Análise de sentimentos em avaliações de clientes do *e-commerce* nacional e comparação de métodos tradicionais de *machine learning* com Redes Neurais *Long Short Term Memory* (LSTM)

Carlos Magno Santos Ribeiro de Brito

Trabalho de Conclussão de Curso
Apresentado ao
Instituto de Matemática e Estatística
da
Universidade Federal da Bahia
para
obtenção do título
de
Especialista em Ciência de Dados
e Big Data

Orientador: Prof. Dr. Eduardo Furtado de Simas Filho

Durante o desenvolvimento desta tese e curso o autor do presente trabalho foi contemplado com uma bolsa para realização da especialização

Salvador, 27 de março de 2023

Análise de sentimentos em avaliações de clientes do *e-commerce* nacional e comparação de métodos tradicionais de *machine learning* com Redes Neurais *Long Short Term Memory* (LSTM)

Carlos Magno Santos Ribeiro de Brito[1]
Eduardo Furtado de Simas Filho[2]

Resumo

Com o advento do coronavirus no mundo, o *e-commerce* e todo o comércio *on-line* no Brasil entre 2020 e 2021 se expandiu e se tornou ainda mais competitivo. Nesse contexto, identificar as necessidades dos consumidores e prever tendências de expansão de vendas é uma necessidade dos empreendedores no mercado atual para se tornar mais competitivo. Tendo como base isso, o presente trabalho objetivou analisar e comparar dados de diferentes *E-commerces* no Brasil com diferentes tipos de modelos conhecidos da ciência de dados, na intenção de que os negócios possam ter bons *tradeoffs* de seus modelos em aplicações e direcionar suas vendas de acordo com as preferências dos consumidores em diferentes nichos de mercados. Como resultado, pode-se verificar as principais diferenças entre os métodos de aprendizado de máquina e rede neural e mensurar onde cada qual poderia ser melhor e mais eficientemente utilizado dentro de um negócio.

Palavras chaves: E-commerce; Machine Learning; Redes Neurais Artificiais.

1 Introdução

O *E-commerce*, ou comércio eletrônico, pode ser definido como uma modalidade de negócio onde todo o processo de compra é feito exclusivamente online, isto é, em aplicativos móveis e computadores é realizado desde a escolha do produto ao pagamento. O comércio eletrônico como conhecemos hoje iniciou no final da década de 1960, mas desde 1993, novas tecnologias permitem as empresas realizar funções de negócios eletrônicos (*e-business*) com maior eficiência, rapidez e menores custos. Dados mais recentes da Associação Brasileira de Comércio Eletrônico (ABComm) estimam que, em 2020, 20,2 milhões de consumidores realizaram uma compra on-line pela primeira vez e 150 mil lojas começaram a vender por meio das plataformas digitais. Foram mais de 342 milhões de negociações feitas pela internet, com um valor médio de R$ 310,00 e um volume financeiro estimado de R$ 106 bilhões (Abcomm, 2020).

Até pouco tempo atrás as lojas virtuais eram utilizadas como plataforma de ampliação de vendas de lojas físicas já renomadas, à exemplo de Casas Bahia, Magazine Luiza, Lojas Americanas e etc., mas com a crescente do *e-commerce* no Brasil nos anos de 2020 e 2021, proveniente da crise do coronavírus, observa-se que as empresas estão apostando em lojas virtuais para a exposição e vendas dos seus produtos, dispensando, muitas vezes, o comercio em lojas físicas.

[1]Universidade Federal da Bahia, *carlos.ribeiro.brito@hotmail.com*
[2]Departamento de Engenharia Elétrica e de Computação, *eduardo.simas@ufba.br*

Por se tratar de um mercado de acesso nacional, com maior eficiência, rapidez e menores custos, a ocorrência das vendas on-line é elevada. Logo, para uma empresa se manter competitiva e sólida neste mercado é preciso planejamento, inovação e, principalmente, entender sobre as necessidades dos clientes e como fidelizá-los. De acordo com efagundes (2021) um empreendimento de sucesso é aquele que consegue utilizar a tecnologia existente, adequada aos consumidores do seu nicho de mercado. Por isso mesmo, é fundamental conhecer o comportamento dos consumidores e as tendencias do negócio.

No lado de quem consome, pode-se citar como vantagens do e-commerce: comodidade para os clientes, acesso a opinião de outros usuários/compradores, funcionamento em tempo integral, diversidade de produtos, variedade de opções de pagamentos, privacidade ao usuário/comprador, cupons de desconto e etc., e são justamente estas vantagens e diversidades de informações de dados na compra e venda de produtos que permitem identificar os nichos de mercados de cada seguimento.

Ainda no que concerne o consumidor, após a efetivação da compra, é solicitado uma avaliação do item. Ao se avaliar com pouca precisão, sem muito critério, ou sem muito zelo, algumas situações podem ser encotradas. A primeira delas é a condição não pouco incomum de uma nota desalinhada com o comentário avaliativo, por exemplo: é possível encontrar notas objetivas elevadas (escala 1 a 5, uma nota 5), porém com comentários que demonstram insatisfação com o produto. A segunda situação diz respeito às avaliações intermediárias, no que caracteriza o que é conhecido como *j-shape rating*. Os usuários plenamente satisfeitos e/ou insatisfeitos tendem a avaliar o produto corretamente, todavia, o restante deles muitas vezes acabam avaliando sem muita convicção e de forma errônea.

Destarte, vista as condições de imprecisão ao se obter essas métricas em específico, com este trabalho objetivou analisar dados de diferentes *e-commerces* no Brasil, mais especificamente as notas e comentários avaliativos dos produtos com intuito de obter uma classificação pertinente da satisfação do cliente e também um *tradeoff* entre assertividade e custo de processamento do modelo. Nesse processo foram concebidos modelos com cinco métodos tradicionais do aprendizado de máquina utilizados, sendo eles a Regressão logística, *Naive Bayes*, *XGBoost*, Florestas Aleatórias, *LightGBM* e, para fins comparativos analisou-se também um modelo construído com rede neural artificial *long short term memory* (LSTM).

2 Materiais e métodos

2.1 Tipo de pesquisa

Para realização deste trabalho, realizou-se uma pesquisa quantitativa descritiva, uma vez que buscou-se analisar dados de forma minuciosa a partir de Databases seccionados e coletados de diversas lojas virtuais no Brasil. Isto é, a pesquisa descritiva compreende a observação, ao registro, a análise, a classificação, a interpretação e a padronização de dados com a finalidade de explicar a correspondência entre as variáveis investigadas.

Em relação ao tratamento do problema, a pesquisa se caracterizou como quantitativa. Segundo Malhotra (2006), o objetivo da pesquisa quantitativa é quantificar os dados e aplicar ferramentas de análises estatísticas. O trabalho utilizou da tratativa, analise, correlação e apresentação de dados, além de aplicações de métodos estatísticos para criação de diferentes cenários, os quais têm como objetivo identificar possíveis tendência entre os dados utilizados e a satisfação do cliente na tomada de decisão em empresas que atuam nesse setor.

Quanto à coleta de dados, este trabalho foi classificado como uma pesquisa documental. A pesquisa documental recorre a fontes mais diversificadas e dispersas, sem tratamento analítico, tais como: tabelas estatísticas, jornais, revistas, relatórios, documentos oficiais,

cartas, filmes, fotografias, pinturas, tapeçarias, relatórios de empresas, vídeos de programas de televisão, etc. (da Fonseca, 2002). A pesquisa documental é um tipo de pesquisa que é baseada dados e informações que não passaram por um processo analítico e/ou sintético. Os dados deste trabalho, de diferentes E-commerces no Brasil, foram disponibilizados pela Olist e estão disponíveis na plataforma *Kaggle*, com o nome *Brazilian E-Commerce Public Dataset* (Olist and Sionek, 2018).

2.2 Fonte e descrição dos dados

Olist é uma startup brasileira que atua no segmento de E-commerce, sobretudo por meio de *marketplace*. De um lado, a Olist concentra vendedores que desejam anunciar em *marketplaces* como Mercado Livre, B2W, Via Varejo, Amazon e etc. Por outro, concentra os produtos de todos os vendedores em uma loja única que fica visível ao consumidor final. Atualmente o negócio reúne mais de 800 colaboradores e mais de 9 mil lojistas, além de 2 milhões de consumidores únicos (Olist, date).

A base de dados escolhida é um database público que consiste em um conjunto de dados que descrevem a rotina de compra de um E-commerce. É possível visualizar diversas características de um produto, como o nome, preço, descrição, nota atribuída, comentários, local de compra, etc. Mais especificamente, para este trabalho, analisou-se os comentários avaliativos (*review_comment_message*) e as notas dadas pelos clientes (*review_score*). O tamanho total do dataset é de aproximadamente 126 *megabytes* e da tabela que *olist_order_review* com 14,45 *megabytes* e aproximadamente 100 mil linhas.

2.3 Ferramentas

Abaixo na Figura 1, tem-se o fluxograma do processo de realização deste trabalho. inicialmente a coleta de dados foi iniciada pelo Olist e assim disponibilizada a público. Em seguida, a análise e a sua validação é feita em trabalho, para verificar se os dados faziam sentido e se tinham a consistência informada. A análise dos resultados se deu após as análises do algoritmo produzido em *python* juntamente com gráficos. As conclusões são feitas no presente trabalho, levando em conta o que foi proposto em ser analisado.

Além disso, as principais bibliotecas utilizadas no trabalho foram:

1. Processamento e análise de dados:
    - Pandas: para manipulação e análise de dados
    - NumPy: para computação numérica
    - Matplotlib: para visualização de dados
    - Seaborn: para visualização estatística de dados
2. Aprendizado de máquina:
    - Scikit-learn: para algoritmos de aprendizado de máquina e pré-processamento de dados
    - TensorFlow: para construção e treinamento de modelos de aprendizado profundo
    - XGBoost: para algoritmos de boosting de gradiente
    - LightGBM: para algoritmos de boosting de gradiente
3. Processamento de Linguagem Natural:

- NLTK: para tarefas de processamento de linguagem natural, como tokenização, stemming e etiquetagem
- SpaCy: para tarefas avançadas de processamento de linguagem natural, como reconhecimento de entidades e análise de dependência

4. Diversos:

- WordCloud: para geração de nuvens de palavras
- PIL: para processamento e manipulação de imagens

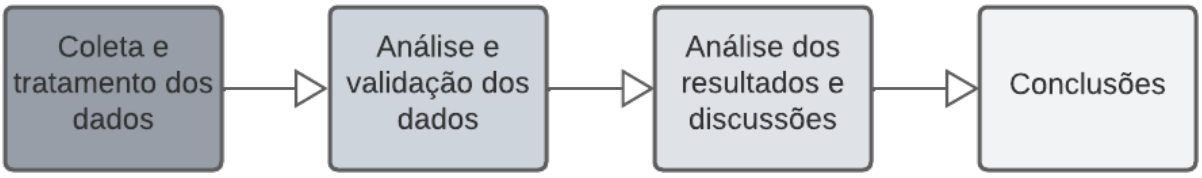

Figura 1: Fluxograma do trabalho

O esquema dos *databases* disponibilizados pelo Olist é representado na Figura 3. Nele, utilizou-se apenas a tabela *olist_order_review*, com os campos disponíveis na Tabela 1. Além dele, o exemplo de como o cliente avalia um produto também é apresentado na Figura 2.

	review_id	order_id	review_comment_title	review_comment_message	...
type	hash	hash	string—null	string—null	
ex	da79b0a377eb	df73dbeba33	bom, mas	atende às expectativas	

	review_score	review_creation_date	review_answer_timestamp
type	number	datestring	datestring
ex	3	2018-01-18 00:00:00	2018-01-18 21:00:00

Tabela 1: Esquema da tabela *olist_order_review*

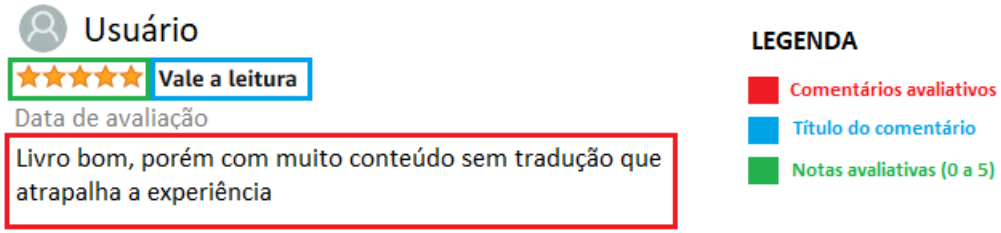

Figura 2: Forma de avaliação de um consumidor

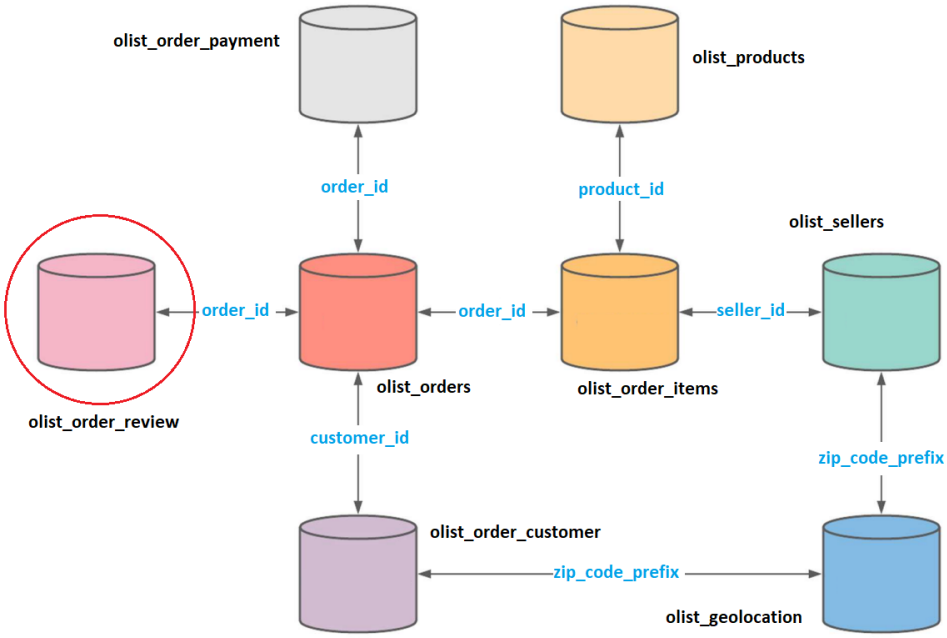

Figura 3: Esquemas do dataset publicado pelo Olist, adaptado pelo autor (2022)

Na primeira fase, utilizando a linguagem de programação Python, os databases em formato CSV foram extraídos e exportados para leitura e armazenamento em diferentes variáveis no intuito de limpar e adequar os dados para análises. Nesta etapa, itens duplicados, com valores nulos e valores discrepantes foram removidos. Além disso, separou-se dois *datasets* contendo informações dos comentários dos clientes (*review_comment_message*) e da sua nota de avaliação (*review_score*). Para o *data_bin* tem-se comentários associados categoricamente com valores binários (0 e 1) a partir das notas, com o valor atribuído de 0 para o intervalo (0,2] e de 1 para o intervalo de (2, 5]. No *data_cat* tem-se apenas os comentários e os reviews numéricos de 1 a 5. Ambas as bases são removidos os valores nulos. Tem-se a Tabela 2 e Tabela 3 exemplificando os casos citados.

	index	text	label
type	bigint	string	boolean
exemplo	1	produto muito ruim	0

Tabela 2: Esquema de geração do *data_bin*

	index	text	label
type	bigint	string	number
exemplo	2	produto muito bom	4

Tabela 3: Esquema de geração do *data_gen*

2.3.1 Máquina para processamento

Todos os métodos foram executados em fila, sequencialmente entre eles, onde cada um deles foi executado em ordem, um de cada vez e de maneira procedural.

A máquina utilizada para todos eles foi o laptop Dell G3 com processador Intel Core i7 de 10ª geração, 16GB de RAM, 512GB de SSD e placa de vídeo NVIDIA RTX 2060 com 6GB

de memória dedicada, incluindo o sistema operacional Windows 11 com WSL 2 instalado e Ubuntu 20.04 como ambiente de execução para as ferramentas de teste de software usadas.

2.4 Métodos

A partir das limpeza dos dados, para análise de ambos datasets gerados, lançou-se mão de métodos clássicos do *Machine learning* sendo eles: Regressão logística, Naive Bayes, Florestas Aleatórias e *XGBoost*. Além deles, para comparação, utilizou-se uma rede neural artificial LSTM. Esses métodos serão aqui melhor detalhados.

2.4.1 Regressão Logística

A regressão logística é um modelo estatístico robusto e eficiente, que permite a previsão da probabilidade de um evento binário de forma precisa e confiável (Hosmer Jr *et al.*, 2013). Este tipo de evento é aquele que pode ocorrer ou não, ou que pode ser classificado em duas categorias distintas. Por exemplo, em uma análise de crédito, o cliente pode ser aprovado ou não. Na medicina, o evento pode ser a cura ou não de uma doença.

Ela apresenta um modelo linear generalizado que utiliza a função logística para modelar a relação entre as variáveis independentes e a variável dependente (Kleinbaum and Klein, 2010). A função logística é uma função sigmoide que transforma uma variável linear em uma probabilidade. A equação da função logística é dada por:

$$p = \frac{1}{1 + e^{-x}} \tag{1}$$

Sendo p a probabilidade do evento ocorrer, x uma variável linear que representa a combinação linear das variáveis independentes, e e a constante de Euler.

A regressão logística utiliza a técnica de máxima verossimilhança para estimar os parâmetros do modelo a partir dos dados observados. A função de verossimilhança é maximizada para encontrar os valores dos coeficientes que melhor se ajustam aos dados. O modelo é ajustado para minimizar a diferença entre as probabilidades previstas pelo modelo e as probabilidades observadas nos dados (McCullagh and Nelder, 1989).

A regressão logística tem sido amplamente utilizada em diversas áreas, como na análise de dados de sobrevivência, na análise de dados de saúde, na análise de dados financeiros, etc. Por exemplo, é utilizada na análise de dados de sobrevivência para modelar a probabilidade de um paciente sobreviver a uma doença com base em fatores como idade, sexo, nível de educação, etc. Na análise de dados financeiros, é utilizada para modelar a probabilidade de um cliente pagar ou não uma dívida com base em fatores como histórico de crédito, renda, etc.

Na Figura 8 tem-se a curva sigmoide que é uma representação visual de como a probabilidade de uma variável dependente binária (como "sim"ou "não") muda em relação a uma variável independente contínua. Ela é uma função em forma de "S"que começa com uma probabilidade baixa para valores muito baixos de x, aumenta rapidamente à medida que x aumenta e se estabiliza em uma probabilidade alta para valores muito altos de x. A inclinação da curva sigmoide no ponto médio (onde $p = 0,5$) é o valor da inclinação da reta da regressão logística.

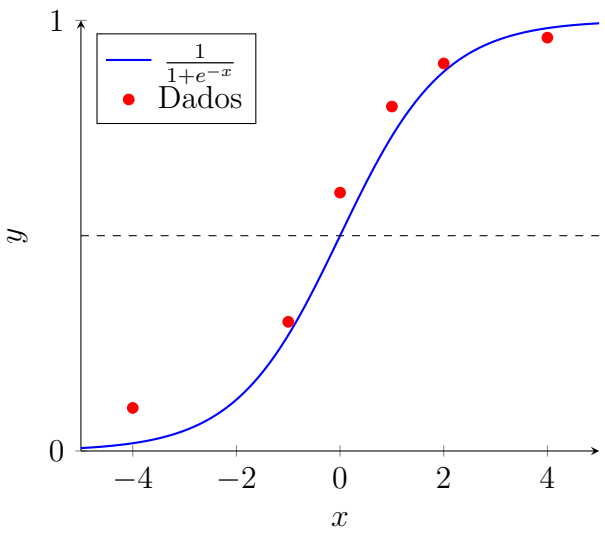

Figura 4: Modelo gráfico da regressão logística (curva sigmoide)

2.4.2 Naive Bayes

A classificação Naive Bayes é um algoritmo de aprendizado de máquina supervisionado que utiliza o teorema de Bayes para classificar instâncias em classes discretas. A principal vantagem do algoritmo Naive Bayes é a sua simplicidade e eficiência computacional, o que o torna uma escolha popular para problemas de classificação em grande escala (Mitchell, 1997).

O algoritmo Naive Bayes é baseado no teorema de Bayes, que fornece uma maneira de calcular a probabilidade condicional de uma hipótese, dado um conjunto de evidências. Na classificação, a hipótese corresponde à classe da instância e as evidências correspondem às características observadas da instância. O algoritmo calcula a probabilidade condicional de cada classe, dado as características observadas da instância, e atribui a instância à classe com a maior probabilidade condicional.

Segundo Domingos and Pazzani (1997), a principal suposição por trás do algoritmo Naive Bayes é a independência condicional das características, ou seja, cada característica contribui independentemente para a probabilidade condicional de cada classe. Embora essa suposição seja muitas vezes violada na prática, o algoritmo Naive Bayes ainda funciona bem em muitos casos e pode ser especialmente útil quando há muitas características.

O algoritmo Naive Bayes tem sido aplicado em muitas áreas, incluindo reconhecimento de fala, processamento de texto e detecção de spam de e-mail. É particularmente útil em aplicações onde há muitas características e as instâncias são discretas ou categóricas. Um estudo empírico sobre o desempenho do algoritmo Naive Bayes em diferentes conjuntos de dados pode ser encontrado em (Rish, 2001).

Embora o algoritmo Naive Bayes seja uma técnica de classificação simples e eficiente, ele também tem algumas limitações. Por exemplo, a suposição de independência condicional das características pode ser inadequada em algumas situações, e a performance do algoritmo pode ser afetada por dados ausentes ou dados ruidosos (Mitchell, 1997).

As equações abaixo representam a base matemática do algoritmo de classificação Naive Bayes:

Dada uma instância $X = x_1, x_2, \ldots, x_n$ com características observadas, o objetivo do algoritmo é determinar a probabilidade condicional $P(C_k|X)$ da instância pertencer a cada classe C_k.

O teorema de Bayes fornece uma maneira de calcular a probabilidade condicional $P(C_k|X)$ em termos das probabilidades a priori $P(C_k)$ e das probabilidades condicionais $P(X|C_k)$, como segue:

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)} \tag{2}$$

Aqui, $P(C_k)$ é a probabilidade a priori da classe C_k e $P(X|C_k)$ é a probabilidade condicional de observar a instância X dada a classe C_k.

O classificador Naive Bayes assume a independência condicional das características dadas a classe, isto é, $P(x_i|C_k, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|C_k)$. Isso permite simplificar a probabilidade condicional $P(X|C_k)$ como:

$$P(X|C_k) = P(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n|C_k) = \prod_{i=1}^{n} P(x_i|C_k) \tag{3}$$

Substituindo essa expressão na equação anterior, tem-se:

$$P(C_k|X) = \frac{\prod_{i=1}^{n} P(x_i|C_k)P(C_k)}{P(X)} \tag{4}$$

Como a probabilidade $P(X)$ é constante para todas as classes, é possível comparar apenas as probabilidades $P(C_k)\prod_{i=1}^{n} P(x_i|C_k)$ para determinar a classe mais provável para a instância X.

O desenho ilustra as variáveis aleatórias x_1, x_2, x_3, x_4 e a classe C_k. As setas que partem das probabilidades condicionais e a priori indicam as fórmulas para o cálculo das probabilidades conjuntas $P(x_1, x_2, x_3, x_4, C_k)$ e, consequentemente, para a classificação de uma instância.

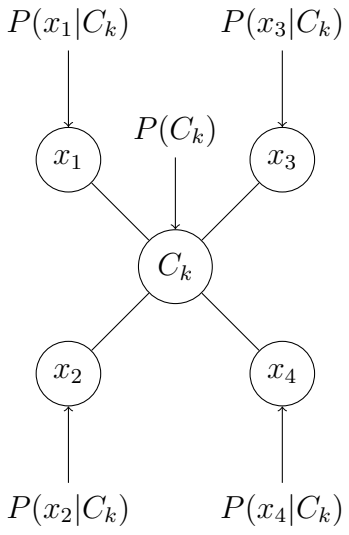

Figura 5: Esquema de funcionamento da classificação Naive Bayes

2.4.3 Árvore de decisão

Antes de discorrer sobre os métodos a seguir de aprendizado de máquina, faz-se necessário deixar elucidado como funcionam as árvores de decisão.

Uma árvore de decisão é um modelo de aprendizado de máquina que usa uma estrutura em forma de árvore para representar uma série de decisões e suas possíveis consequências.

Cada nó interno da árvore representa uma decisão que leva a um ou mais ramos subsequentes, enquanto cada folha representa o resultado final de uma série de decisões.

As árvores de decisão são amplamente utilizadas em áreas como ciência de dados, reconhecimento de padrões, bioinformática e finanças. Elas são populares porque são fáceis de entender e interpretar, permitindo que os usuários examinem os fatores que afetam as decisões e identifiquem os resultados mais prováveis.

Para construir uma árvore de decisão, é necessário um conjunto de dados de treinamento que contenha exemplos de entrada e saída. A árvore é construída a partir da análise dos dados de treinamento para determinar as variáveis mais importantes e suas relações com as saídas.

Existem várias técnicas para a construção de árvores de decisão, incluindo o algoritmo ID3 (*Iterative Dichotomiser 3*), o algoritmo C4.5 e o algoritmo CART (*Classification and Regression Trees*). Cada algoritmo tem suas próprias vantagens e desvantagens, dependendo do problema em questão e dos dados disponíveis.

As árvores de decisão também podem ser usadas em conjunto com outras técnicas de aprendizado de máquina, como o *ensemble learning*, para melhorar a precisão dos modelos e reduzir o risco de *overfitting*.

2.4.4 Florestas aleatórias

De acordo com Breiman (2001), as florestas aleatórias são uma técnica de aprendizado de máquina que combina várias árvores de decisão para construir um modelo de classificação ou regressão. Cada árvore de decisão é construída a partir de um subconjunto aleatório dos dados de treinamento e um subconjunto aleatório dos recursos (também conhecidos como características ou atributos). Esses subconjuntos são criados para garantir que cada árvore de decisão seja diferente e que a floresta aleatória possa capturar várias relações entre os dados e recursos.

A construção de uma árvore de decisão é feita por meio de uma série de etapas. Inicialmente, a árvore começa com um único nó que representa todo o conjunto de dados de treinamento. Em seguida, a árvore é dividida em nós menores usando uma função de divisão que escolhe um recurso e um ponto de divisão que minimiza a impureza dos dados. A impureza é uma medida da desorganização dos dados, que pode ser medida por diferentes critérios, como a entropia ou o índice Gini. O processo de divisão é repetido recursivamente até que os nós finais sejam puros ou um critério de parada seja atingido, como uma profundidade máxima da árvore (Cutler *et al.*, 2001).

Durante a fase de teste, a floresta aleatória retorna a classe mais comum ou a média das saídas das árvores individuais, dependendo se o problema é de classificação ou regressão, respectivamente.

As florestas aleatórias apresentam várias vantagens em relação a outras técnicas de aprendizado de máquina. Em primeiro lugar, elas têm um bom desempenho em dados de alta dimensão, onde o número de recursos é grande em relação ao número de amostras. Em segundo lugar, elas são relativamente insensíveis a outliers e dados ausentes. Em terceiro lugar, elas são facilmente paralelizáveis, permitindo que grandes conjuntos de dados sejam processados em paralelo em clusters de computadores (Cutler *et al.*, 2001).

Em termos de aplicação, elas são amplamente utilizadas em uma variedade de problemas, como reconhecimento de padrões em imagens e sinais, detecção de fraudes em transações financeiras, análise de sentimentos em redes sociais, previsão de preços de ações e análise de dados genômicos (Breiman, 2001).

As equações relacionadas às florestas aleatórias são principalmente as usadas na cons-

trução de cada árvore de decisão. Por exemplo, as equações para o cálculo da impureza dos dados, que é um dos principais critérios de divisão de nós em uma árvore de decisão são dadas:

- O índice Gini:

$$G_i = \sum_{k=1}^{K} p_{i,k}(1 - p_{i,k}), \tag{5}$$

  onde K é o número de classes, $p_{i,k}$ é a proporção de observações da classe k no nó i.

- A entropia:

$$H_i = -\sum_{k=1}^{K} p_{i,k} \log(p_{i,k}), \tag{6}$$

  onde K é o número de classes, $p_{i,k}$ é a proporção de observações da classe k no nó i.

- O critério de divisão de Gini é dado por:

$$G_d = \sum_{i=1}^{q} \frac{n_i}{n} G_i, \tag{7}$$

  onde q é o número de nós filhos resultantes da divisão, n_i é o número de observações no nó i e n é o número total de observações.

- O critério de divisão de entropia é dado por:

$$H_d = -\sum_{i=1}^{q} \frac{n_i}{n} H_i, \tag{8}$$

  onde q é o número de nós filhos resultantes da divisão, n_i é o número de observações no nó i e n é o número total de observações.

Essas equações são usadas para calcular a impureza dos dados em cada nó da árvore de decisão e, assim, decidir qual recurso e ponto de divisão usar para dividir o nó em dois filhos. O processo de divisão é repetido recursivamente para construir a árvore de decisão completa. Em seguida, várias árvores de decisão são combinadas para formar a floresta aleatória.

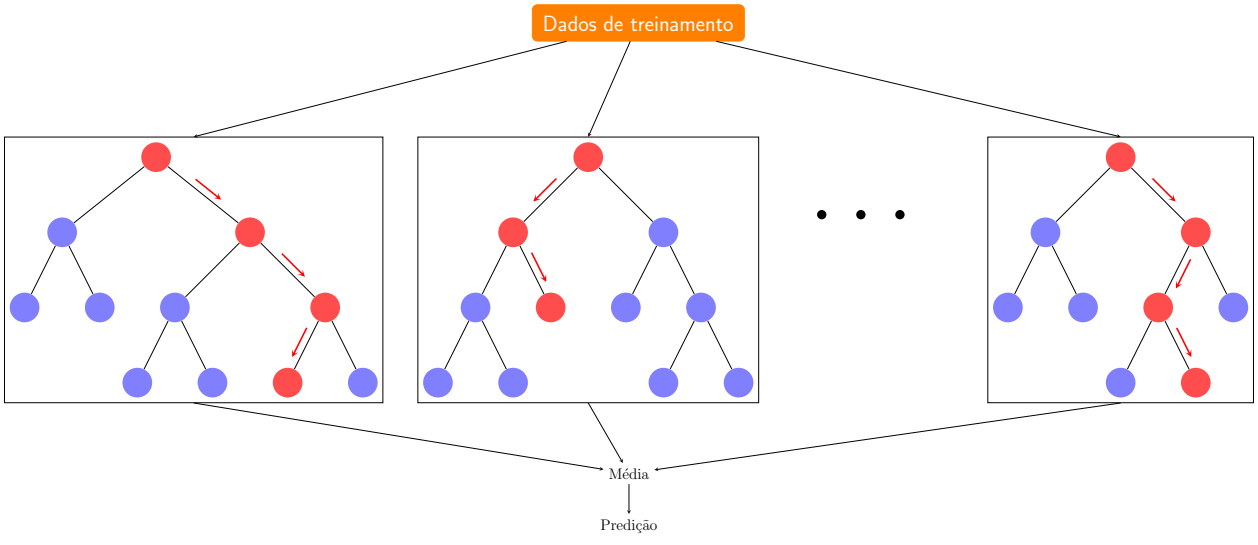

Figura 6: Esquema de funcionamento de uma floresta aleatória

2.4.5 XGBoost

O XGBoost (*Extreme Gradient Boosting*) é um método de aprendizado de máquina baseado em árvores de decisão, assim como as florestas aleatórias, mas com algumas diferenças importantes. Enquanto as florestas aleatórias usam um conjunto de árvores de decisão independentes para fazer uma previsão, o XGBoost usa um conjunto de árvores de decisão sequenciais que são criadas iterativamente. Cada nova árvore é ajustada aos resíduos do modelo anterior, tentando corrigir os erros cometidos pelo modelo atual.

O algoritmo XGBoost foi desenvolvido por Tianqi Chen e Carlos Guestrin em 2016 (Chen and Guestrin, 2016) e é baseado na biblioteca de código aberto de mesmo nome. O XGBoost se tornou um dos algoritmos de aprendizado de máquina mais populares em competições de ciência de dados e é amplamente utilizado na indústria.

O processo de construção do modelo XGBoost pode ser descrito por meio da seguinte equação:

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K}(\mathbf{x}_i) \tag{9}$$

onde $\hat{y}_i$ é a previsão para a i-ésima instância, ϕ é a função de previsão, $\mathbf{x}_i$ é o vetor de características para a i-ésima instância, K é o número total de árvores no modelo e f_k é a k-ésima árvore de decisão.

Para construir o modelo XGBoost, o algoritmo usa um processo iterativo de adição de árvores, onde cada nova árvore é ajustada aos resíduos do modelo anterior, tentando corrigir os erros cometidos pelo modelo atual. Esse processo é descrito pela seguinte equação:

$$\mathbf{F}_k = \mathbf{F}_{k-1} + \gamma_k f_k \tag{10}$$

onde $\mathbf{F}_k$ é a soma cumulativa das previsões das árvores anteriores até a k-ésima árvore, γ_k é a taxa de aprendizado da k-ésima árvore e f_k é a k-ésima árvore de decisão.

O XGBoost também usa um conjunto de hiperparâmetros para ajustar o modelo e evitar overfitting. Alguns dos hiperparâmetros mais importantes incluem:

- Número de árvores: o número total de árvores a serem criadas;
- Profundidade máxima: o número máximo de camadas que cada árvore pode ter;
- Taxa de aprendizado: a taxa na qual o modelo tenta corrigir os erros cometidos pelas árvores anteriores;
- Subamostragem: a fração de amostras de treinamento a serem usadas para treinar cada árvore;
- Colsample: a fração de recursos (características) a serem amostrados aleatoriamente para cada árvore.

Chen e Guestrin descrevem o algoritmo XGBoost em detalhes em seu artigo "XGBoost: A Scalable Tree Boosting System"(Chen and Guestrin, 2016), onde eles mostram que o XGBoost pode superar outros algoritmos de aprendizado de máquina em uma variedade de conjuntos de dados e tarefas.

Uma das principais vantagens do XGBoost é sua eficiência computacional. O algoritmo usa várias técnicas para reduzir o tempo de treinamento e a complexidade do modelo, incluindo a amostragem de características, a paralelização do processo de treinamento e o uso de uma estrutura de dados específica chamada "gradiente e Hessiano".

O XGBoost é amplamente utilizado em competições de ciência de dados e em projetos de aprendizado de máquina na indústria. A biblioteca XGBoost é de código aberto e está disponível para várias linguagens de programação, incluindo Python, R e Java.

As seguintes equações são usadas para calcular os valores dos gradientes e Hessiano para cada instância i:

$$\begin{aligned} g_i &= \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}, \\ h_i &= \frac{\partial^2 L(y_i, \hat{y}_i)}{\partial {\hat{y}_i}^2} \end{aligned} \tag{11}$$

onde L é a função de perda usada para avaliar a qualidade da previsão. O XGBoost usa esses valores de gradiente e Hessiano para ajustar as árvores do modelo, tentando minimizar a função de perda.

2.4.6 LightGBM

LightGBM é um algoritmo de aprendizado de máquina baseado em árvore, desenvolvido pela Microsoft, que utiliza um método de treinamento baseado em gradientes para melhorar a velocidade e a eficiência de modelos baseados em árvore. O algoritmo utiliza uma técnica de amostragem por folha, que pode levar a melhorias significativas no tempo de treinamento e na precisão do modelo.

Segundo Ke *et al.* (2017), o LightGBM apresenta melhorias significativas em relação a outros algoritmos de aprendizado de máquina baseados em árvore, como o XGBoost, tanto em termos de tempo de treinamento quanto de desempenho preditivo. O LightGBM é particularmente eficaz em conjuntos de dados grandes e esparsos, e tem sido amplamente utilizado em aplicações de classificação e regressão.

Uma das principais vantagens do LightGBM é a sua eficiência computacional. Esta técnica de amostragem por folha, em que cada folha da árvore é treinada com um subconjunto aleatório dos dados de treinamento permite que o algoritmo treine árvores mais profundas e complexas sem aumentar significativamente o tempo de treinamento. Além disso, o LightGBM utiliza uma técnica de paralelização de histogramas, que divide os dados em intervalos discretos e constrói histogramas para cada recurso, permitindo que as operações de treinamento sejam executadas em paralelo.

O LightGBM também apresenta melhorias significativas no desempenho preditivo em relação a outros algoritmos de aprendizado de máquina baseados em árvore. Ke *et al.* (2017) mostraram que o LightGBM supera o XGBoost em várias métricas de desempenho, incluindo tempo de treinamento, tempo de previsão e precisão preditiva em vários conjuntos de dados. O LightGBM também é capaz de lidar com conjuntos de dados grandes e esparsos, que são comuns em muitas aplicações do mundo real.

Em relação às suas principais equações definidoras, tem-se:

A) A função objetivo (*loss function*) utilizada pelo LightGBM que é dada por:

$$L(\theta) = \sum_{i=1}^{n} l(y_i, f(\mathbf{x_i}; \theta)) + \sum_{k=1}^{K} \Omega(f_k) \tag{12}$$

Onde l é a função de perda, y_i é o rótulo do exemplo de treinamento i, $f(\mathbf{x_i}; \theta)$ é a predição do modelo para o exemplo $\mathbf{x_i}$, θ é o conjunto de parâmetros do modelo, K é o número de árvores utilizadas pelo modelo, e $\Omega(f_k)$ é a função de regularização utilizada para controlar a complexidade do modelo.

B) O algoritmo utiliza o método de boosting para construir o modelo. Em cada iteração t, o LightGBM adiciona uma nova árvore f_t ao modelo. A predição do modelo após T iterações é dada por:

$$f_T(\mathbf{x}) = \sum_{t=1}^{T} f_t(\mathbf{x}) \tag{13}$$

C) A técnica de gradientes para atualizar os parâmetros do modelo em cada iteração. A atualização dos parâmetros é realizada utilizando o seguinte algoritmo:

1. Computar os gradientes para cada exemplo de treinamento:

$$g_i = \frac{\partial l(y_i, f(\mathbf{x_i}; \theta))}{\partial f(\mathbf{x_i}; \theta)} \tag{14}$$

2. Construir uma nova árvore f_t que minimize a função objetivo:

$$f_t = \arg\min_f \sum_{i=1}^{n} g_i f(\mathbf{x_i}; \theta_{t-1}) + \frac{1}{2} h f^2(\mathbf{x_i}) + \Omega(f) \tag{15}$$

   Onde h é a taxa de aprendizagem (learning rate) utilizada para controlar a magnitude da atualização dos parâmetros.

2.5 Redes Neurais

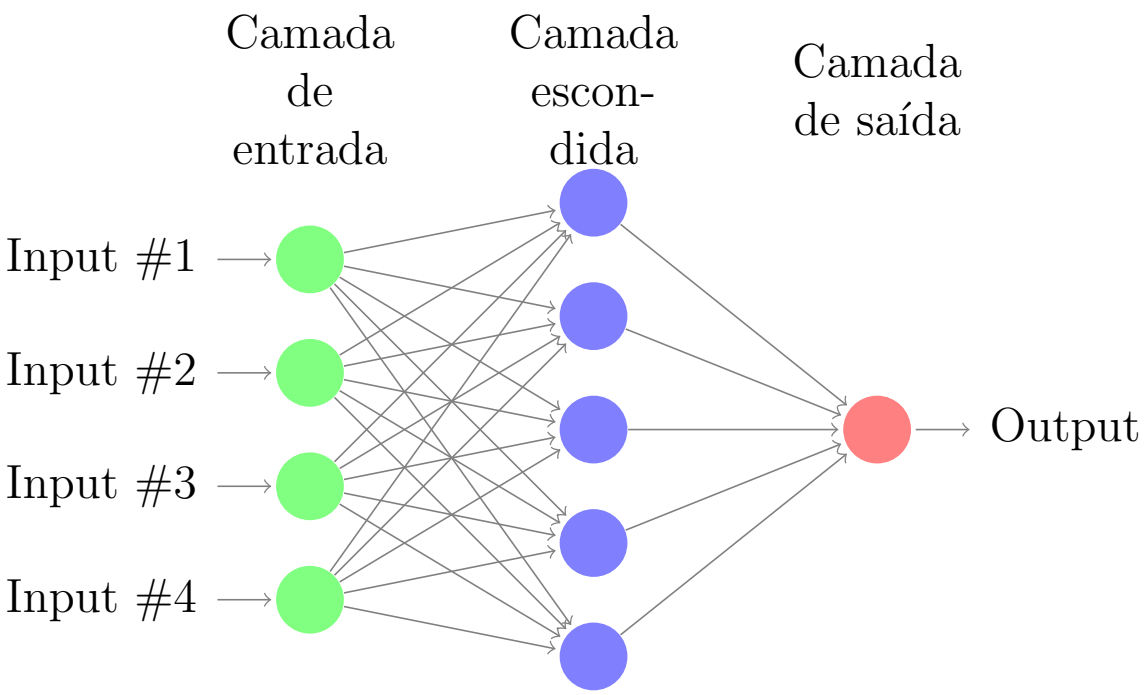

Figura 7: Esquema simples de funcionamento de uma rede neural

Redes Neurais Artificiais (RNAs) são modelos computacionais que se inspiram no funcionamento do cérebro humano e têm sido amplamente utilizadas para tarefas de classificação, previsão e reconhecimento de padrões em diferentes áreas do conhecimento. O conceito de

RNA foi proposto por McCulloch and Pitts (1943), mas foi apenas a partir da década de 1980, com o desenvolvimento de técnicas de treinamento de redes profundas, que as RNAs se tornaram uma ferramenta poderosa para análise de dados LeCun *et al.* (2015).

Uma RNA é composta por camadas de neurônios artificiais, cada um com uma função de ativação que transforma a entrada recebida em uma saída. A primeira camada é a camada de entrada, que recebe os dados a serem processados. A última camada é a camada de saída, que produz a resposta final da RNA. Entre as camadas de entrada e saída, podem ser adicionadas várias camadas ocultas, que ajudam a extrair características dos dados de entrada.

O processo de treinamento de uma RNA consiste em ajustar os pesos sinápticos entre os neurônios para que a rede produza a saída correta para cada entrada. O algoritmo mais comum de treinamento é o Backpropagation, proposto por Rumelhart *et al.* (1986). O Backpropagation utiliza o método do gradiente descendente para minimizar a função de custo da RNA em relação aos pesos sinápticos.

Uma das principais vantagens das RNAs é a capacidade de lidar com dados complexos e não lineares. Além disso, as RNAs podem ser utilizadas em problemas de classificação, previsão e reconhecimento de padrões em diferentes áreas, como visão computacional, processamento de fala, processamento de texto e bioinformática.

No entanto, elas possuem algumas desvantagens, como a dificuldade de interpretação dos resultados e o risco de overfitting, que ocorre quando a rede se ajusta demais aos dados de treinamento e não consegue generalizar para novos dados.

As redes neurais artificiais são compostas por uma série de camadas de neurônios, que realizam operações matemáticas nas entradas recebidas para gerar saídas. A formulação matemática dessas operações pode variar de acordo com a arquitetura da rede, mas em geral envolvem uma combinação linear das entradas, seguida de uma função não linear de ativação.

Abaixo tem-se as equações matemáticas para uma rede neural *feedforward* de três camadas, com $n^{(l)}$ neurônios na camada l:

$$
\begin{aligned}
z_j^{(2)} &= \sum_{i=1}^{n^{(1)}} w_{ij}^{(1)} x_i + b_j^{(1)} \\
h_j^{(2)} &= f(z_j^{(2)}) \\
z_k^{(3)} &= \sum_{j=1}^{n^{(2)}} w_{jk}^{(2)} h_j^{(2)} + b_k^{(2)} \\
y_k &= f(z_k^{(3)})
\end{aligned}
\tag{16}
$$

Na equação acima, x_i representa a entrada na posição i, $w_{ij}^{(1)}$ representa o peso associado à conexão entre o neurônio i da camada de entrada e o neurônio j da camada oculta, $b_j^{(1)}$ é o viés associado ao neurônio j da camada oculta, f é a função de ativação, $h_j^{(2)}$ é a saída do neurônio j da camada oculta, $w_{jk}^{(2)}$ é o peso associado à conexão entre o neurônio j da camada oculta e o neurônio k da camada de saída, $b_k^{(2)}$ é o viés associado ao neurônio k da camada de saída e y_k é a saída final da rede neural.

É importante notar que a escolha da função de ativação pode influenciar significativamente o comportamento da rede neural, permitindo, por exemplo, que ela aprenda a modelar funções não-lineares complexas. Algumas funções de ativação comuns incluem a função sigmoide, a função tangente hiperbólica e a função ReLU (Rectified Linear Unit).

2.5.1 Redes Neurais Artificiais LSTM

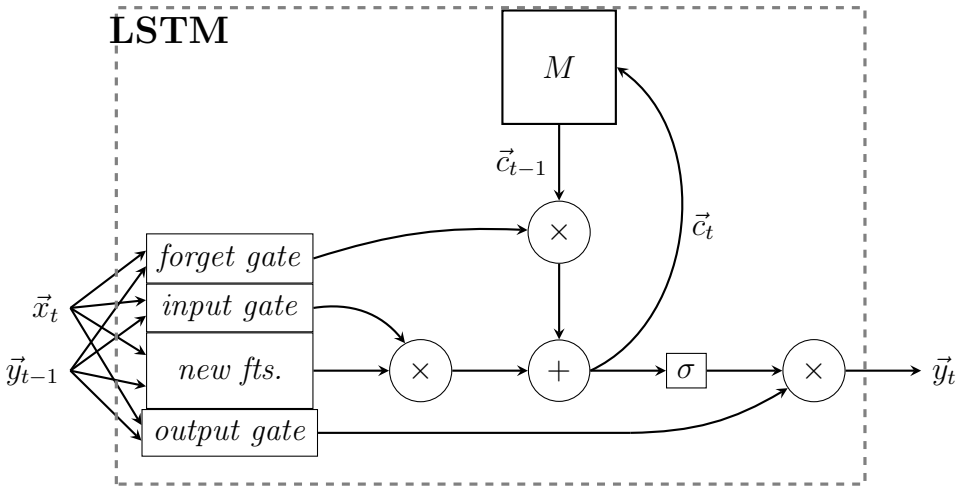

Figura 8: Um diagrama representando um único bloco LSTM, no contexto de redes neurais recorrentes. Fonte: PetarV (2023)

Uma rede neural artificial *Long Short-Term Memory* (LSTM) é um tipo especializado de rede neural recorrente (RNN) projetada para lidar com o problema de dependência a longo prazo em sequências de dados. Foi proposta por Hochreiter and Schmidhuber (1997) como uma solução para o problema de desvanecimento do gradiente em RNNs tradicionais.

O problema de desvanecimento do gradiente ocorre quando o gradiente calculado durante o processo de retropropagação se torna muito pequeno à medida que se propaga pelas camadas da rede, o que dificulta o aprendizado de dependências a longo prazo. Isso limita a capacidade das RNNs tradicionais em lidar com sequências de dados que envolvem dependências a longo prazo, como em análise de séries temporais ou processamento de linguagem natural.

A principal característica da LSTM é a presença de unidades de memória, que são capazes de lembrar informações relevantes por um período prolongado de tempo. Essas unidades são compostas por três portões principais: o portão de entrada, o portão de esquecimento e o portão de saída. O portão de entrada controla a quantidade de informação que é adicionada à unidade de memória, enquanto o portão de esquecimento controla a quantidade de informação que é descartada. O portão de saída controla a quantidade de informação que é enviada para a próxima unidade da rede.

Durante o treinamento, a LSTM é capaz de aprender a modificar seus portões de entrada, esquecimento e saída, a fim de manter as informações relevantes na memória e descartar as informações irrelevantes. Os pesos das conexões entre as unidades de memória são atualizados usando o algoritmo de retropropagação através do tempo (BPTT).

A arquitetura LSTM foi amplamente utilizada em diversas aplicações, incluindo processamento de linguagem natural, reconhecimento de fala e análise de séries temporais. Por exemplo, Graves *et al.* (2013) utilizaram LSTMs para melhorar o reconhecimento de fala em 2013. Além disso, Gers *et al.* (1999) propuseram uma abordagem de aprendizado contínuo com LSTMs, chamada de "learning to forget", que permite que a rede esqueça informações irrelevantes à medida que recebe novos dados.

Para o trabalho específico usou-se o *tensorflow keras* com os seguintes valores:

- unidades de células: 32

- camadas: 2
- épocas: 20
- taxa de aprendizagem: 0.01
- neurônios: 32

Abaixo estão algumas formulações matemáticas para uma unidade LSTM:

$$\begin{aligned} i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ \tilde{c}t &= \tanh(W_c x_t + U_c ht - 1 + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}t \\ o_t &= \sigma(W_o x_t + U_o ht - 1 + b_o) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{17}$$

Nessa formulação, x_t é a entrada na unidade LSTM no tempo t, h_{t-1} é o estado oculto da unidade LSTM no tempo $t-1$, i_t, f_t, $\tilde{c}_t$, c_t, o_t e h_t são as portas de entrada, esquecimento, memória, saída e o estado oculto da unidade LSTM no tempo t, respectivamente. σ é a função de ativação sigmóide, tanh é a função tangente hiperbólica, $\odot$ denota a multiplicação elementar ponto-a-ponto (também conhecida como produto Hadamard) e W e U são matrizes de pesos que são aprendidas durante o treinamento, enquanto b é o vetor de bias.

Essa formulação é apenas um exemplo, já que existem várias variações da arquitetura LSTM, mas ela ilustra a ideia geral da arquitetura. A unidade LSTM é composta por portas que controlam a quantidade de informação que é adicionada, esquecida ou transmitida para a próxima unidade da rede. Além disso, a unidade LSTM mantém uma memória interna que pode ser atualizada ou esquecida ao longo do tempo.

3 Resultados

Após a limpeza dos dados, iniciou-se o processo de análise dos mesmos. Com isso, algumas considerações poderam ser tomadas. Como dito *a priori*, por ser tão abrangente, essa análise foi limitada ao sistema de avaliação feito pelos clientes após a confirmação de entrega do produto. Assim, o consumidor pode dar nota de 1 a 5 para o produto, sendo 1 o mais baixo e 5 o mais alto.

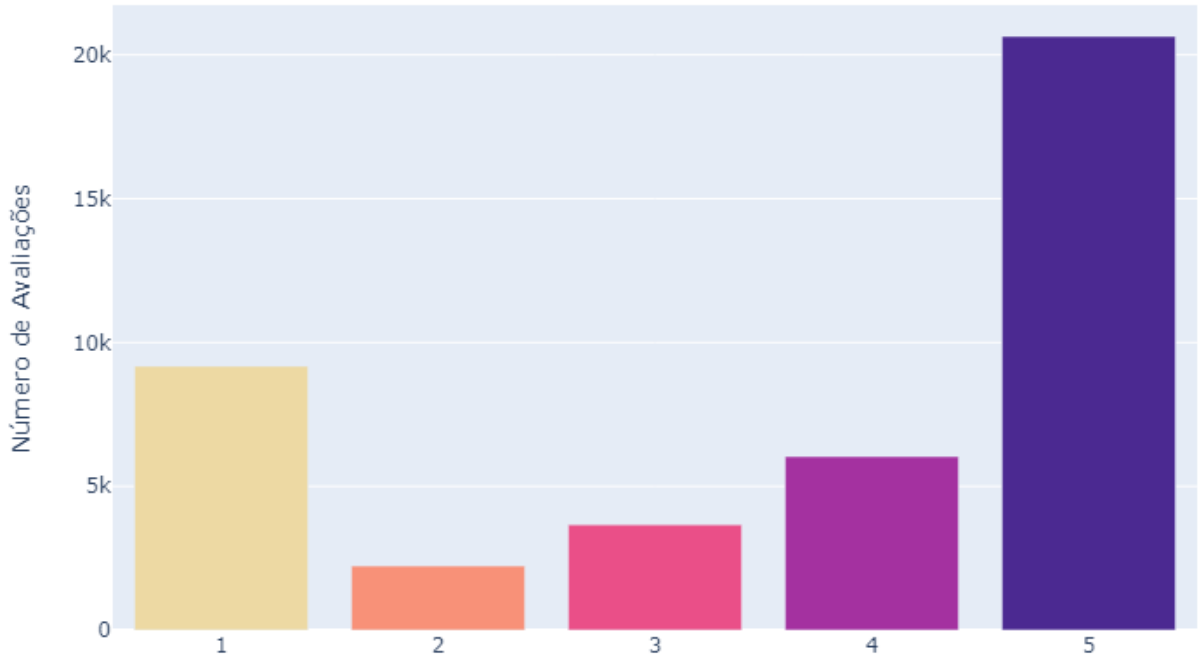

Figura 9: Distribuição das avaliações

À primeira vista com o quantitativo de notas, é possível perceber que essa distribuição é dada em forma de "J"e é bastante típica em *e-commerces*. Grande quantidade de avaliações 5, 4 e 1, pequena quantidade de 2 e 3.

Este tipo de distribuição pode ocorrer por várias razões. Uma possível explicação é que os clientes que estão extremamente satisfeitos ou insatisfeitos com um produto são mais propensos a deixar uma avaliação do que aqueles que têm uma experiência neutra. Isso pode resultar em uma concentração maior de avaliações com classificações muito altas ou muito baixas.

Outra possível explicação é que os clientes que têm uma experiência neutra com um produto podem não se sentir motivados a deixar uma avaliação. Eles podem sentir que o produto foi bom, mas não excepcional, e, portanto, não vale a pena dedicar tempo para escrever uma avaliação. Isso pode levar a um menor número de avaliações com classificações médias.

A distribuição em forma de "J"das avaliações pode ser importante para as empresas entenderem, pois pode fornecer informações sobre a satisfação do cliente e a qualidade do produto. Se um produto tiver uma alta concentração de avaliações muito positivas, pode ser um indicador de forte satisfação do cliente e um produto de alta qualidade. No entanto, também é importante considerar o número de avaliações e a média geral das classificações, já que um pequeno número de avaliações pode distorcer a distribuição.

Seguidamente, plotou-se a distribuição binária das avaliações conforme foi seccionado na etapa de tratamento dos dados. De forma coerente com a Figura 9 e com a divisão adotada, a maior parte das avaliações são positivas.

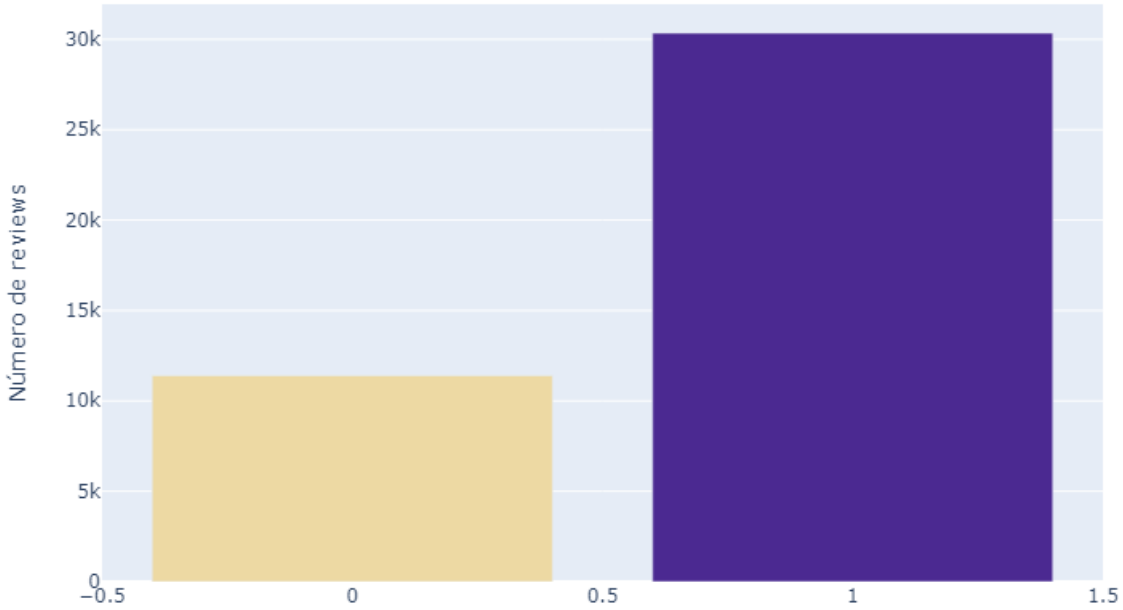

Figura 10: Distribuição das avaliações binárias

Uma nuvem de palavras então é gerada a partir de um conjunto de dados de texto referente aos comentários avaliativos dos clientes. As palavras são extraídas do texto e organizadas em uma nuvem, em que as palavras mais frequentes são maiores e as menos frequentes são menores. A nuvem de palavras pode ser usada para ajudar a identificar tópicos e padrões importantes em um conjunto de dados de texto e para comunicar visualmente essas informações.

Figura 11: Nuvem de palavras destacando os principais termos utilizados

Essa técnica visual pode ser usada em vários campos, incluindo análise de sentimentos, mineração de opiniões, análise de redes sociais, pesquisa de mercado e análise de *feedback* de clientes como o caso em específico. Além disso, por ser uma forma visualmente atraente de resumir informações e destacar pontos importantes, é bastante útil para relatórios e análises.

Tendo feito a tokenização das palavras, isto é, um processo de transformação de todas as palabras de entrada em listas de numeros, filtrado pelas *stopwords* e feito um processo de padronização textual, separou-se o *dataset* entre treino e teste e prosseguiu para as demais análises com os algorítmos de aprendizado de máquina e redes neurais.

Dentre os algoritmos de aprendizado de máquina, encontrou-se as seguintes acurácias conforme a Tabela 4.

modelo	Reg. Logística	F. Aleatórias	XGBoost	Naive Bayes	LightGBM
treino (%)	73.9	99.6	93.5	74.0	86.7
teste (%)	73.3	78.2	82.1	74.0	81.3

Tabela 4: Comapração entre teste e treino nos modelos de aprendizado de máquina

Observa-se que nos modelos XGBoost e LightGBM os valores foram muito próximos e apresentaram os melhores resultados em teste.

A regressão logística tem um tempo de execução muito maior do que os outros modelos, o que pode ser um fator limitante em muitas aplicações. Por outro lado, o Naive Bayes tem um tempo de execução muito curto, mas também tem a menor acurácia entre os modelos apresentados.

Além disso, com as Florestas Aleatórias e XGBoost houve discrepâncias significativas entre as suas respectivas acurácias de treino e de teste. Essa diferença entre a acurácia (ou outra métrica de desempenho) é conhecida como a discrepância de generalização. Isso pode ter acontecido por diferentes motivos:

- Overfitting: O modelo pode memorizar o conjunto de treinamento em vez de aprender os padrões subjacentes dos dados. Isso pode resultar em uma acurácia alta no conjunto de treinamento, mas uma acurácia baixa no conjunto de teste. Para evitar o overfitting, é importante usar técnicas como a regularização, a seleção de características e o aumento de dados.

- Diferenças entre os dados de treinamento e teste: Os dados de treinamento podem ser diferentes dos dados de teste, o que pode levar a discrepâncias na acurácia. Por exemplo, os dados de treinamento podem ter menos variabilidade do que os dados de teste, ou podem conter exemplos raros que não estão presentes no conjunto de teste.

- Tamanho do conjunto de dados: Quando o conjunto de dados é pequeno, é possível que a variação estatística nos resultados seja grande, levando a diferenças na acurácia entre o treinamento e teste.

É importante entender que uma discrepância de generalização não é necessariamente ruim, pois é possível que o modelo esteja apenas se adaptando aos dados de treinamento de forma mais eficaz. No entanto, se a discrepância for muito grande, pode ser um sinal de que o modelo precisa de ajustes. O objetivo é ter um modelo que generalize bem para novos dados e não apenas memorize os dados de treinamento.

Para a rede neural, plotou-se o gráfico da relação das *epochs* pela acuracia do *dataset* de teste e de treino.

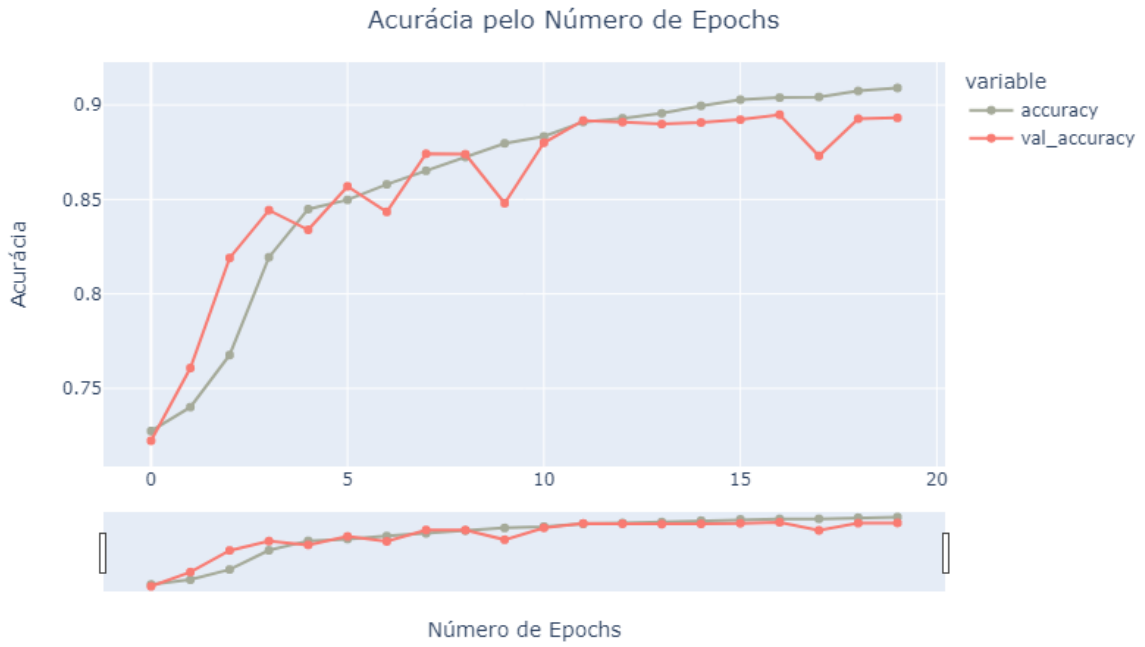

Figura 12: Relação acurácia por *epochs*

A rede neural LSTM tem uma acurácia muito alta, mas um tempo de execução extremamente longo, o que pode torná-la impraticável para muitas aplicações em tempo real. No entanto, para aplicações onde a precisão é o fator mais importante, como diagnóstico médico ou detecção de fraudes financeiras, esse modelo pode ser a melhor opção.

Em redes neurais, uma época refere-se a um ciclo completo de treinamento de todos os dados de treinamento disponíveis na rede. Durante cada um desses ciclos, os dados são passados pela rede em lotes (ou *batch*), a rede faz previsões para cada lote, compara as previsões com as respostas corretas e atualiza os pesos e bias da rede para minimizar o erro (ou função de custo) entre as previsões e as respostas corretas.

Após uma época completa, a rede é capaz de fazer previsões melhores do que antes do treinamento começar. À medida que o treinamento continua avançando, as previsões melhoram gradualmente até que a rede esteja suficientemente treinada para fazer previsões precisas sobre novos dados.

O número de *epochs* que uma rede neural precisa para ser treinada depende do problema e da complexidade da rede. Um número muito baixo de *epochs* pode resultar em previsões imprecisas, enquanto um número muito alto pode levar a um excesso de ajuste (overfitting) dos dados de treinamento.

Por fim, faz-se um comparativo entre a acurácia de todos os modelos para o *dataset* de treino e tem-se a Tabela 5 com o tempo de execução para cada um dos modelos.

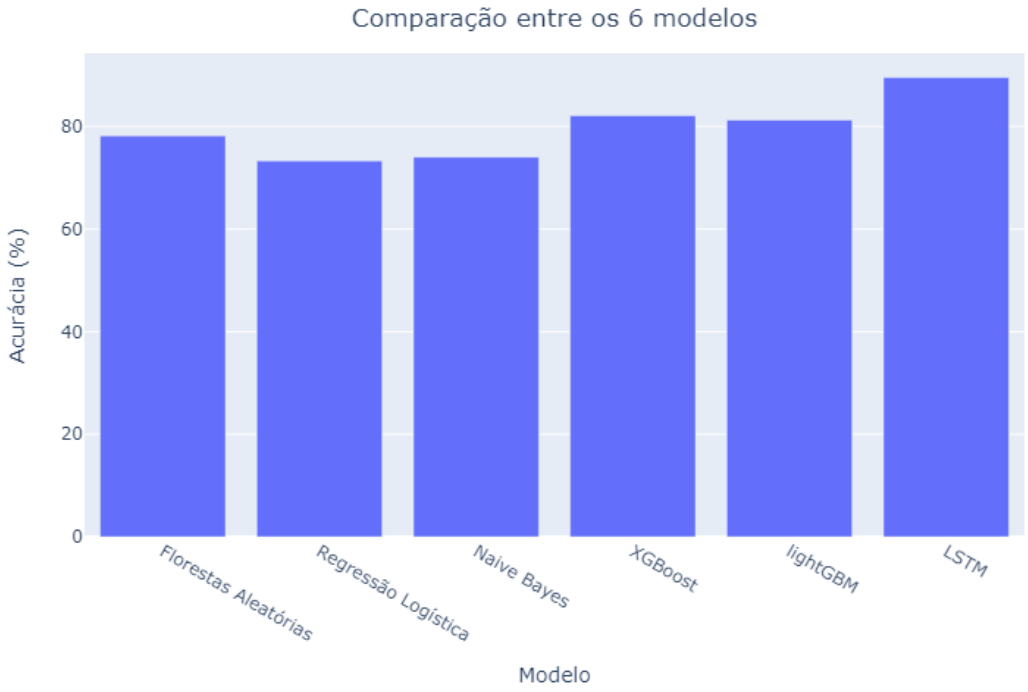

Figura 13: Comparação de acurácia entre modelos

modelo	Reg. Logística	F. Aleatórias	XGBoost	Naive Bayes	LightGBM	LSTM
Tempo (s)	21.5	1.5	3.0	0.1	1.5	1500
Acurácia (%)	73.3	78.2	82.1	74.0	81.3	90.0

Tabela 5: Tempo de execução/Acurácia dos modelos avaliados em teste

Embora a acurácia seja uma métrica importante, ela pode não ser suficiente para avaliar completamente o desempenho de um modelo em alguns casos. Por exemplo, em problemas de classificação desbalanceados, onde uma classe é muito mais comum do que a outra, a acurácia pode ser enganosa. Nesses casos, outras métricas, como a precisão, a recall e a F1-score podem ser mais úteis.

Além disso, a acurácia não leva em consideração a gravidade dos erros de classificação. Por exemplo, em um problema de diagnóstico médico, um falso negativo pode ter consequências mais graves do que um falso positivo. Portanto, é importante avaliar o desempenho do modelo em relação a métricas que levam em consideração a gravidade dos erros de classificação, como a matriz de confusão.

Isso posto, avaliou-se os modelos também pela matriz de confusão e curva ROC.

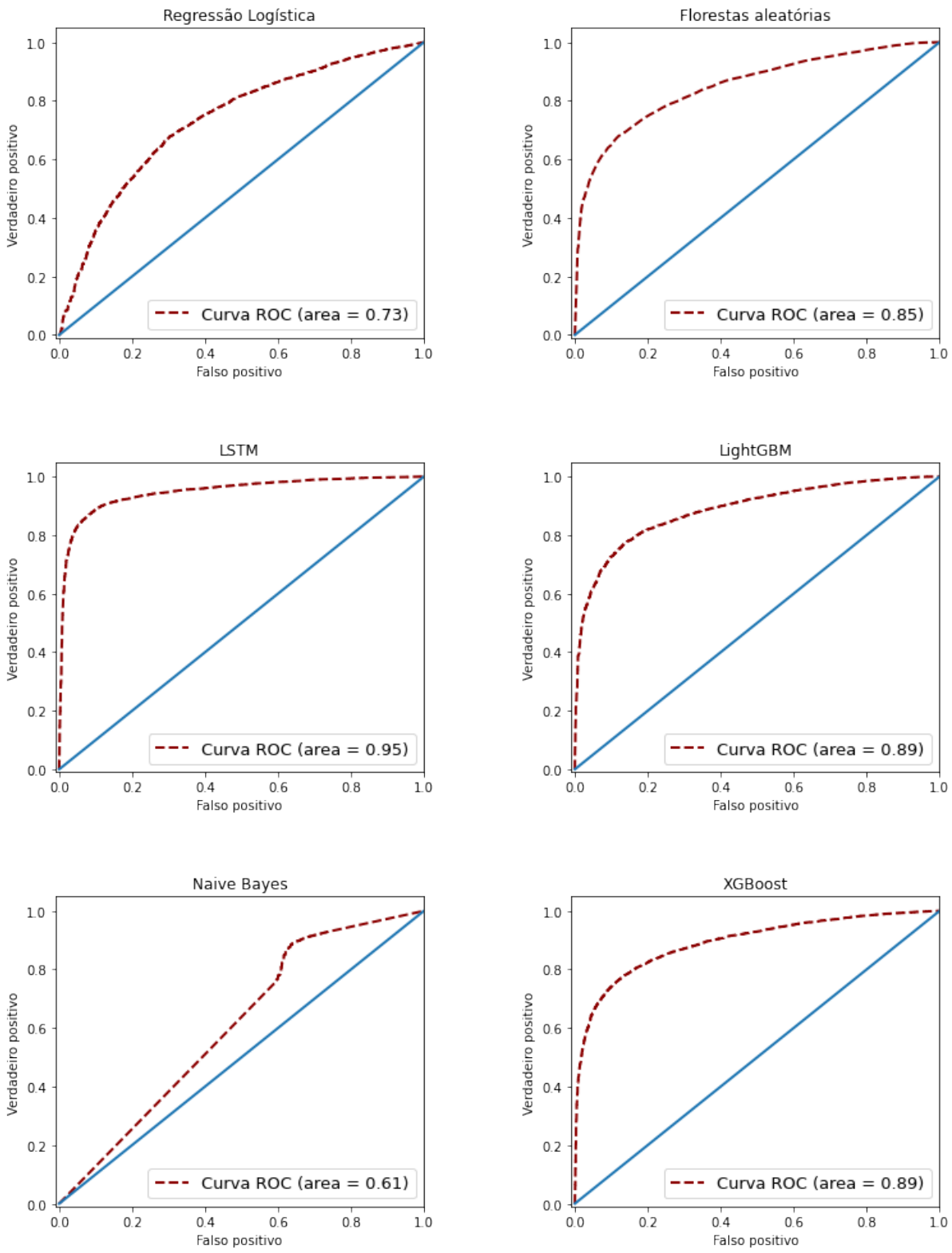

Figura 14: Curvas ROC dos modelos utilizados

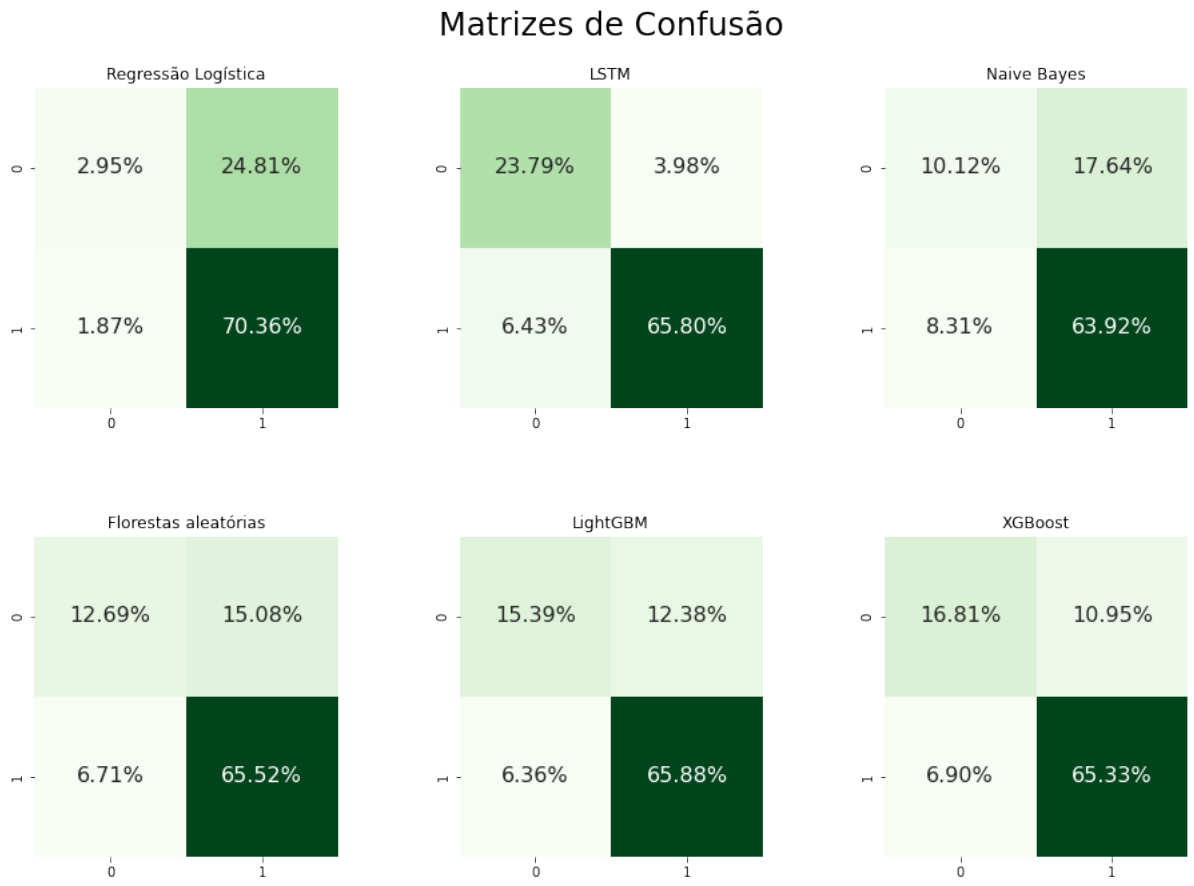

Figura 15: Matrizes de confusão dos modelos utilizados

A matriz de confusão é uma tabela que mostra o número de exemplos que foram classificados corretamente e incorretamente pelo modelo, divididos em quatro categorias: verdadeiro positivo (TP-00), verdadeiro negativo (TN-11), falso positivo (FP-10) e falso negativo (FN-01). A partir dessa matriz, várias métricas de desempenho podem ser calculadas, incluindo a acurácia, precisão e recall. A matriz de confusão pode ajudar a identificar padrões de erros comuns que o modelo está cometendo.

Essas três equações podem ser representadas da seguinte forma:

$$\text{Acurácia} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{FP} + \text{VN} + \text{FN}} \tag{18}$$

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{19}$$

$$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{20}$$

Já a curva ROC (*Receiver Operating Characteristic*) é uma curva que mostra a relação entre a taxa de verdadeiros positivos (TPR) e a taxa de falsos positivos (FPR) para diferentes valores de limiar de classificação. Em outras palavras, a curva ROC mostra como o modelo se comporta em termos de sensibilidade e especificidade para diferentes pontos de corte de probabilidade de classificação. Quanto mais próxima a curva ROC estiver do canto superior esquerdo, melhor será o desempenho do modelo. A área sob a curva ROC (AUC-ROC) é uma medida resumida da performance do modelo, sendo que o valor máximo é 1,0, o que indica uma performance perfeita.

Ao avaliar a curva ROC, podemos observar que a LSTM também apresentou a maior área sob a curva, indicando uma maior capacidade de distinguir entre as classes. Esse resultado é consistente com a análise das matrizes de confusão, que mostraram que a LSTM teve o melhor desempenho na classificação correta das amostras.

O XGBoost e LightGBM também apresentaram resultados promissores, tendo o XGBoost uma acurácia um pouco melhor de 82% e uma área sob a curva ROC de 0,89. No entanto, as matrizes de confusão indicaram que os modelos cometeram mais erros do que a LSTM na classificação de algumas amostras.

Por outro lado, a regressão logística e o Naive Bayes apresentaram as piores performances, com acurácias de 73% e 74%, respectivamente, e áreas sob a curva ROC menores do que os outros modelos avaliados.

A escolha entre usar a matriz de confusão e/ou a curva ROC depende do objetivo do problema e da preferência do usuário. Enquanto a matriz de confusão oferece informações detalhadas sobre o desempenho do modelo em cada classe, a curva ROC pode ser mais útil para escolher o melhor modelo entre vários modelos ou ajustar o ponto de corte de classificação para atingir um determinado objetivo, como minimizar a taxa de falsos positivos ou maximizar a taxa de verdadeiros positivos.

4 Conclusões

Com base na análise dos resultados, podemos concluir que a rede neural artificial LSTM apresentou a melhor performance em relação aos outros algoritmos de *machine learning* avaliados. A acurácia da LSTM foi de 90%, o que indica que o modelo foi capaz de classificar corretamente a grande maioria dos dados.

Se tratando de *tradeoff*, na prática, muitas vezes é necessário encontrar um equilíbrio entre a rapidez da resposta e a precisão do modelo. Isso ocorre porque muitas aplicações em tempo real exigem que a resposta seja obtida em um curto espaço de tempo, enquanto outras aplicações, como pesquisa médica ou financeira, priorizam a precisão.

A escolha do modelo ideal depende de vários fatores, como o tamanho dos dados, a complexidade do problema e a disponibilidade de recursos de computação. Modelos mais complexos, como as redes neurais, podem ter uma acurácia muito alta, mas exigem uma grande quantidade de tempo de processamento, enquanto modelos mais simples, como regressão logística ou Naive Bayes, têm tempos de processamento menores, mas podem ter uma acurácia menor.

As Florestas Aleatórias e o XGBoost são modelos intermediários em termos de complexidade e tempo de execução. Já o LightGBM possui um baixo tempo de execução no geral. Esses modelos podem ser mais adequados para muitas aplicações, pois têm uma acurácia razoável e são ágeis o suficiente para muitas tarefas em tempo real.

Outrossim, é necessário destacar que o XGBoost e LightGBM possuem algumas vantagens em relação a redes neurais como a LSTM.

Uma das delas é a sua capacidade de lidar com dados heterogêneos e faltantes de forma eficiente, o que pode ser um desafio para redes neurais. Além disso, com a visão de um negócio, onde a velocidade é importante, o LightGBM e XGBoost são modelos mais simples e rápidos de treinar do que redes neurais, o que pode ser uma vantagem em cenários em que o tempo e recursos computacionais são limitados.

O XGBoost também apresenta interpretabilidade, uma vez que ele permite identificar as variáveis mais importantes para a classificação dos dados. As redes neurais, embora apresentem alta performance em muitas tarefas, são frequentemente consideradas caixas pretas, dificultando a interpretação dos resultados.

Já o LightGBM é consideravelmente mais rápido do que o XGBoost, pois usa uma técnica de otimização chamada de "histogram-based"para encontrar os melhores pontos de divisão de árvore. Isso permite que ele processe conjuntos de dados maiores e mais complexos em menos tempo.

Outras vantagens envolvendo os dois melhores modelos de aprendizado de máquina envolvem: 1 - O LightGBM usar menos memória do que o XGBoost, pois utiliza uma técnica de amostragem chamada "leaf-wise"em vez de "level-wise". Isso permite que ele crie árvores mais profundas com menos nós. 2 - O LightGBM ter melhor desempenho em conjuntos de dados esparsos ao ser melhor em lidar com conjuntos de dados esparsos, pois utiliza uma técnica de exclusão de zeros para reduzir a sobrecarga computacional em recursos esparsos

Portanto, embora a LSTM tenha apresentado a melhor performance na análise classificatória realizada, o LightGBM e XGBoost podem ser boas opções em cenários em que a interpretabilidade e o processamento de dados faltantes são prioridades.

Por outro lado, a LSTM consegue lidar com dados sequenciais e com dependências de longo prazo, o que pode ser um desafio para algoritmos de aprendizado de máquina tradicionais. Isso torna a LSTM uma escolha popular para tarefas de processamento de linguagem natural, reconhecimento de fala, análise de séries temporais, entre outras aplicações em que o histórico de dados é importante. Além disso, ela é capaz de aprender padrões complexos em dados sequenciais sem a necessidade de engenharia manual de características, o que pode ser um processo demorado e sujeito a erros nos outros algoritmos, incluindo o XGBoost. Isso pode resultar em um desempenho melhor para a LSTM em tarefas em que há uma grande quantidade de dados sequenciais.

Outra vantagem da LSTM é a sua capacidade de lidar com dados de entrada de diferentes tipos e tamanhos, como sequências de palavras, imagens e dados numéricos. Isso torna a LSTM uma escolha popular em aplicações em que os dados podem ter diferentes formatos, como reconhecimento de fala e tradução automática.

Para análise de sentimentos em reviews de usuários, a LSTM pode ser mais vantajosa do que o LightGBM e XGBoost. Isso ocorre porque as análises de sentimentos geralmente envolvem dados sequenciais, como sequências de palavras ou frases em uma revisão, e as LSTMs são projetadas especificamente para trabalhar com dados sequenciais e são capazes de aprender padrões em sequências de dados de maneira mais eficiente do que algoritmos de aprendizado de máquina.

Além disso, as LSTMs têm a capacidade de capturar dependências de longo prazo nos dados sequenciais, o que pode ser importante para a análise de sentimentos em reviews de usuários, já que as opiniões dos usuários muitas vezes são expressas em frases e parágrafos complexos e detalhados. Isso possivelmente pode ser percebido como o único modelo que mostrou maior quantidade de resultados de avaliações negativas corretamente, conforme a sua matriz de confusão. Os demais modelos tiveram dificuldade em classificar corretamente as avaliações negativas dos consumidores.

No entanto, é importante ressaltar que a escolha do algoritmo de *machine learning* depende do conjunto de dados e do problema em questão. Algoritmos mais simples podem ser mais eficazes em alguns cenários, enquanto algoritmos mais complexos, como redes neurais, podem ser necessários para lidar com dados mais complexos e variáveis. Em última análise, a escolha do algoritmo deve ser baseada em uma análise cuidadosa dos dados e das necessidades específicas do problema em questão.

Como sugestão para trabalhos fututos pode-se citar os seguintes:

- Investigação da aplicabilidade dos modelos avaliados em diferentes conjuntos de dados e tipos de problema, para avaliar se os resultados obtidos se mantêm consistentes em diferentes cenários.

- Avaliação de outras métricas de desempenho além da acurácia e área sob a curva ROC, como F1-score, precisão e recall, para avaliar melhor o desempenho dos modelos em diferentes situações.
- Investigação de técnicas de interpretabilidade em redes neurais, como SHAP e LIME, para tornar os resultados desses modelos mais compreensíveis e confiáveis em aplicações práticas.
- Exploração de técnicas de otimização de tempo de execução para redes neurais, como a utilização de GPUs ou técnicas de redução de dimensionalidade, para tornar esses modelos mais práticos para aplicações em tempo real.
- Comparação de diferentes versões dos modelos avaliados, como o uso de árvores de decisão em vez de florestas aleatórias, para avaliar se algumas variações são mais adequadas para determinados tipos de dados ou problemas.

Referências

Abcomm (2020). Comércio eletrônico deve crescer 18bilhões. `https://abcomm.org/noticias/comercio-eletronico-deve-crescer-18-em-2020-e-movimentar-r-106-bilhoes/`. Acessado em 15/11/2022.

Breiman, L. (2001). Random forests. *Machine learning*, **45**(1), 5–32.

Brynjolfsson, E., Hu, Y. J., and Rahman, M. S. (2013). Competing in the age of omnichannel retailing. *MIT Sloan Management Review*, **54**(4), 23–29.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Cutler, A., Cutler, D. R., and Stevens, J. R. (2001). Random forests. In *Machine learning*, pages 157–176. Springer.

da Fonseca, J. (2002). *Apostila de metodologia da pesquisa científica*. João José Saraiva da Fonseca.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, **29**(2-3), 103–130.

efagundes (2021). O que é e-commerce? `https://efagundes.com/artigos/o-que-e-e-commerce/`. Acessado em 15/11/2022.

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

Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm. In *International Conference on Artificial Neural Networks*, pages 850–855. Springer.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied logistic regression*, volume 398. John Wiley & Sons.

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

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., and Ye, Q. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, **30**, 3146–3154.

Kleinbaum, D. G. and Klein, M. (2010). *Logistic regression: a self-learning text*, volume 106. Springer.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, **521**(7553), 436–444.

Malhotra, N. (2006). *Pesquisa de Marketing: Uma Orientação*. Bookman.

McCullagh, P. and Nelder, J. A. (1989). *Generalized linear models*, volume 37. CRC press.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, **5**(4), 115–133.

Mitchell, T. (1997). *Machine learning*. McGraw Hill.

Olist (no date). Sobre nós. `https://olist.com/pt-br/sobre-nos/`. Accessado em 15/11/2022.

Olist and Sionek, A. (2018). Brazilian e-commerce public dataset by olist. `https://www.kaggle.com/dsv/195341`.

PetarV (atualizado em 2023). Tikz. long short-term memory. `https://github.com/PetarV-/TikZ/tree/master/Long%20short-term%20memory`. Acesso em: 09/02/2023.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, **1**(1), 81–106.

Rish, I. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, pages 41–46. AAAI Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Parallel distributed processing: Explorations in the microstructure of cognition. *MIT Press*, **1**.

Ting, K. M. (1998). Discretization: An enabling technique. *Data mining and knowledge discovery*, **2**(4), 391–412.