove hashtags
    #input_str = re.sub(r'#[A-Za-z0-9]+','',str(input_str))
    #remove mençoes
    #input_str = re.sub(r'@[A-Za-z0-9]+','',str(input_str))
    # remove pontuação
    input_str = input_str.translate(translator)
    tokens = word_tokenize(input_str)
    result = [i for i in tokens if not i in sw] # remove stop words
    for word in result:
        stem_result.append(stemmer.stem(word))
    str1 = ' '.join(stem_result)
    caption1['caption'][i] = input_str

def stopwords(row):
```

```
    words = []
    text = [word.lower() for word in str(row['caption']).split() if word.lower()
    not in sw]
    words.extend(text)
    words = sorted(list(set(words)))
    return words
caption1['token'] = caption1.apply(stopwords, axis=1)

count_vectorizer = CountVectorizer(analyzer = "word", tokenizer = None,
preprocessor = None, stop_words = None, max_features = 100)

sentences = []
for i in range( 0, caption1['caption'].size ):
    if( (i+1)%1000 == 0 ):
        print(str(round(i/30000*100))+"% done")
    sentences.append(caption1['caption'][i])

word_features = count_vectorizer.fit_transform(sentences)

word_features = word_features.toarray()

vocab = count_vectorizer.get_feature_names_out()
word_features = pd.DataFrame(data=word_features, columns=vocab)
len(word_features)

caption1['sentiment'] = ''
for i in range( 0, caption1['caption'].size):
    if( (i+1)%1000 == 0 ):
        print(str(round(i/caption1['caption'].size*100))+"% done")
    analysis = TextBlob(caption1['caption'][i])
    caption1['sentiment'][i] = analysis.sentiment[0]

df_bagof = word_features
```

```
df_nlp_image0 = caption1[['people','sentiment']]

# mesmo esquema para as demais variáveis textuais

# Engenharia de atributos

df_final['indica1'] = (df_final['mean_preview']/df_final['median_preview'])
df_final['indica2'] = (df_final['edge_media_preview']-df_final['mean_preview'])
df_final['indica3'] = (df_final['edge_media_to_comment']-df_final['mean_comment'])
/df_final['mean_comment']
df_final['indica4'] = (df_final['edge_media_preview']-df_final['median_preview'])
df_final['indica5'] = (df_final['edge_media_preview']
-(df_final['edge_media_to_comment']))/df_final['std_preview']
df_final['indica6'] = (df_final['mean_preview']-df_final['mean_comment'])
/df_final['std_preview']
df_final['indica7'] = df_final['edge_media_to_comment']/df_final['mean_comment']

# Normalização

df_processo = df_final.copy()
# padronizar para não ter impacto de escala das variáveis
cs = MinMaxScaler()
Data_Continuous = cs.fit_transform(df_processo[v_continuas])

Data_Continuous_df = pd.DataFrame(Data_Continuous, columns=v_continuas)
v_resposta = df_processo['number_of_likes_mean']

df_ts = df_processo.drop(columns=v_continuas+['number_of_likes_mean'])
df_ts = pd.concat([df_ts, Data_Continuous_df], axis = 1)
df_ts_fim = pd.concat([df_ts, v_resposta], axis = 1)
```

A.4 Modelagem

```
df_sv0 = df_ts_fim#.drop(columns=['indica1','indica2','indica3','indica4','indica5',
'indica6','indica7'])
X = df_sv0.drop(columns=['number_of_likes_mean'])
y = df_sv0['number_of_likes_mean']
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0,
test_size=0.20)

# otimização de parâmetros LightGBM
def treinar_modelo(params):
  learning_rate = params[0]
  num_leaves = params[1]
  min_child_samples = params[2]
  subsample = params[3]
  colsample_bytree = params[4]

  #print(params, '\n')
  regressor_lgbm = lgb.LGBMRegressor(learning_rate=learning_rate,
  num_leaves=num_leaves, min_child_samples=min_child_samples,
  subsample=subsample,colsample_bytree=colsample_bytree,random_state=0)
  regressor_lgbm.fit(X_train,y_train,eval_metric='mean_absolute_error')
  predicoes = regressor_lgbm.predict(X_test)

  mae = mean_absolute_error(y_test, predicoes)
  mse = mean_squared_error(y_test, predicoes)
  rmse = math.sqrt(mse)

  return mae

  space = [(1e-3,1e-1),
         (2,128),
         (1,100),
```

```
        (0.05,1.0),
        (0.1,1.0)]

resultado_light = dummy_minimize(treinar_modelo,space,random_state=1,
verbose=1,n_calls=50)

## métricas de teste
mse = mean_squared_error(y_test, predicoes)
rmse = math.sqrt(mse)
display('MAE: %.5f' % mean_absolute_error(y_test, predicoes))
display('MSE: %.5f' % mse)
display('RMSE: %.5f' % rmse)
display('R2 Score: %.5f' % r2_score(y_test, predicoes))
print(regressor_lgbm.get_params())

# intervalo de confiança
y_real = y_test
y_pred = predicoes

def calcular_intervalo_confianca(metrica, y_real, y_pred, confianca=0.95):
    # Calcular a métrica
    erro = metrica(y_real, y_pred)

    # Tamanho do conjunto de teste
    m = len(y_real)

    # Calcular os pesos Wk
    wk = np.square(y_real - y_pred)

    # Calcular a média dos pesos
    media_w = np.mean(wk)

    # Calcular a estimativa da variância dos pesos
```

```
    variancia_w = np.mean(np.square(wk - media_w))

    # Calcular a estimativa do desvio padrão
    desvio_padrao = np.sqrt(variancia_w / m)

    # Calcular o intervalo de confiança
    margem_erro = 2 * np.sqrt(1 / m * variancia_w)
    inferior = erro - margem_erro
    superior = erro + margem_erro

    return inferior, superior

# Métrica: Mean Absolute Error (MAE)
inferior, superior = calcular_intervalo_confianca(mean_absolute_error, y_real,
y_pred)
print(f"Intervalo de confiança para MAE: [{inferior}, {superior}]")

# Métrica: Mean Squared Error (MSE)
inferior, superior = calcular_intervalo_confianca(mean_squared_error, y_real,
y_pred)
print(f"Intervalo de confiança para MSE: [{inferior}, {superior}]")

# Métrica: Root Mean Squared Error (RMSE)
inferior, superior = calcular_intervalo_confianca(lambda y_true, y_pred:
np.sqrt(mean_squared_error(y_true, y_pred)), y_real, y_pred)
print(f"Intervalo de confiança para RMSE: [{inferior}, {superior}]")

# Métrica: R^2 (Coefficient of Determination)
inferior, superior = calcular_intervalo_confianca(r2_score, y_real, y_pred)
print(f"Intervalo de confiança para R^2: [{inferior}, {superior}]")


## representação gráfica:
```

```
sns.set_style('white')
ax=sns.scatterplot(x=y_test, y=predicoes,edgecolor='black',color='lightsteelblue')
z = np.polyfit(y_test, predicoes, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test),"r--")
ax.set_title("Valores previstos vs Valores verdadeiros",fontdict={'size': 11,
'weight': 'bold'})
ax.set_xlabel("Número real de curtidas/média-curtidas")
ax.set_ylabel("Número previsto de curtidas/média-curtidas")
plt.savefig('ajuste_inicial_lightgbm.png', format='png',dpi=300);

# Seleção de Variáveis

importancias = pd.DataFrame({'feature': X_train.columns, 'importance':
regressor_lgbm.feature_importances_})

# Classificar as variáveis por importância
importancias = importancias.sort_values(by='importance', ascending=False)

# Selecionar as top N variáveis (por exemplo, as 10 mais importantes)
top_n = 50
variaveis_selecionadas = importancias['feature'][:top_n].tolist()

# otimização de parâmetros PMC

space = [Categorical(['sgd', 'adam'],name='solver'),
         Real(0.0001,0.05,name='alpha'),
         Categorical(['constant','adaptive'],name='learning_rate'),
         Integer(1,4,name='n_hidden_layer'),
         Integer(100,500,name='n_neurons_per_layer')]

@use_named_args(space)
def objective(**params):
```

```
    n_neurons=params['n_neurons_per_layer']
    n_layers=params['n_hidden_layer']

     n_layers and n_neurons per layer
    params['hidden_layer_sizes']=(n_neurons,)*n_layers

    params.pop('n_neurons_per_layer')
    params.pop('n_hidden_layer')
    print(params)
    rna = MLPRegressor(max_iter=2000,learning_rate_init=0.01,
    activation='relu',random_state=0)
    rna.set_params(**params)
    rna.fit(X_train,y_train)
    predicoes = rna.predict(X_test)

    mae = mean_absolute_error(y_test, predicoes)
    mse = mean_squared_error(y_test, predicoes)
    rmse = math.sqrt(mse)

    return mae

rna = MLPRegressor(hidden_layer_sizes=(359,), # número de neuronios
                   max_iter=1971, # número máximo de iterações/epócas
                   tol=0.00001,  # tolerancia: para o algoritmo convergir
                   learning_rate_init=0.001, #taxa de aprendizado inicial
                   solver="adam",
                   activation='tanh', # função de ativação
                   learning_rate='constant', # taxa constante
                   alpha=0.006679,
                   batch_size=86,
                   verbose=0) # ver o que está acontecendo em cada época
rna.fit(X_train,y_train)
```

```
print(-cross_val_score(rna,X_train,y_train,scoring='neg_mean_absolute_error')
.mean())
print(-cross_val_score(rna,X_train,y_train,scoring='neg_mean_squared_error')
.mean())
print(-cross_val_score(rna,X_train,y_train,scoring='neg_root_mean_squared_error')
.mean())
print(cross_val_score(rna,X_train,y_train,scoring='r2')
.mean())
print('\n')

rna_previsao = rna.predict(X_test)

mse = mean_squared_error(y_test, rna_previsao)
rmse = math.sqrt(mse)
display('MAE: %.5f' % mean_absolute_error(y_test, rna_previsao))
display('MSE: %.5f' % mse)
display('RMSE: %.5f' % rmse)
display('R2 Score: %.5f' % r2_score(y_test, rna_previsao))

# intervalo de confiança
y_real = y_test
y_pred = rna_previsao

def calcular_intervalo_confianca(metrica, y_real, y_pred, confianca=0.95):
    # Calcular a métrica
    erro = metrica(y_real, y_pred)

    # Tamanho do conjunto de teste
    m = len(y_real)

    # Calcular os pesos Wk
    wk = np.square(y_real - y_pred)
```

```
    # Calcular a média dos pesos
    media_w = np.mean(wk)

    # Calcular a estimativa da variância dos pesos
    variancia_w = np.mean(np.square(wk - media_w))

    # Calcular a estimativa do desvio padrão
    desvio_padrao = np.sqrt(variancia_w / m)

    # Calcular o intervalo de confiança
    margem_erro = 2 * np.sqrt(1 / m * variancia_w)
    inferior = erro - margem_erro
    superior = erro + margem_erro

    return inferior, superior


# Métrica: Mean Absolute Error (MAE)
inferior, superior = calcular_intervalo_confianca(mean_absolute_error, y_real,
y_pred)
print(f"Intervalo de confiança para MAE: [{inferior}, {superior}]")

# Métrica: Mean Squared Error (MSE)
inferior, superior = calcular_intervalo_confianca(mean_squared_error, y_real,
y_pred)
print(f"Intervalo de confiança para MSE: [{inferior}, {superior}]")

# Métrica: Root Mean Squared Error (RMSE)
inferior, superior = calcular_intervalo_confianca(lambda y_true, y_pred:
np.sqrt(mean_squared_error(y_true, y_pred)), y_real, y_pred)
print(f"Intervalo de confiança para RMSE: [{inferior}, {superior}]")
```

```
# Métrica: R^2 (Coefficient of Determination)
inferior, superior = calcular_intervalo_confianca(r2_score, y_real, y_pred)
print(f"Intervalo de confiança para R^2: [{inferior}, {superior}]")

sns.set_style('white')
ax=sns.scatterplot(x=y_test, y=rna_previsao,edgecolor='black',
color='lightsteelblue')
z = np.polyfit(y_test, rna_previsao, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test),"r--")
ax.set_title("Valores previstos vs Valores verdadeiros",fontdict={'size': 11,
'weight': 'bold'})
ax.set_xlabel("Número real de curtidas/média-curtidas")
ax.set_ylabel("Número previsto de curtidas/média-curtidas")
```