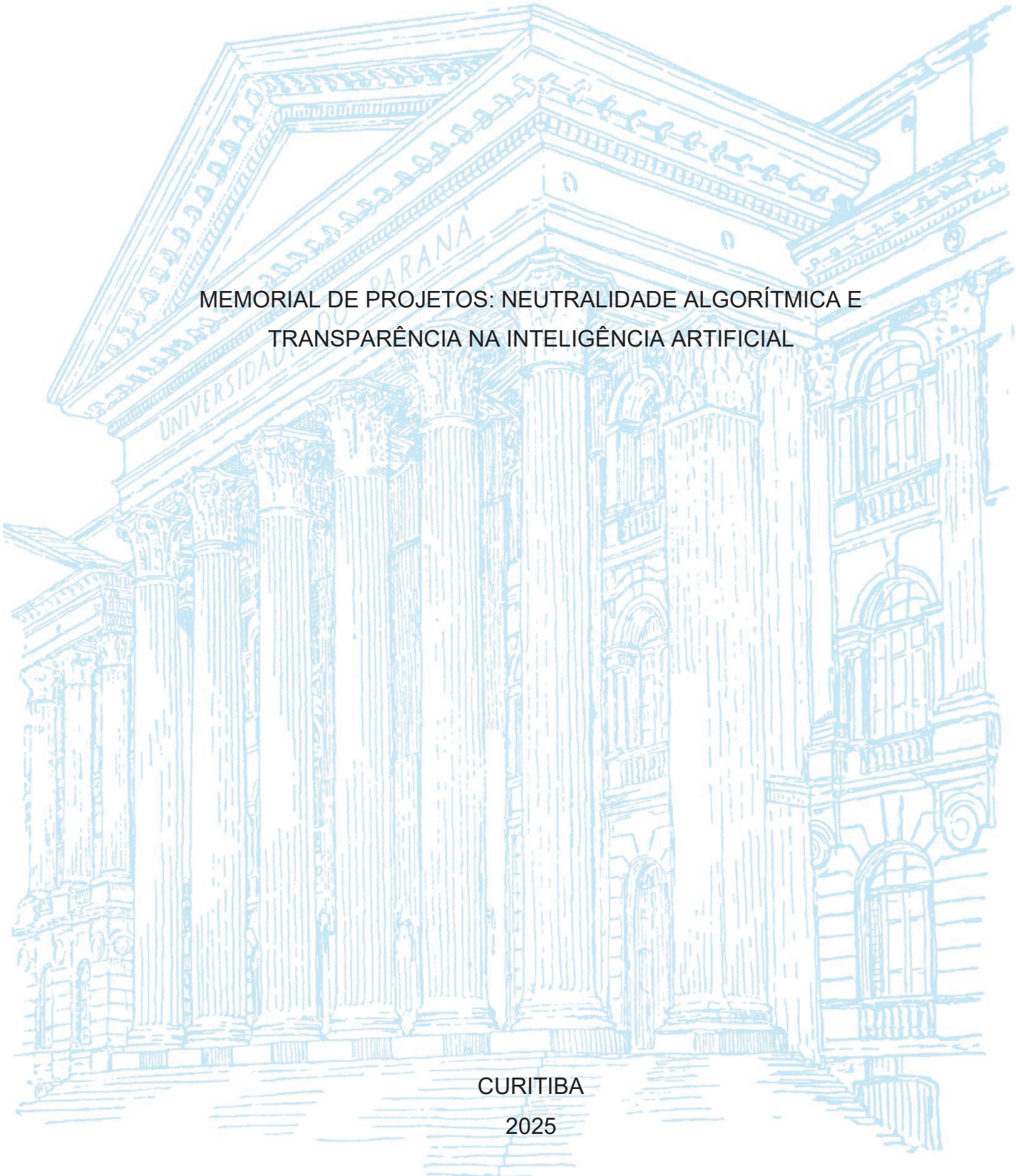
UNIVERSIDADE FEDERAL DO PARANÁ

EDUARDO MARCHETTI DE ARAUJO SERPA



MEMORIAL DE PROJETOS: NEUTRALIDADE ALGORÍTMICA E TRANSPARÊNCIA NA INTELIGÊNCIA ARTIFICIAL

CURITIBA

2025

EDUARDO MARCHETTI DE ARAUJO SERPA

MEMORIAL DE PROJETOS: NEUTRALIDADE ALGORÍTMICA E TRANSPARÊNCIA NA INTELIGÊNCIA ARTIFICIAL

Memorial de Projetos apresentado ao curso de Especialização em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada.

Orientador: Prof. Dr. Jaime Wojciechowski

CURITIBA

2025

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação Inteligência Artificial Aplicada da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **EDUARDO MARCHETTI DE ARAUJO SERPA**, intitulada: **MEMORIAL DE PROJETOS: NEUTRALIDADE ALGORÍTMICA E TRANSPARÊNCIA NA INTELIGÊNCIA ARTIFICIAL**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua **aprovação** no rito de defesa.

A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 23 de Outubro de 2025.

JAIME WOJCIECHOWSKI
Presidente da Banca Examinadora

RAFAELA MANTOVANI FONTANA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

RESUMO

Este trabalho analisa os desafios éticos, sociais e regulatórios relacionados ao uso crescente da Inteligência Artificial (IA), com ênfase na neutralidade algorítmica, na transparência e na responsabilidade dos processos decisórios automatizados. Embora a IA proporcione avanços expressivos em eficiência, personalização e análise de grandes volumes de dados, diversos estudos demonstram que algoritmos podem reproduzir e até amplificar desigualdades humanas pré-existentes. Casos como falhas em sistemas de reconhecimento facial, vieses em ferramentas de recrutamento e decisões discriminatórias em análises de crédito ilustram os riscos à equidade, à inclusão e aos direitos fundamentais. A falta de mecanismos robustos de explicabilidade, auditoria e supervisão humana compromete a confiança pública e a legitimidade das instituições. Para mitigar esses impactos, propõe-se a adoção de um framework de governança algorítmica sustentado em três pilares: diversidade e qualidade dos dados, transparência dos processos e conformidade regulatória. A integração entre tecnologia, ética e direito é essencial para assegurar o desenvolvimento de sistemas justos, auditáveis e responsáveis, alinhados aos princípios de equidade e proteção de direitos no contexto da transformação digital.

Palavras-chave: neutralidade; transparência; ética; regulação; governança.

ABSTRACT

This paper analyzes the ethical, social, and regulatory challenges associated with the growing use of Artificial Intelligence (AI), emphasizing algorithmic neutrality, transparency, and accountability in automated decision-making processes. Although AI offers remarkable advances in efficiency, personalization, and the analysis of large volumes of data, numerous studies show that algorithms can reproduce and even amplify pre-existing human inequalities. Cases such as failures in facial recognition systems, biases in recruitment tools, and discriminatory decisions in credit analysis illustrate the risks to equity, inclusion, and fundamental rights. The lack of robust mechanisms for explainability, auditing, and human oversight undermines public trust and institutional legitimacy. To mitigate these impacts, this study proposes the adoption of an algorithmic governance framework based on three pillars: data diversity and quality, process transparency, and regulatory compliance. Integrating technology, ethics, and law is essential to ensure the development of fair, auditable, and accountable systems aligned with the principles of equity and rights protection in the context of digital transformation.

Keywords: neutrality; transparency; ethics; regulation; governance.

SUMÁRIO

1 PARECER TÉCNICO

A Inteligência Artificial (IA) tem avançado rapidamente e traz ganhos importantes, como maior eficiência, personalização e análise de grandes volumes de dados. No Brasil, a expansão do acesso digital sustenta esse movimento: em 2022, 87,2% das pessoas com 10 anos ou mais usaram a internet, com forte crescimento entre idosos, e o celular segue como principal dispositivo de acesso. De acordo com a Agência IBGE (2023, p. 1), “161,6 milhões de pessoas com 10 anos ou mais utilizaram a Internet no país em 2022.” Estudos do CETIC.br (2024, p. 3) apontam que “o percentual de domicílios com acesso à internet chegou a 84%, com diferenças significativas entre áreas urbanas e rurais.” Essas variações por classe e região importam para a inclusão e para o desenho de sistemas de IA.

Contudo, surgem preocupações éticas e sociais referentes ao seu uso. Rossetti e Angelucci (2021) explicam que a ética algorítmica é um campo emergente voltado a compreender os impactos sociais e morais das tecnologias digitais. Vilalta (2024) complementa que a transparência é um requisito essencial para garantir governança e confiança pública, embora enfrente desafios técnicos e regulatórios.

A neutralidade algorítmica é um dos principais temas abordados em discussões sobre ética no campo da Inteligência Artificial. Segundo Rodrigues e Chai (2023, p. 98), “os algoritmos, embora na teoria sejam externados como dotados de neutralidade e imparcialidade, na prática tendem a refletir determinadas percepções de mundo.” Fonseca (2020) reforça que a percepção de neutralidade é ilusória, pois os códigos carregam vieses humanos.

Um exemplo desse comportamento é o reconhecimento facial, que gera erros mais frequentemente ao identificar pessoas negras. “Os sistemas de reconhecimento facial apresentaram taxas de erro até 100 vezes maiores para indivíduos negros e asiáticos” (Grother et al., 2019, p. 2, tradução nossa). Outro caso é o da Amazon, cujo algoritmo de recrutamento penalizava mulheres por ter sido treinado com dados historicamente masculinos (Reuters, 2018).

Neste contexto, a transparência algorítmica é fundamental para padronizar resultados e corrigir vieses. Rossetti e Angelucci (2021) ressaltam que a explicabilidade é um requisito essencial para assegurar justiça e responsabilidade nos sistemas de inteligência artificial. Pesquisadores complementam que:

> Da mesma maneira, a explicabilidade é essencial para os usuários e para a sociedade. Primeiro, porque o maior conhecimento público acerca dos processos algorítmicos propicia um debate mais informado sobre o implemento e o avanço das novas tecnologias. Em segundo lugar, porque ela possibilita a melhoria das decisões de IA, ajudando a eliminar *outputs* discriminatórios e a mitigar vieses de modelos algorítmicos (Nunes e Andrade, 2023, p. 18).

Garantir a transparência nos sistemas de Inteligência Artificial é crucial para evitar decisões não justificadas. Nascimento (2021, p. 8) reforça que “os problemas relacionados a vieses da automação e da desqualificação estão na vanguarda dessas preocupações” Ruediger (2023) acrescenta que a opacidade algorítmica também ameaça processos democráticos, especialmente em plataformas digitais.

Segundo a União Europeia (2024), a regulamentação tem como objetivo harmonizar normas e promover segurança e confiança no uso da inteligência artificial. Além disso, leis como a LGPD (Brasil, 2018) e o Marco Civil da Internet (Brasil, 2014) estabelecem princípios para proteção de dados e direitos digitais. Como determina a LGPD (Brasil, 2018, art. 1º), “esta Lei dispõe sobre o tratamento de dados pessoais […] com o objetivo de proteger os direitos fundamentais de liberdade e de privacidade e o livre desenvolvimento da personalidade da pessoa natural.”

Do ponto de vista das políticas públicas, a Estratégia Brasileira de Inteligência Artificial (EBIA), coordenada pelo MCTI, orienta ações voltadas ao uso ético, à pesquisa e à inovação, incorporando princípios como transparência e prestação de contas. Segundo o MCTI (2023), a EBIA estabelece diretrizes para fomentar um ecossistema de IA confiável e alinhado aos direitos fundamentais.

A evolução da Inteligência Artificial traz benefícios significativos, mas também amplia desafios éticos e regulatórios. Mesmo com avanços normativos, como legislações nacionais e diretrizes internacionais, a conformidade legal por si só não garante sistemas livres de vieses ou decisões transparentes. De acordo com Zhao (2022), a adoção em larga escala da inteligência artificial requer mecanismos claros de explicabilidade para sustentar a confiança social e evitar impactos negativos sobre direitos fundamentais. Sem essa transparência, áreas sensíveis como saúde, educação e processos democráticos podem ser comprometidas.

Para enfrentar esses riscos, recomenda-se um framework de governança algorítmica estruturado em três pilares: diversidade nos dados, explicabilidade e alinhamento regulatório. Rodrigues e Chai (2023) destacam que a diversidade nos dados é fundamental para reduzir vieses sistêmicos e garantir maior

representatividade nos modelos. Esses elementos, aliados a práticas internas robustas, fortalecem a governança e contribuem para um ecossistema de IA mais confiável e ético.

REFERÊNCIAS

AGÊNCIA IBGE Notícias. **161,6 milhões de pessoas com 10 anos ou mais utilizaram a Internet no país**, em 2022. 9 nov. 2023. Disponível em: < https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/38307-161-6-milhoes-de-pessoas-com-10-anos-ou-mais-de-idade-utilizaram-a-internet-no-pais-em-2022>. Acesso em: 20 out. 2025.

BRASIL. **Lei nº 12.965, de 23 de abril de 2014 (Marco Civil da Internet)**. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2011-2014/2014/lei/l12965.htm>. Acesso em: 20 out. 2025.

BRASIL. **Lei nº 13.709, de 14 de agosto de 2018 (LGPD)**. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm>. Acesso em: 20 out. 2025.

CENTRO REGIONAL DE ESTUDOS PARA O DESENVOLVIMENTO DA SOCIEDADE DA INFORMAÇÃO (CETIC.br); NÚCLEO DE INFORMAÇÃO E COORDENAÇÃO DO PONTO BR (NIC.br). **TIC Domicílios 2023 – Resumo Executivo**. São Paulo: Comitê Gestor da Internet no Brasil, 2024. Disponível em: <https://cetic.br/media/docs/publicacoes/2/20240826110955/resumo_executivo_tic_domicilios_2023.pdf>. Acesso em: 20 out. 2025.

FONSECA, B. **A ilusão da neutralidade algorítmica**. Understanding AI – IEA/USP, 2020. Disponível em: <https://understandingai.iea.usp.br/nota-critica/a-ilusao-da-neutralidade-algoritmica/>. Acesso em: 20 out. 2025.

GRASSI, A.; RUEDIGER, M. A.; et al. **Eleições 2022, desinformação e ataques ao sistema eleitoral**. FGV ECMI, 2023. Disponível em: <https://repositorio.fgv.br/items/8aa942f0-3f01-47d6-9c33-2a20d2581d29/full>. Acesso em: 20 out. 2025.

GROTHER, P.; NGAN, M.; HANAOKA, K. ***FRVT Part 3: Demographic Effects***. NISTIR 8280, 2019. Disponível em: <https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8280.pdf>. Acesso em: 20 out. 2025.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). **PNAD Contínua – Acesso à internet e à televisão e posse de telefone móvel celular para uso pessoal 2022 (Informativo)**. Rio de Janeiro: IBGE, 2023. Disponível em: <https://biblioteca.ibge.gov.br/visualizacao/livros/liv102040_informativo.pdf>. Acesso em: 20 out. 2025.

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO (MCTI). **Inteligência Artificial – Estratégia Brasileira de IA (EBIA)**. Portal do MCTI, 2024. Disponível em: <https://www.gov.br/mcti/pt-br/acompanhe-o-mcti/transformacaodigital/inteligencia-artificial>. Acesso em: 20 out. 2025.

NASCIMENTO, S. M.; et al. **Inteligência artificial e suas implicações éticas e legais: revisão integrativa**. Revista Bioética, 2024. Disponível em: <https://www.scielo.br/j/bioet/a/NjRmBYTfwTy9HFQPf7dm8Ny/?format=pdf>. Acesso em: 20 out. 2025.

NUNES, D. J. C.; ANDRADE, O. M. **O uso da inteligência artificial explicável enquanto ferramenta para compreender decisões automatizadas: possível caminho para aumentar a legitimidade e confiabilidade dos modelos algorítmicos?**. Revista Eletrônica do Curso de Direito da UFSM, v. 18, n. 1, 2023. Disponível em: <https://periodicos.ufsm.br/revistadireito/article/download/69329/61096/379227>. Acesso em: 20 out. 2025.

REUTERS. **Amazon desiste de ferramenta secreta de recrutamento que mostrou viés contra mulheres**. Época Negócios, 10 out. 2018. Disponível em: <https://epocanegocios.globo.com/Empresa/noticia/2018/10/amazon-desiste-de-ferramenta-secreta-de-recrutamento-que-mostrou-vies-contra-mulheres.html>. Acesso em: 20 out. 2025.

RODRIGUES, J. C.; CHAI, C. G. **Inteligência artificial e racismo algorítmico: análise da neutralidade dos algoritmos frente aos episódios de violação de direitos nos meios digitais**. Revista de Direito e Tecnologia, v. 5, n. 2, p. 33–52, 2023. Disponível em: <https://juslaboris.tst.jus.br/bitstream/handle/20.500.12178/215797/2023_rodrigues_julia_inteligencia_artificial.pdf>. Acesso em: 20 out. 2025.

ROSSETTI, R.; ANGELUCCI, A. **Ética algorítmica: questões e desafios éticos do avanço tecnológico da sociedade da informação**. Galáxia (São Paulo), n. 46, 2021. Disponível em: <https://www.scielo.br/j/gal/a/R9F45HyqFZMpQp9BGTfZnyr/?format=html>. Acesso em: 20 out. 2025.

UNIÃO EUROPEIA. **Regulamento (UE) 2024/1689 (AI Act)**. Jornal Oficial da União Europeia, 12 jul. 2024. Disponível em: <https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng>. Acesso em: 20 out. 2025.

VILALTA, L. **É possível termos transparência de algoritmos para sistemas de inteligência artificial?** Jornal da USP, 05 ago. 2024. Disponível em: <https://jornal.usp.br/artigos/e-possivel-termos-transparencia-de-algoritmos-para-sistemas-de-inteligencia-artificial/>. Acesso em: 20 out. 2025.

ZHAO, H.; et al. ***Explainability for Large Language Models: A Survey.*** *ACM Transactions on Intelligent Systems and Technology*, v. 15, n. 2, p. 1–38, fev. 2024. DOI: https://doi.org/10.1145/3639372. Disponível em: <https://dl.acm.org/doi/10.1145/3639372>. Acesso em: 21 out. 2025.

APÊNDICE A - INTRODUÇÃO À INTELIGÊNCIA

A – ENUNCIADO

1 ChatGPT

a) **(6,25 pontos)** Pergunte ao ChatGPT o que é Inteligência Artificial e cole aqui o resultado.
b) **(6,25 pontos)** Dada essa resposta do ChatGPT, classifique usando as 4 abordagens vistas em sala. Explique o porquê.
c) **(6,25 pontos)** Pesquise sobre o funcionamento do ChatGPT (sem perguntar ao próprio ChatGPT) e escreva um texto contendo no máximo 5 parágrafos. Cite as referências.
d) **(6,25 pontos)** Entendendo o que é o ChatGPT, classifique o próprio ChatGPT usando as 4 abordagens vistas em sala. Explique o porquê.

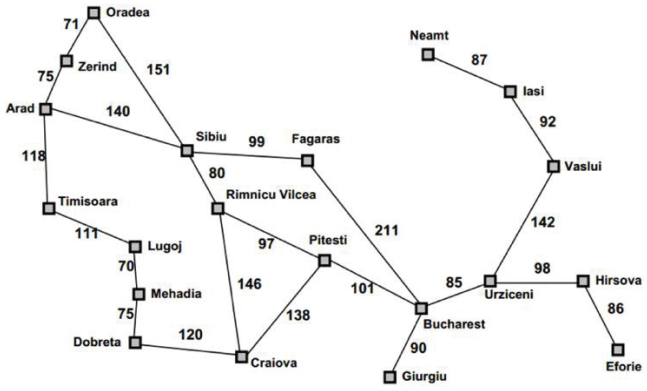
2 Busca Heurística

Realize uma busca utilizando o algoritmo A* para encontrar o melhor caminho para chegar a **Bucharest** partindo de **Lugoj**. Construa a árvore de busca criada pela execução do algoritmo apresentando os valores de *f(n)*, *g(n)* e *h(n)* para cada nó. Utilize a heurística de distância em linha reta, que pode ser observada na tabela abaixo.

Essa tarefa pode ser feita em uma **ferramenta de desenho**, ou até mesmo no **papel**, desde que seja digitalizada (foto) e convertida para PDF.

a) **(25 pontos)** Apresente a árvore final, contendo os valores, da mesma forma que foi apresentado na disciplina e nas práticas. Use o formato de árvore, não será permitido um formato em blocos, planilha, ou qualquer outra representação.

NÃO É NECESSÁRIO IMPLEMENTAR O ALGORITMO.



Arad	366	**Mehadia**	241
Bucareste	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figura 3.22 Valores de *hDLR* — distâncias em linha reta para Bucareste.

3 Lógica

Verificar se o argumento lógico é válido.

Se as uvas caem, então a raposa as come
Se a raposa as come, então estão maduras
As uvas estão verdes ou caem

Logo

A raposa come as uvas se e somente se as uvas caem

Deve ser apresentada uma prova, no mesmo formato mostrado nos conteúdos de aula e nas práticas.

Dicas:

1. Transformar as afirmações para lógica:

p: as uvas caem
q: a raposa come as uvas
r: as uvas estão maduras

2. Transformar as três primeiras sentenças para formar a base de conhecimento

R1: $p \rightarrow q$
R2: $q \rightarrow r$
R3: $\neg r \vee p$

3. Aplicar equivalências e regras de inferência para se obter o resultado esperado. Isto é, com essas três primeiras sentenças devemos derivar $q \leftrightarrow p$. Cuidado com a ordem em que as fórmulas são geradas.

Equivalência Implicação: $(\alpha \rightarrow \beta)$ equivale a $(\neg\alpha \vee \beta)$

Silogismo Hipotético: $\alpha \rightarrow \beta, \beta \rightarrow \gamma \vdash \alpha \rightarrow \gamma$

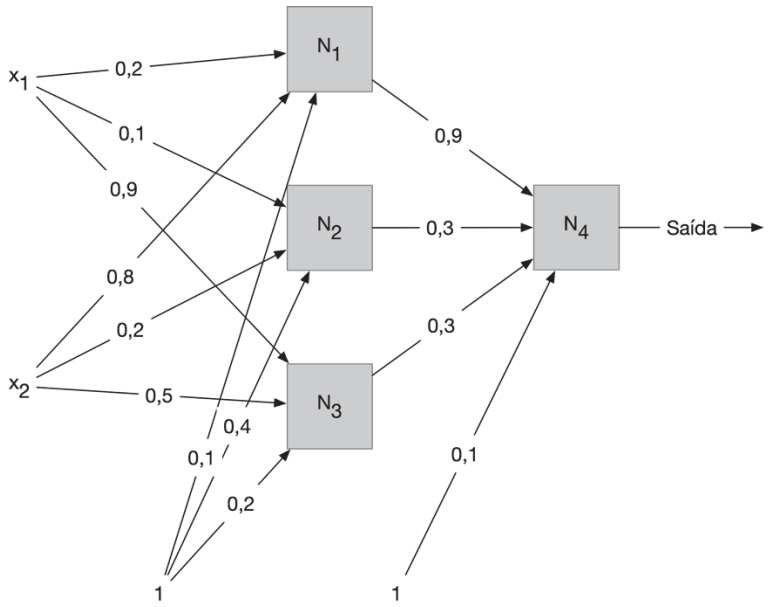
Conjunção: $\alpha, \beta \vdash \alpha \wedge \beta$

Equivalência Bicondicional: $(\alpha \leftrightarrow \beta)$ equivale a $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

a) **(25 pontos)** Deve-se mostrar todos os passos e regras aplicadas, **no mesmo formato apresentado nas aulas e nas práticas**. As equivalências e regras necessárias estão descritas acima e no material.

4 Redes Neurais Artificiais

Seja a RNA da figura abaixo.



Os neurônios N_1, N_2 e N_3 possuem função de ativação linear. Já N_4 possui função de ativação tangente hiperbólica (pesquise a fórmula e aplique).

Dada a entrada x_1=-3, x_2=1, dê:

a) **(6,25 pontos)** Valor de saída do neurônio N1
b) **(6,25 pontos)** Valor de saída do neurônio N2
c) **(6,25 pontos)** Valor de saída do neurônio N3
d) **(6,25 pontos)** Valor de saída da rede como um todo

B – RESOLUÇÃO

Questão 1

a) Model GPT 3.5 Inteligência Artificial (IA) é um campo da ciência da computação que se concentra no desenvolvimento de sistemas e máquinas capazes de realizar tarefas que normalmente requerem inteligência humana. Esses sistemas são projetados para simular processos cognitivos humanos, como aprendizado, raciocínio, resolução de problemas, percepção e compreensão da linguagem natural. A IA abrange uma ampla gama de técnicas e abordagens, incluindo aprendizado de máquina, redes neurais, processamento de linguagem natural, visão computacional e muito mais. Seu objetivo é criar sistemas que possam aprender e se adaptar a novas situações, tornando-se cada vez mais eficientes e autônomos em suas operações. A IA tem aplicações em diversos setores, como saúde, finanças, automotivo, jogos, assistentes virtuais, entre outros.

Model GPT-4 Inteligência Artificial (IA) refere-se à simulação da inteligência humana em máquinas que são programadas para pensar como humanos e imitar suas ações. O termo também pode ser aplicado a qualquer máquina que exiba traços associados à mente humana, como aprendizado e resolução de problemas. A capacidade ideal de uma máquina de IA é realizar tarefas que normalmente requerem inteligência humana, incluindo reconhecimento de padrões, interpretação de linguagem natural, raciocínio, aprendizado a partir da experiência, adaptação a novas situações e a capacidade de manipular objetos no mundo físico.

A IA é um campo amplo com muitas subáreas, incluindo:

1. Aprendizado de Máquina (Machine Learning): Uma abordagem de IA que envolve o treinamento de algoritmos para aprender padrões em dados e fazer previsões ou decisões baseadas em dados novos, sem ser explicitamente programado para realizar a tarefa.
2. Processamento de Linguagem Natural (PLN): Uma subárea da IA focada em dar às máquinas a capacidade de entender, interpretar e responder à linguagem humana de forma útil.
3. Visão Computacional: A capacidade de máquinas e sistemas para derivar informações significativas de imagens digitais, vídeos e outros inputs visuais e tomar ações ou fazer recomendações baseadas nessa informação.
4. Robótica Inteligente: A integração da IA em robôs, permitindo que eles realizem tarefas complexas, aprendam com suas experiências e interajam com o ambiente de maneira mais sofisticada.
5. Sistemas de Recomendação: Algoritmos de IA que analisam o comportamento do usuário e padrões de dados para sugerir itens ou ações que possam ser de interesse do usuário, como visto em plataformas de streaming e lojas online.

A IA pode variar desde sistemas simples, como uma calculadora ou um aplicativo de reconhecimento de voz, até sistemas extremamente complexos, como carros autônomos ou robôs com capacidade de aprendizado profundo. O desenvolvimento da IA promete transformar muitos aspectos da vida cotidiana e tem aplicação em diversos campos, incluindo medicina, educação, finanças, logística, arte, entretenimento e muito mais.

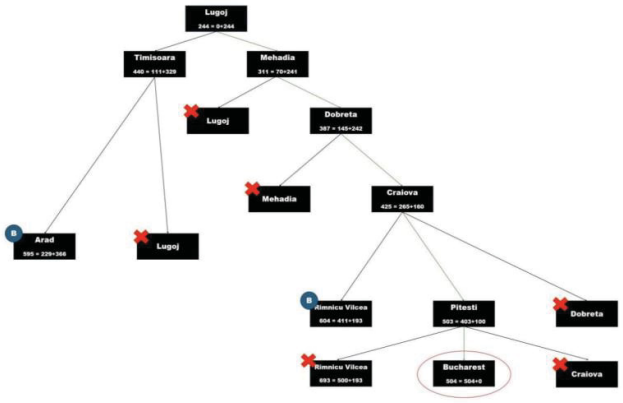
b) Resposta: Conforme a resultado acima classifico a resposta do ChatGPT em AGIR RACIONALMENTE, pois o objetivo GPT é sempre fornecer uma resposta com o melhor resultado e rapidamente conforme as informações fornecidas, ou seja, ele adapta da situação (no caso o usuário), sempre se preocupando em tomar uma ação racional. Ao contrário da abordagem AGIR COMO HUMANO, onde o objetivo é imitar o comportamento humano, algo que não é esperado de uma IA, no qual o objetivo é ter resultados precisos, racionais e rápidos diante das mais diversas situações. O mesmo podemos desconsiderar a abordagem de PENSAR COMO OS HUMANOS, cujo objetivo é implementar fielmente o processo de pensamento, e PENSAR RACIONALMENTE, que visa sempre fornecer um resultado logicamente correto.

c) ChatGPT é uma inteligência artificial baseada em Generative Pre-Trained Transformer, traduzido do inglês significa Transformador Generativo Pré-Treinado, disponibilizado via chatbot por meio de uma interface web e via API REST, permitindo a integração com diversas aplicações. O ChatGPT foi treinado com uma vasta gama de informações (dados), permitindo que ele compreenda e responda a diversas perguntas dos mais variados temas, por exemplo, matemática, saúde, pontos turísticos e entre outros. Por isso, ele também é considerado um modelo de IA conversional e linguagem natural. Os modelos de redes neurais utilizados na criação do ChatGPT são baseados na técnica transformer mencionada no artigo Vaswani et al. (2017). Com isso, modelos do ChatGPT conseguem realizar geração de textos, interpretação de áudios, criação de imagens, análise de imagens se baseando nas informações fornecidas pelo usuário via plataforma web ou API rest.

d) Agir racionalmente (classificação definida) O ChatGPT se alinha mais estreitamente com a abordagem de agir racionalmente. Ele é projetado para realizar tarefas de maneira lógica e eficiente, produzindo respostas que atendem aos requisitos de uma dada solicitação de acordo com seu treinamento e as diretrizes predefinidas. Essa capacidade de gerar respostas apropriadas e úteis num vasto domínio de conhecimento, otimizando para objetivos específicos (como responder a perguntas, fornecer informações, criar imagens, interpretar áudio etc.), exemplifica a ação racional conforme definido na inteligência artificial.

Questão 2

FIGURA 1 – ARVORE DE BUSCA HEURÍSTICA



FONTE: De autoria própria (2024)

Questão 3

a)

Premissas:

P1: $p \rightarrow q$

P2: $q \rightarrow r$

P3: $\neg r \vee p$

Conclusão: Q: q ↔ p

Provar que: p → q , q → r , ~r v p ⊢ q ↔ p

R1 : p → q

R2 : q → r

R3 : ~r v p

R4 : r → p (EI R3)

R5 : q → p (SH R2 e R4)

R6 : (p → q) ^ (q → p) (CONJ R1 e R5)

R7 : p ↔ q (BICOND R6)

O argumento é VÁLIDO

A raposa come as uvas se e somente se as uvas caem.

Questão 4

a)

N1 = (0,2x-3) + (0,8x1) + (01,x1)

N1 = (-0,6) + 0,8 + 0,1

N1 = 0,3

b)

N2 = (0,1x-3) + (0,2x1) + (0,4x1)
N2 = (-0,3) + 0,2 + 0,4
N2 = 0,3

c)
N3 = (0,9x-3) + (0,5x1) + (0,2x1)
N3 = (-2,7) + 0,5 + 0,2
N3 = -2,00

d)
N4 = (0,9x0,3) + (0,3x0,3) + (0,3x-2,0) + (0,1x1)
N4 = 0,27 + 0,09 + (-0,6) + 0,1
N4 = -0,14
N4 = tanh(-0,14) = -0,1390

APÊNDICE B - LINGUAGEM DE PROGRAMAÇÃO APLICADA

A – ENUNCIADO

Nome da base de dados do exercício: *precos_carros_brasil.csv*

Informações sobre a base de dados:

Dados dos preços médios dos carros brasileiros, das mais diversas marcas, no ano de 2021, de acordo com dados extraídos da tabela FIPE (Fundação Instituto de Pesquisas Econômicas). A base original foi extraída do site Kaggle (Acesse aqui a base original). A mesma foi adaptada para ser utilizada no presente exercício.

Observação: As variáveis *fuel*, *gear* e *engine_size* foram extraídas dos valores da coluna *model*, pois na base de dados original não há coluna dedicada a esses valores. Como alguns valores do modelo não contêm as informações do tamanho do motor, este conjunto de dados não contém todos os dados originais da tabela FIPE.

Metadados:

Nome do campo	Descrição
year_of_reference	O preço médio corresponde a um mês de ano de referência
month_of_reference	O preço médio corresponde a um mês de referência, ou seja, a FIPE atualiza sua tabela mensalmente
fipe_code	Código único da FIPE
authentication	Código de autenticação único para consulta no site da FIPE
brand	Marca do carro
model	Modelo do carro
fuel	Tipo de combustível do carro
gear	Tipo de engrenagem do carro
engine_size	Tamanho do motor em centímetros cúbicos

year_model	Ano do modelo do carro. Pode não corresponder ao ano de fabricação
avg_price	Preço médio do carro, em reais

Atenção: ao fazer o download da base de dados, selecione o formato .csv. É o formato que será considerado correto na resolução do exercício.

1 Análise Exploratória dos dados

A partir da base de dados **precos_carros_brasil.csv**, execute as seguintes tarefas:

a. Carregue a base de dados media_precos_carros_brasil.csv
b. Verifique se há valores faltantes nos dados. Caso haja, escolha uma tratativa para resolver o problema de valores faltantes
c. Verifique se há dados duplicados nos dados
d. Crie duas categorias, para separar colunas numéricas e categóricas. Imprima o resumo de informações das variáveis numéricas e categóricas (estatística descritiva dos dados)
e. Imprima a contagem de valores por modelo (model) e marca do carro (brand)
f. Dê um breve explicação (máximo de quatro linhas) sobre os principais resultados encontrados na Análise Exploratória dos dados

2 Visualização dos dados

A partir da base de dados **precos_carros_brasil.csv,** execute as seguintes tarefas:

a. Gere um gráfico da distribuição da quantidade de carros por marca
b. Gere um gráfico da distribuição da quantidade de carros por tipo de engrenagem do carro
c. Gere um gráfico da evolução da média de preço dos carros ao longo dos meses de 2022 (variável de tempo no eixo X)
d. Gere um gráfico da distribuição da média de preço dos carros por marca e tipo de engrenagem
e. Dê uma breve explicação (máximo de quatro linhas) sobre os resultados gerados no item d
f. Gere um gráfico da distribuição da média de preço dos carros por marca e tipo de combustível
g. Dê uma breve explicação (máximo de quatro linhas) sobre os resultados gerados no item f

3 Aplicação de modelos de machine learning para prever o preço médio dos carros

A partir da base de dados **precos_carros_brasil.csv**, execute as seguintes tarefas:

a. Escolha as variáveis **numéricas** (modelos de Regressão) para serem as variáveis independentes do modelo.A variável target é **avg_price. Observação:** caso julgue necessário, faça a transformação de variáveis categóricas em variáveis numéricas para inputar no modelo. Indique **quais variáveis** foram transformadas e **como** foram transformadas
b. Crie partições contendo 75% dos dados para treino e 25% para teste
c. Treine modelos RandomForest (biblioteca RandomForestRegressor) e XGBoost (biblioteca XGBRegressor) para predição dos preços dos carros. **Observação**: caso julgue necessário, mude os parâmetros dos modelos e rode novos modelos. Indique quais parâmetros foram inputados e indique o treinamento de cada modelo
d. Grave os valores preditos em variáveis criadas
e. Realize a análise de importância das variáveis para estimar a variável target, **para cada modelo treinado**

f. Dê uma breve explicação (máximo de quatro linhas) sobre os resultados encontrados na análise de importância de variáveis
g. Escolha o melhor modelo com base nas métricas de avaliação MSE, MAE e R^2
h. Dê uma breve explicação (máximo de quatro linhas) sobre qual modelo gerou o melhor resultado e a métrica de avaliação utilizada

B - RESOLUÇÃO

Questão 1

```
    ### Importação das bibliotecas que serão utilizadas pelo script.
    # Biblioteca Pandas - Manipulação de dados
    import pandas as pd
    # Biblioteca Seaborn - Criação de gráficos
    import seaborn as sns
    # Biblioteca Matplotlib - Criação de gráficos
    import matplotlib.pyplot as plt

    # OPCIONAL - Biblioteca para ignorar mensagens de warning (aviso) ao
rodar uma célula de código
    import warnings
    warnings.filterwarnings('ignore')

    # Bibliotecas de machine learning
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestRegressor
    from xgboost import XGBRegressor
    from sklearn.preprocessing import LabelEncoder

    # Métricas de avaliação dos modelos
    from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

#a. Carreguar a base de dados media_precos_carros_brasil.csv

    # Função read_csv para importar os dados do arquivo
    # > Arquivo .csv precisa estar na mesma pasta do arquivo .ipynb
    dados = pd.read_csv('precos_carros_brasil.csv')

    #Quantidade de linhas e colunas antes do pré-processamento dos dados
    dados.shape

    (267542, 11)
```

```
#b. Verificar se há valores faltantes nos dados. Caso haja, escolha uma tratativa para resolver o problema de valores faltantes

    #Verificando se existem valores faltantes nos dados
    dados.isna().any()

    year_of_reference     True
    month_of_reference    True
    fipe_code             True
    authentication        True
    brand                 True
    model                 True
    fuel                  True
    gear                  True
    engine_size           True
    year_model            True
    avg_price_brl         True
    dtype: bool

    # No resultado comando acima, verificamos que existem valores faltantes em todas as colunas/variáveis

    # Verificando a quantidade de valores faltantes por coluna
    dados.isna().sum()

    year_of_reference     65245
    month_of_reference    65245
    fipe_code             65245
    authentication        65245
    brand                 65245
    model                 65245
    fuel                  65245
    gear                  65245
    engine_size           65245
    year_model            65245
    avg_price_brl         65245
    dtype: int64

    #Optou-se por remover os valores de linhas faltantes
    dados = dados.dropna(axis=0)
```

```
    dados.shape

    (202297, 11)

    # Verificando a quantidade de linhas que sobraram após a exclusão
    dados.shape

    (202297, 11)

#c. Verificar se há dados duplicados nos dados

    # Verificação se existem dados duplicados
    dados.duplicated().sum()

    2

    # Remoção dos dados duplicados no dataframe original
    dados = dados.drop_duplicates()

    # Imprimir o tamanho do dataframe após remoção das duplicações
    dados.shape

    (202295, 11)

#d. Criar duas categorias, para separar colunas numéricas e categóricas.
Imprimir o resumo de informações das variáveis numéricas e categóricas
(estatística descritiva dos dados)

    # Imprimir os tipos de dados do dataframe para identificação da quantidade
de colunas de cada tipo
    dados.dtypes

    year_of_reference     float64
    month_of_reference     object
    fipe_code              object
    authentication         object
    brand                  object
    model                  object
    fuel                   object
    gear                   object
    engine_size            object
```

```
    year_model            float64
    avg_price_brl         float64
    dtype: object

    # Criando 2 categorias para armazenar as colunas numéricas e as
categóricas
    numericas_cols = [col for col in dados.columns if dados[col].dtype !=
'object']
    categoricas_cols = [col for col in dados.columns if dados[col].dtype ==
'object']

    # Resumo das variáveis numéricas - Imprime alguns valores de medidas de
tendências centrais
    dados[numericas_cols].describe()

           year_of_reference     year_model  avg_price_brl
    count      202295.000000  202295.000000  202295.000000
    mean         2021.564695    2011.271514   52756.765713
    std             0.571904       6.376241   51628.912116
    min          2021.000000    2000.000000    6647.000000
    25%          2021.000000    2006.000000   22855.000000
    50%          2022.000000    2012.000000   38027.000000
    75%          2022.000000    2016.000000   64064.000000
    max          2023.000000    2023.000000  979358.000000

    # Resumo das variáveis numéricas - Imprime alguns valores de medidas de
tendências centrais
    dados[categoricas_cols].describe()

           month_of_reference fipe_code authentication   brand  \
    count              202295    202295         202295  202295
    unique                 12      2091         202295       6
    top               January  003281-6   cfzlctzfwrcp    Fiat
    freq                24260       425              1   44962

                                           model           fuel       gear
engine_size
    count                                 202295     202295   202295
202295
    unique                                  2112          3       2          29
    top     Palio Week. Adv/Adv TRYON 1.8 mpi Flex  Gasoline  manual       1,6
```

```
    freq                                                425     168684   161883
47420

    # Converter os dados relativos a anos de float para int

    dados['year_of_reference'] = dados['year_of_reference'].astype(int)
    dados['year_model'] = dados['year_model'].astype(int)
```

e. Imprimir a contagem de valores por modelo (model) e marca do carro (brand)

```
    # Contagem de valores por modelo
    dados['model'].value_counts()

    model
    Palio Week. Adv/Adv TRYON 1.8 mpi Flex    425
    Focus 1.6 S/SE/SE Plus Flex 8V/16V 5p     425
    Focus 2.0 16V/SE/SE Plus Flex 5p Aut.     400
    Saveiro 1.6 Mi/ 1.6 Mi Total Flex 8V      400
    Corvette 5.7/ 6.0, 6.2 Targa/Stingray     375
                                            ...
    STEPWAY Zen Flex 1.0 12V Mec.               2
    Saveiro Robust 1.6 Total Flex 16V CD        2
    Saveiro Robust 1.6 Total Flex 16V           2
    Gol Last Edition 1.0 Flex 12V 5p            2
    Polo Track 1.0 Flex 12V 5p                  2
    Name: count, Length: 2112, dtype: int64

    contagem_modelos = dados['model'].value_counts()
    total_modelos_unicos = len(contagem_modelos)
    print("Total de modelos únicos:", total_modelos_unicos)

    Total de modelos únicos: 2112

    # Contagem de valores por marca
    dados['brand'].value_counts()

    brand
    Fiat               44962
    VW - VolksWagen    44312
    GM - Chevrolet     38590
    Ford               33150
```

```
    Renault            29191
    Nissan             12090
    Name: count, dtype: int64

    contagem_por_marca = dados['brand'].value_counts()
    total_marcas_unicas = len(contagem_por_marca)
    print("Total de marcas únicas:", total_marcas_unicas)

    Total de marcas únicas: 6
```

f) Ao carregar o arquivo, identificou-se um total de 267.542 linhas e 11 colunas. Posteriormente, observou-se que 65.245 linhas continham valores faltantes, as quais foram removidas, resultando em 202.297 linhas. Além disso, foram detectadas e excluídas 2 linhas duplicadas, resultando em 202.295 linhas únicas. Foi realizada uma contagem de valores por modelo, totalizando 2.112, e por marca, totalizando 6.

Questão 2

```
#a. Gerar um gráfico da distribuição da quantidade de carros por marca

    # Gráfico da distribuição por marca
    plt.figure(figsize=(20,10))

grafico=plt.bar(dados['brand'].unique(),dados['brand'].value_counts())#plt.
bar para gráfico de barras
    plt.bar(dados['brand'].unique(),    dados['brand'].value_counts())    #
plt.bar para gráfico de barras. Variáveis nos eixos X e Y
    plt.title('Distribuição da quantidade de carros por marca') # plt.title
para inserir título no gráfico
    plt.ylabel('Total de carros') # # plt.ylabel para inserir título no
gráfico
    plt.bar_label(grafico,fmt="%.00f",size=18, label_type="edge");

[]

#b. Gerar um gráfico da distribuição da quantidade de carros por tipo de
engrenagem do carro.

    # Gráfico da distribuição por tipo de engranagem do carro
    plt.figure(figsize=(15,10))
    #plt.bar para gráfico de barras
    grafico1=plt.bar(dados['gear'].unique(),dados['gear'].value_counts())
```

```
    # plt.bar gráfico de barras.Var nos eixos X e Y
    plt.bar(dados['gear'].unique(), dados['gear'].value_counts())
    # plt.title para inserir título no gráfico
    plt.title('Distribuição do tipo de engranagem do carro')
    # plt.ylabel para inserir título no gráfico
    plt.ylabel('Total de tipos de engrenagem')
    plt.bar_label(grafico1,fmt="%.00f", label_type="edge");
    plt.rcParams.update({'font.size': 18})

[]

#c. Gerar um gráfico da evolução da média de preço dos carros ao longo dos meses de 2022 (variável de tempo no eixo X).

    # Filtrar os dados para o ano de 2022
    dados_2022 = dados[dados['year_of_reference'] == 2022].copy()
    # Converter a coluna 'year_of_reference' para valores inteiros
    dados_2022['year_of_reference'] = dados_2022['year_of_reference'].astype(int)
    # Calculando a média por mês
    media_preco_mes_2022 = dados_2022.groupby(
        ['year_of_reference', 'month_of_reference'])['avg_price_brl'].mean()
    # Ordenando os resultados pelo mês
    media_preco_mes_formatada = media_preco_mes_2022.reindex(
        pd.date_range(start='2022-01-01', end='2022-12-01', freq='MS').strftime('%B'), level=1)
    # Utilizando a função reset_index para criar uma ordem e facilitar a criação do gráfico
    media_preco_mes_formatada = media_preco_mes_formatada.reset_index(name='avg_price_brl')

    # Alterar a variável de int para string
    #dados2022['year_of_reference'] = dados2022['year_of_reference'].astype(str)

    print(media_preco_mes_formatada)

        year_of_reference month_of_reference  avg_price_brl
    0                2022            January   54840.270037
    1                2022           February   55824.519882
```

```
    2                2022              March   56848.951914
    3                2022              April   57150.037325
    4                2022                May   57799.763776
    5                2022               June   58065.611398
    6                2022               July   57893.997056
    7                2022             August   57923.544105
    8                2022          September   58198.936989
    9                2022            October   58227.410144
    10               2022           November   58215.626236
    11               2022           December   57997.243992

    # Gráfico da Média de Preço dos Carros por Mês em 2022
    plt.figure(figsize=(20,10))

grafico2=plt.bar(media_preco_mes_formatada['month_of_reference'],media_prec
o_mes_formatada['avg_price_brl'])#plt.bar para gráfico de barras

plt.bar(media_preco_mes_formatada['month_of_reference'],media_preco_mes_for
matada['avg_price_brl']) # plt.bar para gráfico de barras. Variáveis nos
eixos X e Y
    plt.title('Média de Preço dos Carros por Mês em 2022')
    plt.xlabel('Mês')
    plt.ylabel('Preço Médio')
    plt.bar_label(grafico2, fmt="{:.2f}", size=14, label_type="edge")
    plt.xticks(rotation=45, ha='right')
    plt.rcParams.update({'font.size': 18})

[]

#d. Gerar um gráfico da distribuição da média de preço dos carros por marca
e tipo de engrenagem

    # Verificar quantidade de marcas
    dados['brand'].value_counts()

    brand
    Fiat               44962
    VW - VolksWagen    44312
    GM - Chevrolet     38590
    Ford               33150
    Renault            29191
```

```
    Nissan             12090
    Name: count, dtype: int64

    # Verificar quantidade de tipos de engrenagem
    dados['gear'].value_counts()

    gear
    manual       161883
    automatic     40412
    Name: count, dtype: int64

    # Quantidade de ocorrências de modelo x engrenagem
    #quantidade = dados.groupby(['brand','gear']).size()
    quantidade = dados.groupby('brand')['gear'].value_counts()

    quantidade.head(12)

    brand            gear
    Fiat             manual       42069
                     automatic     2893
    Ford             manual       24876
                     automatic     8274
    GM - Chevrolet   manual       27828
                     automatic    10762
    Nissan           manual        7120
                     automatic     4970
    Renault          manual       21346
                     automatic     7845
    VW - VolksWagen  manual       38644
                     automatic     5668
    Name: count, dtype: int64

    # colocar em um dataframe os dados que serão usados nos gráficos
    mediaPrecoMarcaEngrenagem           =           dados.groupby(['brand',
'gear'])['avg_price_brl'].mean().round(2)
    # Nomeando a coluna do preço médio
    mediaPrecoMarcaEngrenagem                                              =
mediaPrecoMarcaEngrenagem.reset_index(name='Preço Médio')
    mediaPrecoMarcaEngrenagem.head(15)

                    brand       gear  Preço Médio
```

```
    0              Fiat  automatic     97396.80
    1              Fiat     manual     39694.44
    2              Ford  automatic     84769.11
    3              Ford     manual     51784.33
    4    GM - Chevrolet  automatic     88156.92
    5    GM - Chevrolet     manual     52119.42
    6            Nissan  automatic     94230.60
    7            Nissan     manual     52680.62
    8           Renault  automatic     44028.01
    9           Renault     manual     47648.80
    10  VW - VolksWagen  automatic     99734.98
    11  VW - VolksWagen     manual     40390.33

    # Criando o gráfico
    plt.figure(figsize=(35,15))
    plt.xlabel("Marca") # Título do eixo X
    plt.ylabel("Preço Médio") # Título do eixo Y
    plt.title("Distribuição da média de preço dos carros por marca e tipo de
engrenagem") # Aumento do tamanho da fonte
    sns.barplot(x='brand',        y='Preço        Médio',        hue='gear',
data=mediaPrecoMarcaEngrenagem,hue_order=['automatic', 'manual']);
    plt.rcParams.update({'font.size': 20})

[]

#f. Gere um gráfico da distribuição da média de preço dos carros por marca e
tipo de combustível

    # Colocar em um dataframe os dados que serão usados nos gráficos
    mediaPrecoMarcaCombustivel           =           dados.groupby(['brand',
'fuel'])['avg_price_brl'].mean().round(2)
    # Nomeando a coluna do preço médio
    mediaPrecoMarcaCombustivel                                              =
mediaPrecoMarcaCombustivel.reset_index(name='Preço Médio')
    mediaPrecoMarcaCombustivel.head(20)

                brand      fuel  Preço Médio
    0              Fiat   Alcohol     11509.51
    1              Fiat    Diesel     99814.45
    2              Fiat  Gasoline     37197.29
    3              Ford   Alcohol     10148.91
```

```
    4              Ford    Diesel     94525.53
    5              Ford  Gasoline     45844.52
    6    GM - Chevrolet   Alcohol     13697.72
    7    GM - Chevrolet    Diesel     99817.32
    8    GM - Chevrolet  Gasoline     56497.13
    9            Nissan    Diesel     95534.07
    10           Nissan  Gasoline     59043.29
    11          Renault    Diesel    104529.93
    12          Renault  Gasoline     37058.01
    13  VW - VolksWagen   Alcohol     13392.68
    14  VW - VolksWagen    Diesel    139216.28
    15  VW - VolksWagen  Gasoline     44653.80

    # Criando o gráfico
    plt.figure(figsize=(20,10))
    plt.xlabel("Marca", fontsize=20) # Título do eixo X
    plt.ylabel("Preço Médio", fontsize=20) # Título do eixo Y
    plt.title("Distribuição da média de preço dos carros por marca e tipo de
combustível", fontsize=20) # Aumento do tamanho da fonte
    sns.barplot(x='brand',        y='Preço        Médio',        hue='fuel',
data=mediaPrecoMarcaCombustivel,hue_order=['Alcohol',
'Diesel','Gasoline']);
```

e) No gráfico da média de preço dos carros por marca e tipo de engrenagem, destaca-se que a Volkswagen (VW) lidera com o valor médio mais alto em carros automáticos, seguida pela Nissan em carros manuais. Enquanto isso, a Renault e a Fiat registram os valores médios mais baixos, respectivamente, nessas categorias. Essa análise evidencia as variações nos preços médios de acordo com a marca e o tipo de transmissão.

g) No gráfico de distribuição da média de preço dos carros por marca e tipo de combustível, destaca-se que o Diesel é o tipo mais caro, seguido por Gasolina e Álcool, respectivamente. As marcas com os valores mais elevados nas respectivas categorias são Volkswagen (VW), Nissan e GM - Chevrolet. Por outro lado, as marcas com os valores mais baixos são Ford, Renault e Ford, respectivamente. Essa análise evidencia as variações nos preços médios conforme o tipo de combustível e a marca do carro.

Questão 3

```
#a. Escolher as variáveis numéricas (modelos de Regressão) para serem as
variáveis independentes do modelo.A variável target é avg_price.

    # Mostrar o tipo de dados
    dados.dtypes
```

```
year_of_reference       int32
month_of_reference     object
fipe_code              object
authentication         object
brand                  object
model                  object
fuel                   object
gear                   object
engine_size            object
year_model              int32
avg_price_brl         float64
dtype: object

# Exibindo a quantidade de valores únicos por cada variável
dados.nunique()

year_of_reference          3
month_of_reference        12
fipe_code               2091
authentication        202295
brand                      6
model                   2112
fuel                       3
gear                       2
engine_size               29
year_model                24
avg_price_brl          88510
dtype: int64

# Atribuir os valores para outro dataframe
dadosPredicao = dados

# Remoção das colunas desnecessárias
dadosPredicao = dadosPredicao.drop('month_of_reference', axis=1)
dadosPredicao = dadosPredicao.drop('fipe_code', axis=1)
dadosPredicao = dadosPredicao.drop('authentication', axis=1)
#dadosPredicao = dadosPredicao.drop('year_of_reference', axis=1)
dadosPredicao.head(10)
```

```
       year_of_reference          brand                                  model
\
    0               2021  GM - Chevrolet             Corsa Wind 1.0 MPFI /
EFI 2p
    1               2021  GM - Chevrolet             Corsa Wind 1.0 MPFI /
EFI 2p
    2               2021  GM - Chevrolet             Corsa Wind 1.0 MPFI /
EFI 2p
    3               2021  GM - Chevrolet             Corsa Wind 1.0 MPFI /
EFI 2p
    4              2021  GM - Chevrolet  Corsa Pick-Up GL/ Champ 1.6 MPFI /
EFI
    5              2021  GM - Chevrolet  Corsa Pick-Up GL/ Champ 1.6 MPFI /
EFI
    6               2021  GM - Chevrolet     Corsa GL 1.6 MPFI / 1.4 EFI 2p
e 4p
    7              2021  GM - Chevrolet             Corsa Sedan GL 1.6 MPFI
4p
    8              2021  GM - Chevrolet             Corsa Sedan GL 1.6 MPFI
4p
    9              2021  GM - Chevrolet         Corsa Sedan GLS 1.6 16V MPFI
4p

           fuel    gear engine_size  year_model  avg_price_brl
    0  Gasoline  manual           1        2002         9162.0
    1  Gasoline  manual           1        2001         8832.0
    2  Gasoline  manual           1        2000         8388.0
    3   Alcohol  manual           1        2000         8453.0
    4  Gasoline  manual         1,6        2001        12525.0
    5  Gasoline  manual         1,6        2000        12020.0
    6  Gasoline  manual         1,6        2000         9632.0
    7  Gasoline  manual         1,6        2001        10508.0
    8  Gasoline  manual         1,6        2000        10167.0
    9  Gasoline  manual         1,6        2001        12841.0

    # Alterar valores categóricos em numéricos
    #brand                  object
    #model                  object
    #fuel                   object
    #gear                   object
    #engine_size            object
```

```
    #### Inserir importação LabelEncoder no início do arquivo
    dadosPredicao['brand']                                                  =
LabelEncoder().fit_transform(dadosPredicao['brand'])
    dadosPredicao['model']                                                  =
LabelEncoder().fit_transform(dadosPredicao['model'])
    dadosPredicao['fuel']                                                   =
LabelEncoder().fit_transform(dadosPredicao['fuel'])
    dadosPredicao['gear']                                                   =
LabelEncoder().fit_transform(dadosPredicao['gear'])
    dadosPredicao['engine_size']                                            =
LabelEncoder().fit_transform(dadosPredicao['engine_size'])

    #dadosPredicao.head()
    dadosPredicao.value_counts()

year_of_reference   brand   model   fuel   gear   engine_size   year_model
avg_price_brl
2021                5       667     2      1      5             2014         27590.0
10

    # Verificar quantidade de marcas
    dadosPredicao['brand'].value_counts()

    brand
    0    44962
    5    44312
    2    38590
    1    33150
    4    29191
    3    12090
    Name: count, dtype: int64

    # Mapa de correlação das variáveis numéricas com variável Target
    plt.figure(figsize=(20,10))
    sns.heatmap(dadosPredicao.corr("spearman"), annot = True)
    plt.title("Mapa de Correlação das Variáveis Numéricas\n", fontsize = 15)
    plt.show()

[]
```

```
#b. Crie partições contendo 75% dos dados para treino e 25% para teste

    # Variável X contém apenas variáveis numéricas de interesse para a
análise, excluindo a variável target
    X = dadosPredicao.drop(['avg_price_brl'],axis = 1)
    X.head()
```

	year_of_reference	brand	model	fuel	gear	engine_size	year_model
0	2021	2	297	2	1	0	2002
1	2021	2	297	2	1	0	2001
2	2021	2	297	2	1	0	2000
3	2021	2	297	0	1	0	2000
4	2021	2	260	2	1	5	2001

```
    # Variável Y contém apenas a variável target
    Y = dadosPredicao['avg_price_brl']
    Y.head()

    0     9162.0
    1     8832.0
    2     8388.0
    3     8453.0
    4    12525.0
    Name: avg_price_brl, dtype: float64

    # Divisão do parâmetros para treino (75%) e teste (25%)
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
0.25, random_state = 42)

    # Verificação dos dados do treinamento
    print(X_train.shape)
    X_train.head(10)

    (151721, 7)
```

	year_of_reference	brand	model	fuel	gear	engine_size	year_model
156364	2022	0	364	1	1	10	2020
93758	2021	4	1099	2	1	5	2011
43313	2021	1	549	2	1	8	2008
114972	2022	0	385	1	1	10	2013

```
    69993               2021      4    970     2     1            0        2018
    125124              2022      1    898     2     1            0        2016
    171836              2022      0   1994     2     1            0        2013
    87814               2021      5   1695     2     1            5        2017
    59771               2021      1   1469     1     1           15        2007
    85504               2021      4    204     2     1            0        2004

    # Verificação dos dados do teste
    print(X_test.shape)
    X_test.head(10)

    (50574, 7)

            year_of_reference    brand    model    fuel    gear    engine_size
year_model
    180633              2022      0   1235     2     1            5        2015
    13130               2021      4    224     2     1            0        2005
    163315              2022      0   1984     2     1            2        2003
    121464              2022      2   1114     2     1            6        2008
    14044               2021      4    310     2     0            5        2019
    79483               2021      5   1373     2     1            8        2012
    104655              2022      2   1879     1     1            8        2004
    84287               2021      1    451     1     1           21        2007
    103710              2022      5    717     2     1            0        2012
    100088              2022      1    460     1     1           24        2006

    # Verificando a variável target
    Y_test.head()

    180633    42595.0
    13130     10989.0
    163315     9087.0
    121464    26965.0
    14044     57102.0
    Name: avg_price_brl, dtype: float64

#c. Treine modelos RandomForest (biblioteca RandomForestRegressor) e XGBoost
(biblioteca XGBRegressor) para predição dos preços os carros. Observação:
caso julgue necessário, mude os parâmetros dos modelos e rode novos modelos.
Indique quais parâmetros foram inputados e indique o treinamento de cada
modelo
```

```
RandomForest

    # Algoritmo Random Forest, sem especificar nenhum parâmetro (número de
árvores, número de ramificações, etc)
    model_rf = RandomForestRegressor()

    # Ajuste do modelo, de acordo com as variáveis de treinamento
    model_rf.fit(X_train, Y_train)
    # Tempo para rodar esse bloco de código SEM a variável year_of_reference:
23.7s
    # Tempo para rodar esse bloco de código COM a variável year_of_reference:
30.7s

    RandomForestRegressor()

    # Predição dos valores com base nos dados de teste
    valores_preditos_rf = model_rf.predict(X_test)

    # Valores preditos
    valores_preditos_rf

    array([ 42187.96172948,  12079.22055913,   8922.48170197, ...,
           109823.1116493 ,   9643.05146144,  22960.47166006])

XGBoost

    # Importar Bibliotecas
    model_xgboost = XGBRegressor()

    # Ajuste do modelo, de acordo com as variáveis de treinamento
    model_xgboost.fit(X_train, Y_train)

    XGBRegressor(base_score=None, booster=None, callbacks=None,
                 colsample_bylevel=None, colsample_bynode=None,
                 colsample_bytree=None,                          device=None,
early_stopping_rounds=None,
                 enable_categorical=False,                eval_metric=None,
feature_types=None,
                 gamma=None, grow_policy=None, importance_type=None,
```

```
                interaction_constraints=None,          learning_rate=None,
max_bin=None,
                max_cat_threshold=None, max_cat_to_onehot=None,
                max_delta_step=None, max_depth=None, max_leaves=None,
                min_child_weight=None,                          missing=nan,
monotone_constraints=None,
                multi_strategy=None, n_estimators=None, n_jobs=None,
                num_parallel_tree=None, random_state=None, ...)

    # Predição dos valores com base nos dados de teste
    valores_preditos_xgboost = model_xgboost.predict(X_test)
    valores_preditos_xgboost

    array([ 44367.266,  12109.644,  10512.041, ..., 110412.94 ,  11347.781,
            24380.184], dtype=float32)

Random Forest com alteração de parâmetros

    ### Random Forest com alteração de parâmetros

    # Com parâmetros iguais aos do exercício da disciplina
    model_rf_parametros        =        RandomForestRegressor(max_depth=29,
min_samples_leaf=32, min_samples_split=28,
                          n_estimators=208, random_state=43)

    # Ajuste do modelo, de acordo com as variáveis de treinamento
    model_rf_parametros.fit(X_train, Y_train)

    RandomForestRegressor(max_depth=29,                min_samples_leaf=32,
min_samples_split=28,
                          n_estimators=208, random_state=43)

    # Predição dos valores com base nos dados de teste
    valores_preditos_rf_parametros = model_rf_parametros.predict(X_test)

#d. Grave os valores preditos em variáveis criadas

    # Valores preditos no Random Forest sem inclusão de parâmetros
    valores_preditos_rf = model_rf.predict(X_test)

    # Valores preditos no XGboost
```

```
    valores_preditos_xgboost = model_xgboost.predict(X_test)

    # Valores preditos no Random Forest com alteração de parâmetros
    valores_preditos_rf_parametros = model_rf_parametros.predict(X_test)

#e. Realize a análise de importância das variáveis para estimar a variável
target, para cada modelo treinado

    # Análise da Importância das variáveis

    # Random Forest

    model_rf.feature_importances_
    feature_importances = pd.DataFrame(model_rf.feature_importances_, index
=     X_train.columns,     columns=['importance']).sort_values('importance',
ascending = False)
    feature_importances

                        importance
    engine_size           0.452190
    year_model            0.393490
    model                 0.057937
    gear                  0.032744
    fuel                  0.032458
    brand                 0.018702
    year_of_reference     0.012478

    # XGboost

    model_xgboost.feature_importances_
    feature_importances  =  pd.DataFrame(model_xgboost.feature_importances_,
index  =  X_train.columns,  columns=['importance']).sort_values('importance',
ascending = False)
    feature_importances

                        importance
    engine_size           0.443018
    year_model            0.197655
    fuel                  0.154716
    gear                  0.120949
    brand                 0.048413
```

```
    model                0.020340
    year_of_reference    0.014909

    # Random Forest com alteração de parâmetros

    model_rf_parametros.feature_importances_
    feature_importances                                                    =
pd.DataFrame(model_rf_parametros.feature_importances_,        index       =
X_train.columns, columns=['importance']).sort_values('importance', ascending
= False)
    feature_importances

                       importance
    engine_size          0.466002
    year_model           0.411574
    model                0.038422
    fuel                 0.034104
    gear                 0.023301
    brand                0.016202
    year_of_reference    0.010395

#g. Escolha o melhor modelo com base nas métricas de avaliação MSE, MAE e R²

    # Random Forest
    # MSE - calcula o erro quadrático médio das predições do nosso modelo.
Quanto maior o MSE, pior é o modelo.
    mse = mean_squared_error(Y_test, valores_preditos_rf)
    mse

    20013214.759729225

    # Random Forest
    # O MAE calcula a média da diferença absoluta entre o valor predito e o
valor real.
    # Nesse caso, os erros são penalizados linearmente, ou seja, todos terão
o mesmo peso na média.
    mae = mean_absolute_error(Y_test, valores_preditos_rf)
    mae

    2336.850652570212
```

```
    # Random Forest
    # O R² é uma métrica que varia entre 0 e 1 e é uma razão que indica o
quão bom o nosso modelo.
    # Quanto maior seu valor, melhor é o modelo
    r2_score(Y_test, valores_preditos_rf)

    0.9925636190237587

    # XGBoost
    # MSE - calcula o erro quadrático médio das predições do nosso modelo.
Quanto maior o MSE, pior é o modelo.
    mse = mean_squared_error(Y_test, valores_preditos_xgboost)
    mse

    39450171.01302586

    # XGBoost
    # O MAE calcula a média da diferença absoluta entre o valor predito e o
valor real.
    # Nesse caso, os erros são penalizados linearmente, ou seja, todos terão
o mesmo peso na média.
    mae = mean_absolute_error(Y_test, valores_preditos_xgboost)
    mae

    3669.041110367435

    # XGBoost
    # O R² é uma métrica que varia entre 0 e 1 e é uma razão que indica o
quão bom o nosso modelo.
    # Quanto maior seu valor, melhor é o modelo
    r2_score(Y_test, valores_preditos_xgboost)

    0.9853413604584382

    # Random Forest com alteração de parâmetros
    # MSE - calcula o erro quadrático médio das predições do nosso modelo.
Quanto maior o MSE, pior é o modelo.
    mse = mean_squared_error(Y_test, valores_preditos_rf_parametros)
    mse

    146317605.61954153
```

```
    # Random Forest com alteração de parâmetros
    # O MAE calcula a média da diferença absoluta entre o valor predito e o
valor real.
    # Nesse caso, os erros são penalizados linearmente, ou seja, todos terão
o mesmo peso na média.
    mae = mean_absolute_error(Y_test, valores_preditos_rf_parametros)
    mae

    4557.4308940321735

    # Random Forest com alteração de parâmetros
    # O R² é uma métrica que varia entre 0 e 1 e é uma razão que indica o
quão bom o nosso modelo.
    # Quanto maior seu valor, melhor é o modelo
    r2_score(Y_test, valores_preditos_rf_parametros)

    0.9456322498918177
```

f) Na análise de importância de variáveis, observou-se uma consistência em relação à influência das características do veículo, como o tamanho do motor e o ano do modelo, na predição do preço médio. Além disso, fatores como o tipo de combustível e a transmissão também desempenharam um papel significativo. A marca do veículo, o modelo específico e o ano de referência da observação, tiveram uma importância menor, indicando que outros atributos podem ser mais relevantes na determinação do preço médio.

h) O modelo Random Forest teve o menor MAE (2336.85) e o maior R² (0.992563), indicando melhor precisão e explicação da variabilidade dos dados. O XGBoost apresentou um desempenho ligeiramente inferior ao Random Forest, mas ainda muito bom. A alteração dos parâmetros do Random Forest não resultou em um melhor desempenho.

TABELA 1 – MODELOS DE MACHINE LEARNING PARA PREVER PREÇO MÉDIO DOS CARROS

Modelo	Métrica	Resultado
Random Forest	MAE	2336.850652570212
Random Forest	MSE	20013214.759729225
Random Forest	R2	0.9925636190237587
Random Forest com Alteração de parâmetros	MAE	4557.4308940321735
Random Forest com alteração de parâmetros	MSE	146317605.61954153
Random Forest com alteração de parâmetros	R2	0.9456322498918177
XGBoost	MAE	3669.041110367435

XGBoost	MSE	39450171.01302586
XGBoost	R2	0.9853413604584382

FONTE: De autoria própria (2024)

APÊNDICE C - LINGUAGEM R

A – ENUNCIADO

1 Pesquisa com Dados de Satélite (Satellite)

O banco de dados consiste nos valores multiespectrais de pixels em vizinhanças 3x3 em uma imagem de satélite, e na classificação associada ao pixel central em cada vizinhança. O objetivo é prever esta classificação, dados os valores multiespectrais.

Um quadro de imagens do Satélite Landsat com MSS (*Multispectral Scanner System*) consiste em quatro imagens digitais da mesma cena em diferentes bandas espectrais. Duas delas estão na região visível (correspondendo aproximadamente às regiões verde e vermelha do espectro visível) e duas no infravermelho (próximo). Cada pixel é uma palavra binária de 8 bits, com 0 correspondendo a preto e 255 a branco. A resolução espacial de um pixel é de cerca de 80m x 80m. Cada imagem contém 2340 x 3380 desses pixels. O banco de dados é uma subárea (minúscula) de uma cena, consistindo de 82 x 100 pixels. Cada linha de dados corresponde a uma vizinhança quadrada de pixels 3x3 completamente contida dentro da subárea 82x100. Cada linha contém os valores de pixel nas quatro bandas espectrais (convertidas em ASCII) de cada um dos 9 pixels na vizinhança de 3x3 e um número indicando o rótulo de classificação do pixel central.

As classes são: solo vermelho, colheita de algodão, solo cinza, solo cinza úmido, restolho de vegetação, solo cinza muito úmido.

Os dados estão em ordem aleatória e certas linhas de dados foram removidas, portanto você não pode reconstruir a imagem original desse conjunto de dados. Em cada linha de dados, os quatro valores espectrais para o pixel superior esquerdo são dados primeiro, seguidos pelos quatro valores espectrais para o pixel superior central e, em seguida, para o pixel superior direito, e assim por diante, com os pixels lidos em sequência, da esquerda para a direita e de cima para baixo. Assim, os quatro valores espectrais para o pixel central são dados pelos atributos 17, 18, 19 e 20. Se você quiser, pode usar apenas esses quatro atributos, ignorando os outros. Isso evita o problema que surge quando uma vizinhança 3x3 atravessa um limite.

O banco de dados se encontra no pacote **mlbench** e é completo (não possui dados faltantes).

Tarefas:

1. Carregue a base de dados Satellite
2. Crie partições contendo 80% para treino e 20% para teste
3. Treine modelos RandomForest, SVM e RNA para predição destes dados.
4. Escolha o melhor modelo com base em suas matrizes de confusão.
5. Indique qual modelo dá o melhor o resultado e a métrica utilizada

2 Estimativa de Volumes de Árvores

Modelos de aprendizado de máquina são bastante usados na área da engenharia florestal (mensuração florestal) para, por exemplo, estimar o volume de madeira de árvores sem ser necessário abatê-las.

O processo é feito pela coleta de dados (dados observados) através do abate de algumas árvores, onde sua altura, diâmetro na altura do peito (dap), etc, são medidos de forma exata. Com estes dados, treina-se um modelo de AM que pode estimar o volume de outras árvores da população.

Os modelos, chamados alométricos, são usados na área há muitos anos e são baseados em regressão (linear ou não) para encontrar uma equação que descreve os dados. Por exemplo, o modelo de Spurr é dado por:

$$\textbf{Volume} = b0 + b1 * dap^2 * Ht$$

Onde dap é o diâmetro na altura do peito (1,3metros), Ht é a altura total. Tem-se vários modelos alométricos, cada um com uma determinada característica, parâmetros, etc. Um modelo de regressão envolve aplicar os dados observados e encontrar b0 e b1 no modelo apresentado, gerando assim uma equação que pode ser usada para prever o volume de outras árvores.

Dado o arquivo **Volumes.csv**, que contém os dados de observação, escolha um modelo de aprendizado de máquina com a melhor estimativa, a partir da estatística de correlação.

Tarefas

1. Carregar o arquivo Volumes.csv (http://www.razer.net.br/datasets/Volumes.csv)
2. Eliminar a coluna NR, que só apresenta um número sequencial
3. Criar partição de dados: treinamento 80%, teste 20%
4. Usando o pacote "caret", treinar os modelos: Random Forest (rf), SVM (svmRadial), Redes Neurais (neuralnet) e o modelo alométrico de SPURR

    - O modelo alométrico é dado por: Volume = b0 + b1 * dap^2 * Ht

      alom <- nls(VOL ~ b0 + b1*DAP*DAP*HT, dados, start=list(b0=0.5, b1=0.5))

5. Efetue as predições nos dados de teste
6. Crie suas próprias funções (UDF) e calcule as seguintes métricas entre a predição e os dados observados

    - Coeficiente de determinação: R^2

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \widehat{y_i})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

onde y_i é o valor observado, $\widehat{y_i}$ é o valor predito e $\bar{y}$ é a média dos valores y_i observados. Quanto mais perto de 1 melhor é o modelo;

- Erro padrão da estimativa: S_{yx}

$$S_{yx} = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \widehat{y_i})^2}{n-2}}$$

esta métrica indica erro, portanto quanto mais perto de 0 melhor é o modelo;

- Syx%

$$S_{yx}\% = \frac{S_{yx}}{\bar{y}} * 100$$

esta métrica indica porcentagem de erro, portanto quanto mais perto de 0 melhor é o modelo;

7. Escolha o melhor modelo.

B – RESOLUÇÃO

Questão 1

As tarefas de número 1 a 3 serão detalhadas em outro documento e no arquivo R. Em relação às tarefas 4 e 5, após o treinamento dos modelos RandomForest, SVM e RNA (nnet), o modelo SVM foi identificado como o melhor dos 3 modelos, pois leva um tempo de execução semelhante ao RandomForest, porém tem uma acurácia levemente superior de acordo com a comparação entre as matrizes de confusão. O RNA (nnet) teve uma acurácia inferior ao SVM e ao RandomForest e ainda levou muito mais tempo de execução.

Complementando a escolha pelo SVM, a equipe fez testes usando o pacote caret para o treinar os modelos RandomForest e SVM, porém o treinamento levava vários minutos, o que dificulta a execução de testes e por isso optamos por utilizar os pacotes “RandomForest” para o RandomForest e os pacotes “e1071” e “kernlab” para usar no SVM. Estes pacotes deixaram a execução muitos mais rápida, pois levava segundos, enquanto usando o pacote caret, levava alguns minutos para os mesmos algoritmos. Adicionalmente, além do ganho de tempo, os resultados foram semelhantes em relação à acurácia dos modelos.

As métricas utilizadas para escolher o SVM como o melhor modelo para usar neste conjunto de dados foram a acurácia apresentada nas matrizes de confusão em conjunto com o tempo de execução do treinamento.

Abaixo constam os resultados das acurácias da matriz de confusão de cada modelo que foram usadas para a escolha do melhor modelo.

```
Resumo dos Modelos> print(metricas)
Modelo Acuracia
1 RandomForest 0.8512461
2 SVM 0.8551402
3 RNA 0.8045171
```

Os comandos abaixo foram usados para carregamento dos pacotes/bibliotecas, carregamento dos dados para o exercício e para filtrar somente as colunas que serão usadas. Esses comandos não geram saídas no console.

```
# Carregar pacotes
library(mlbench) # Para carregar a base de dados Satellite
library(caret) # Para treinar os modelos de aprendizado de máquina
library(randomForest) # Para treinar o modelo Random Forest
library(kernlab) # Para treinar o modelo SVM
library(e1071) # Para treinar o modelo SVM
library(neuralnet) # Para treinar o modelo Rede Neural

# Carregar a base de dados Satellite
data(Satellite)

# Carregar e selecionar apenas as colunas x.17, x.18, x.19, x.20 e a coluna
de classes
dados_selecionados <- satellite[, c("x.17", "x.18", "x.19", "x.20",
"classes")]
```

Os próximos comandos mostram a estrutura inicial dos dados selecionados:

```
# Visualizar a estrutura inicial dos dados
cat("\nEstrutura inicial dos dados:\n")
str(dados_selecionados)
```

O comando a seguir exibe a saída:

```
Estrutura inicial dos dados:
> str(dados_selecionados)
```

```
'data.frame':   6435 obs. of 5 variables:
$ x.17 : num 92 84 84 76 76 80 80 76 ...
$ x.18 : num 112 103 99 99 99 112 107 91 95 ...
$ x.19 : num 118 104 104 104 108 118 113 104 100 ...
$ x.20 : num 85 81 78 81 81 85 88 85 74 78 ...
$ classes: Factor w/ 6 levels "red soil", "cotton crop", ...
```

Nos próximos comandos, são feitas as separações dos dados para treinamento (80%) e para teste (20%) dos modelos. Os comandos não têm saída específica no console.

```
# Criação de partições contendo 80% para treino e 20% para teste
set.seed(123)
indice <- createDataPartition(dados_selecionados$classes, p = 0.8, list = FALSE)

# Criar conjuntos de treinamento e teste
dados_treino <- dados_selecionados[indice, ]
dados_teste <- dados_selecionados[-indice, ]
```

Os comandos a seguir exibem a quantidade de dados nos conjuntos de treino e de teste

```
# Exibir a quantidade de dados nos conjuntos de treino e teste
cat(paste("\nQuantidade de dados no conjunto de treino:", nrow(dados_treino), "\n"))
cat(paste("Quantidade de dados no conjunto de teste:", nrow(dados_teste), "\n"))
```

A saída do comando é esta:

```
> # Exibir a quantidade de dados nos conjuntos de treino e teste
>   cat(paste("\nQuantidade de dados no conjunto de treino:", nrow(dados_treino), "\n"))
Quantidade de dados no conjunto de treino: 5151
> cat(paste("Quantidade de dados no conjunto de teste:", nrow(dados_teste), "\n"))
Quantidade de dados no conjunto de teste: 1284
```

Os comandos abaixo são usados para treinar os modelos RandomForest, SVM e RNA. O modelo gera uma saída, porém pela quantidade de linhas, não serão incluídas neste documento.

```
#Treinamento de modelos RandomForest, SVM e RNA para predição dos dados
cat(c("Treinando Modelo Random Forest:\n"))
modelo_rf <- randomForest(classes ~ ., data = dados_treino)

cat(c("\nTreinando Modelo SVM:\n"))
modelo_svm <- svm(classes ~ ., data = dados_treino, kernel = "radial")

cat(c("\nTreinando Modelo RNA:\n"))
modelo_rna <- caret::train(classes ~ ., data = dados_treino, method = "nnet")
```

Os comandos abaixo são usados para efetuar a predição dos valores e para gerar a matriz de confusão de cada modelo para possibilitar as comparações entre elas.

```
# avaliação dos modelos com base em suas matrizes de confusão
pred_rf <- predict(modelo_rf, newdata = dados_teste)
confusao_rf <- confusionMatrix(pred_rf, dados_teste$classes)

pred_svm <- predict(modelo_svm, newdata = dados_teste)
confusao_svm <- confusionMatrix(pred_svm, dados_teste$classes)

pred_rna <- predict(modelo_rna, newdata = dados_teste)
confusao_rna <- confusionMatrix(pred_rna, dados_teste$classes)
```

Os últimos comandos exibem as acurácias de cada modelo para possibilitar a análise e escolha do melhor modelo considerando a matriz de confusão.

```
# comparação das métricas de desempenho dos modelos
metricas <- data.frame(Modelo = c("Random Forest", "SVM", "RNA"),
                  Acuracia = c(confusao_rf$overall['Accuracy'],
                               confusao_svm$overall['Accuracy'],
confusao_rna$overall['Accuracy']))

cat(c("\nResumo dos Modelos"))
print(metricas)
```

Questão 2

As tarefas de número 1 a 6 serão detalhadas em outro documento e no arquivo R. Em relação a tarefa 7, após o treinamento dos modelos Random Forest (rf), SVM (svmRadial), Redes Neurais (neuralnet) e modelo alométrico de SPURR, o modelo usando Redes Neurais foi considerado o melhor, pois foi

superior aos demais em todas as métricas de comparação, pois foi o que ficou mais próximo a 1 na métrica R2 e nas métricas Syx e Syx% foi o que ficou mais próximo de zero. A escolha se baseou nos resultados da tabela abaixo.

TABELA 2 – RESULTADO DO TREINAMENTO DE MODELOS COM BASE SATELLITE

Modelo	R2	Syx	Syx_percent
Redes Neurais	0.9099191	0.1208231	8.976552
Alométrico de SPURR	0.8694429	0.1454567	10.806705
Random Forest	0.8486654	0.1566040	11.634890
SVM	0.7899082	0.1845178	13.708744

FONTE: De autoria própria (2024)

Os comandos abaixo foram usados para carregamento dos pacotes/bibliotecas, carregamento dos dados para o exercício, já considerando os separadores e o cabeçalho. Esses comandos não geram saídas no console.

```
# Carregar pacotes
library(mlbench)  # Para carregar o conjunto de dados volumes.csv
library(caret)    # Para treinar os modelos de aprendizado de máquina
library(randomForest) # Para treinar o modelo Random Forest
library(kernlab)  # Para treinar o modelo SVM
library(e1071)    # Para treinar o modelo SVM
library(neuralnet)  # Para treinar o modelo Rede Neural

# Carregar o arquivo volumes.csv
volumes <- read.csv2("http://www.razer.net.br/datasets/volumes.csv", sep =
";", header = TRUE)
```

Este comando exibe a estrutura inicial dos dados, inclusive contendo a coluna "NR".

```
# Visualizar a estrutura inicial dos dados
cat("\nEstrutura inicial dos dados:\n")
str(volumes)
```

A saída do comando é esta abaixo:

```
Estrutura inicial dos dados:
> str(volumes)
```

```
'data.frame':   100 obs. of 5 variables:
$ NR : int 1 2 3 4 5 6 7 8 9 10 ...
$ DAP: num 34 41.5 29.6 34.3 34.5 29.9 28.4 29.5 36.3 36.3 ...
$ HT : num 27 27.9 26.4 27.1 26.2 ...
$ HP : num 1.8 2.75 1.15 1.95 1.1 1.9 2.3 2.4 1.8 1.5 ...
$ VOL: num 0.897 1.62 0.801 1.079 0.98 ...
```

Os próximos comandos irão excluir a coluna "NR" do data frame e posteriormente será exibida a nova estrutura dos dados:

```
# Eliminar a coluna NR
volumes <- volumes[, -1]  # Exclui a primeira coluna

# Visualizar a estrutura inicial dos dados
cat("\nDados após exclusão da coluna NR e conversão para numérico:\n")
str(volumes)
```

A saída do comando é esta abaixo, já indicando que a coluna "NR" foi removida:

```
Dados após exclusão da coluna NR e conversão para numérico:
> str(volumes)
'data.frame':   100 obs. of 4 variables:
$ DAP: num 34 41.5 29.6 34.3 34.5 29.9 28.4 29.5 36.3 36.3 ...
$ HT : num 27 27.9 26.4 27.1 26.2 ...
$ HP : num 1.8 2.75 1.15 1.95 1.1 1.9 2.3 2.4 1.8 1.5 ...
$ VOL: num 0.897 1.62 0.801 1.079 0.98 ...
```

Os comandos abaixo irão criar as partições de treino e de teste, fazendo a separação de 80% para treino e 20% para teste. Os próximos comandos irão exibir as quantidades de dados em cada conjunto.

```
# Criação de partições contendo 80% para treino e 20% para teste
set.seed(123)
indice <- createDataPartition(volumes$VOL, p = 0.8, list = FALSE)

# Criar conjuntos de treinamento e teste
dados_treino <- volumes[indice, ]
dados_teste <- volumes[-indice, ]

cat(paste("\nQuantidade de dados no conjunto de treino:",
nrow(dados_treino)))
```

```
cat(paste("\nQuantidade de dados no conjunto de teste:", nrow(dados_teste)))
```

A saída dos comandos acima está exibida a seguir:

```
Quantidade de dados no conjunto de treino: 80
Quantidade de dados no conjunto de teste: 20
```

Os próximos comandos são usados para treinamento dos modelos e para efetuar as predições.

```
# Treinar os modelos
# Random Forest
cat("\nTreinando Modelo Random Forest:\n")
modelo_rf <- train(VOL ~ ., data = dados_treino, method = "rf")

# SVM
cat("\nTreinando Modelo SVM:\n")
modelo_svm <- train(VOL ~ ., data = dados_treino, method = "svmRadial")

# Redes Neurais
cat("\nTreinando Modelo Redes Neurais\n")
modelo_neuralnet <- neuralnet(VOL ~ ., data = dados_treino)

# Treinando Modelo alométrico de SPURR
cat("\nTreinando Modelo alométrico de SPURR\n")
alom <- NLS(VOL ~ b0 + b1*DAP*HT, data = dados_treino, start = list(b0 = 0.5,
b1 = 0.5))

# Efetuar as predições nos dados de teste
pred_rf <- predict(modelo_rf, newdata = dados_teste)
pred_svm <- predict(modelo_svm, newdata = dados_teste)
pred_neuralnet <- predict(modelo_neuralnet, newdata = dados_teste)
pred_alom <- predict(alom, newdata = dados_teste)
```

Os comandos abaixo têm a finalidade de implementar as funções usadas para calcular as métricas.

```
# Criar funções para calcular as métricas
# Coeficiente de determinação: R2
calcular_R2 <- function(y_real, y_predito) {
  media_y_real <- mean(y_real)
  ss_tot <- sum((y_real - media_y_real)^2)
```

```
  ss_res <- sum((y_real - y_predito)^2)
  r2 <- 1 - (ss_res / ss_tot)
  return(r2)
}

# Erro padrão da estimativa: Syx
calcular_syx <- function(y_real, y_predito) {
  n <- length(y_real)
  syx <- sqrt(sum((y_real - y_predito)^2) / (n - 2))
  return(syx)
}

# Syx%
calcular_syx_percent <- function(y_real, y_predito) {
  syx <- calcular_syx(y_real, y_predito)
  media_y_real <- mean(y_real)
  syx_percent <- (syx / media_y_real) * 100
  return(syx_percent)
}
```

Os comandos abaixo, são usadas as funções criadas anteriormente. Cada função é chamada, os devidos parâmetros são incluídos e cada resultado é atribuído a uma variável.

```
# Calculando as métricas para cada modelo
R2_rf <- calcular_R2(dados_teste$VOL, pred_rf)
Syx_rf <- calcular_syx(dados_teste$VOL, pred_rf)
Syx_percent_rf <- calcular_syx_percent(dados_teste$VOL, pred_rf)

R2_svm <- calcular_R2(dados_teste$VOL, pred_svm)
Syx_svm <- calcular_syx(dados_teste$VOL, pred_svm)
Syx_percent_svm <- calcular_syx_percent(dados_teste$VOL, pred_svm)

R2_neuralnet <- calcular_R2(dados_teste$VOL, pred_neuralnet)
Syx_neuralnet <- calcular_syx(dados_teste$VOL, pred_neuralnet)
Syx_percent_neuralnet       <-       calcular_syx_percent(dados_teste$VOL,
pred_neuralnet)

R2_alom <- calcular_R2(dados_teste$VOL, pred_alom)
Syx_alom <- calcular_syx(dados_teste$VOL, pred_alom)
Syx_percent_alom <- calcular_syx_percent(dados_teste$VOL, pred_alom)
```

Os comandos abaixo criam um data frame com os dados dos resultados das métricas de cada modelo e depois fazem a ordenação considerando a coluna com o resultado da métrica R2.

```
# Escolher o melhor modelo
# Comparando as métricas R2, Syx e Syx_percent para decidir o melhor modelo
resultados <- data.frame(Modelo = c("Random Forest", "SVM", "Redes Neurais",
"Alométrico de SPURR"),
                         R2 = c(R2_rf, R2_svm, R2_neuralnet, R2_alom),
                         Syx = c(Syx_rf, Syx_svm, Syx_neuralnet, Syx_alom),
                         Syx_percent    =    c(Syx_percent_rf,    Syx_percent_svm,
Syx_percent_neuralnet, Syx_percent_alom))

# Ordena o dataframe com base na coluna "R2"
resultados_ordenados <- resultados[order(-resultados$R2), ]

#cat("Resultados dos modelos")
# Exibe os resultados ordenados
print(resultados_ordenados)
```

APÊNDICE D - ESTATÍSTICA APLICADA I

A – ENUNCIADO

1) Gráficos e tabelas

(15 pontos) Elaborar os gráficos box-plot e histograma das variáveis “age” (idade da esposa) e “husage” (idade do marido) e comparar os resultados

(15 pontos) Elaborar a tabela de frequencias das variáveis “age” (idade da esposa) e “husage” (idade do marido) e comparar os resultados

2) Medidas de posição e dispersão

(15 pontos) Calcular a média, mediana e moda das variáveis “age” (idade da esposa) e “husage” (idade do marido) e comparar os resultados

(15 pontos) Calcular a variância, desvio padrão e coeficiente de variação das variáveis “age” (idade da esposa) e “husage” (idade do marido) e comparar os resultados

3) Testes paramétricos ou não paramétricos

(40 pontos) Testar se as médias (se você escolher o teste paramétrico) ou as medianas (se você escolher o teste não paramétrico) das variáveis “age” (idade da esposa) e “husage” (idade do marido) são iguais, construir os intervalos de confiança e comparar os resultados.

Obs:

Você deve fazer os testes necessários (e mostra-los no documento pdf) para saber se você deve usar o unpaired test (paramétrico) ou o teste U de Mann-Whitney (não paramétrico), justifique sua resposta sobre a escolha.

Lembre-se de que os intervalos de confiança já são mostrados nos resultados dos testes citados no item 1 acima.

B – RESOLUÇÃO

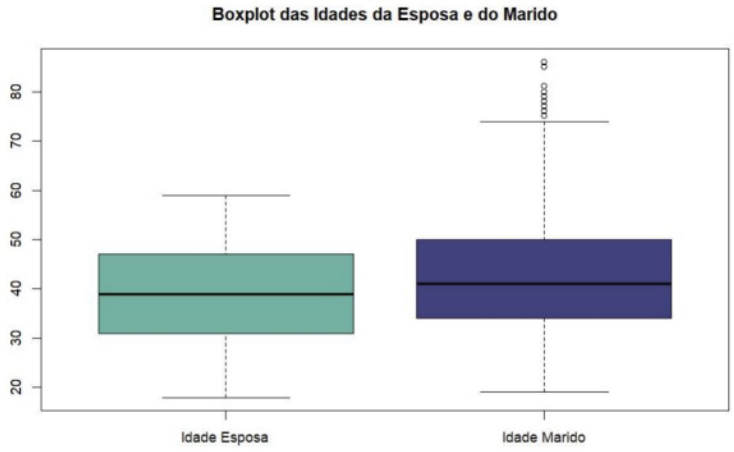
Questão 1

a)

```
# Carregar o arquivo de dados
>load("salarios.RData")
# Visualização dos dados
```

```
>head(salarios)
# Boxplot
> boxplot(salarios$age, salarios$husage,
+ names = c("Idade Esposa", "Idade Marido"),
+ col = c("blue", "red"),
+ main = "Boxplot das Idades da Esposa e do Marido")
```

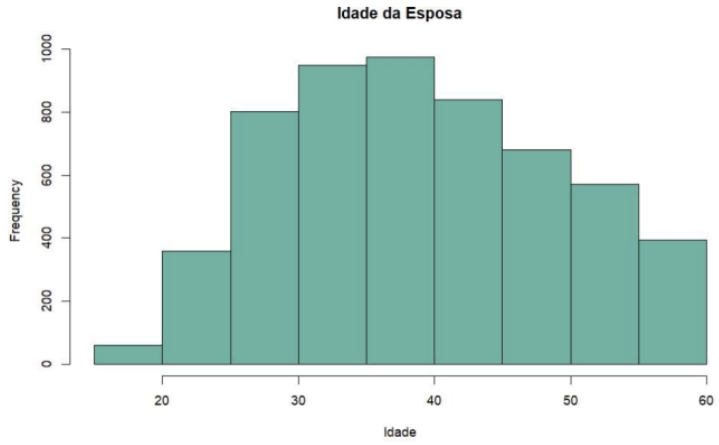
FIGURA 2 – BOXPLOT DAS IDADES DA ESPOSA E DO MARIDO



FONTE: De autoria própria (2024)

```
#Histograma 1
> hist(salarios$age,
+ main = "Idade da Esposa",
+ xlab = "Idade",
+ ylab = "Frequency",
+ col = "#69b3a2")
```

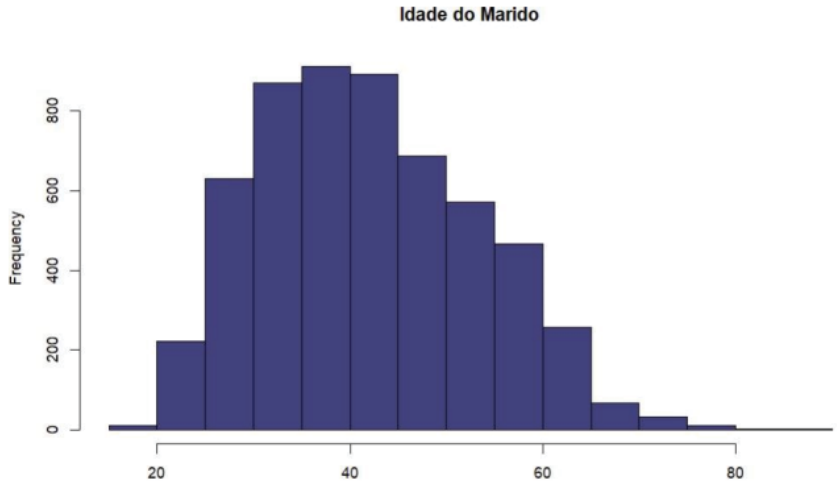
FIGURA 3 – IDADE DA ESPOSA



FONTE: De autoria própria (2024)

```
#Histograma 2
> hist(salarios$husage,
+ main = "Idade do Marido",
+ xlab = "Idade",
+ ylab = "Frequency",
+ col = "#404080")
```

FIGURA 4 – IDADE DO MARIDO

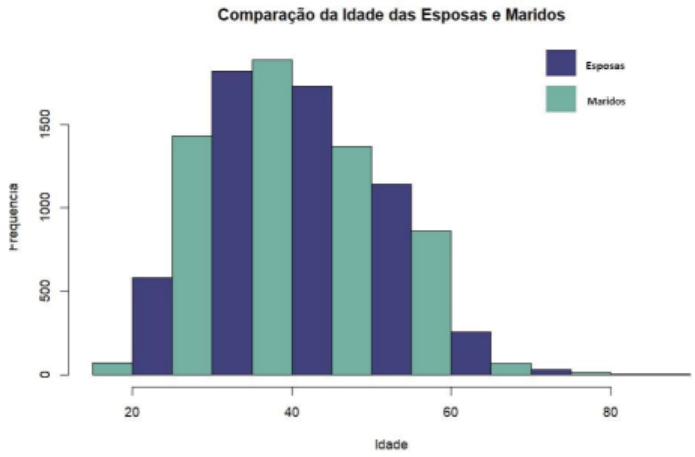


FONTE: De autoria própria (2024)

```
# Gerando histograma comparativo
hist(c(salarios$age, salarios$husage),
```

```
 main = "Comparação da Idade das Esposas e Maridos",
 xlab = "Idade", ylab = "Frequência",
 col = c("#69b3a2", "#404080"),
 breaks = 10,
)
```

FIGURA 5 – COMPARAÇÃO DA IDADE DAS ESPOSAS E MARIDOS



FONTE: De autoria própria (2024)

Comparação dos Histogramas

No histograma com as idades dos maridos existe uma distribuição maior dos dados, visto que existe uma maior varidade de valores observados. Os valores mais elevados podem ser consideradas outliers devido a distância que possuem da média, porém a grande parte dos dados tem maior proximidade com a média.

No histograma com as idades das esposas existe uma concentração ainda maior em relação a média dos dados, além disso existem uma menor varidade de valores observados.

Comparação dos gráficos box plot.

O limite inferior do gráfico com as idades dos maridos é por volta de 20 anos e o limite superior é por volta de 80 anos. Nos valores próximos a 80 anos existem alguns outliers, que já foram identificados anteriormente na análise do histograma.

O limite superior do gráfico das esposas é menor em relação ao gráfico dos maridos, chegando apenas a 60 anos. O limite inferior também é por volta de 20 anos, ficando semelhante com o limite inferior dos maridos. Nessa amosta das idades das esposas não constam outliers.

b)

```
# Calculando a distribuição de frequência
> tabela_age <- fdt(salarios$age)
> tabela_husage <- fdt(salarios$husage)
# Mostrando a distribuicao de frequencia
> print (tabela_husage)
Class limits      f     rf    rf(%) cf    cf(%)
[18.81,23.671)    102   0.02  1.81  102   1.81
[23.671,28.531)   466   0.08  8.27  568   10.08
[28.531,33.392)   809   0.14  14.36 1377  24.44
[33.392,38.253)   895   0.16  15.89 2272  40.33
[38.253,43.114)   917   0.16  16.28 3189  56.60
[43.114,47.974)   629   0.11  11.16 3818  67.77
[47.974,52.835)   649   0.12  11.52 4467  79.29
[52.835,57.696)   541   0.10  9.60  5008  88.89
[57.696,62.556)   394   0.07  6.99  5402  95.88
[62.556,67.417)   152   0.03  2.70  5554  98.58
[67.417,72.278)   51    0.01  0.91  5605  99.49
[72.278,77.139)   21    0.00  0.37  5626  99.86
[77.139,81.999)   6     0.00  0.11  5632  99.96
[81.999,86.86)    2     0.00  0.04  5634  100.00

> print(tabela_age)
Class limits f rf rf(%) cf cf(%)
[17.82,20.804) 61 0.01 1.08 61 1.08
[20.804,23.787) 161 0.03 2.86 222 3.94
[23.787,26.771) 312 0.06 5.54 534 9.48
[26.771,29.754) 505 0.09 8.96 1039 18.44
[29.754,32.738) 562 0.10 9.98 1601 28.42
[32.738,35.721) 571 0.10 10.13 2172 38.55
[35.721,38.705) 624 0.11 11.08 2796 49.63
[38.705,41.689) 510 0.09 9.05 3306 58.68
[41.689,44.672) 542 0.10 9.62 3848 68.30
[44.672,47.656) 432 0.08 7.67 4280 75.97
[47.656,50.639) 389 0.07 6.90 4669 82.87
[50.639,53.623) 358 0.06 6.35 5027 89.23
[53.623,56.606) 304 0.05 5.40 5331 94.62
[56.606,59.59) 303 0.05 5.38 5634 100.00

######### Comparação das Tabelas de Frequência
```

```
# A maior frequência das idades das esposas está entre 35,721 e 38,705 (624 ocorrências) e em seguida o intervalo entre 32,738 e35,721 (571 ocorrências). A menor frequência está no intervalo entre 17,82 e 20,804, contado com apenas 61 ocorrências.

#Na frequência das idades dos maridos constam 917 ocorrências nos intervalos de 38,253 e 43,114, portanto esse intervalo tem a maior frequência. A menor frequência está no intervalo entre 81,999 e 86,86, contando com apenas 2 ocorrências. Outra observação são os intervalos entre 72,278 e 77,139 e entre 77,139 e 81,999 que constam com apenas 6 e 2 ocorrências respectivamente.

#Esses dados citados por último se caracterizam como os outliers da amostra.
```

Questão 2

a)

```
media_age <- mean(salarios$age)
mediana_age <- median(salarios$age)
moda_age <- names(sort(table(salarios$age), decreasing = TRUE))[1]

media_husage <- mean(salarios$husage)
mediana_husage <- median(salarios$husage)
moda_husage <- names(sort(table(salarios$husage), decreasing = TRUE))[1]

# Comparação dos resultados
resultadoA <- data.frame(
  Variavel = c("Idade da Esposa", "Idade do Marido"),
  Media = c(media_age, media_husage),
  Mediana = c(mediana_age, mediana_husage),
  Moda = c(moda_age, moda_husage)
)
print(resultadoA)

  Variavel Media Mediana Moda
1 Idade da Esposa 39.42758 39 37
2 Idade do Marido 42.45296 41 44
>

#Ao analisar as idades de esposas e maridos, percebemos que os maridos geralmente são mais velhos com idade máxima de 86 e para as esposas a idade máxima é 59.
```

```
> summary(salarios$age)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  18.00 31.00 39.00 39.43 47.00 59.00
> summary(salarios$husage)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  19.00 34.00 41.00 42.45 50.00 86.00

#Com uma média de 42,45 anos para os maridos e 39,43 anos para as esposas, a diferença é clara. A mediana também suporta isso, mostrando maridos com 41 anos e esposas com 39.

#Isso indica que, mesmo removendo outliers, a tendência de os maridos serem mais velhos se mantém. Além disso, a moda nos dá uma diferença maior ainda, com 44 anos para maridos e 37 para esposas, o que pode sugerir uma maior dispersão nas idades dos maridos.

#Esses números mostram uma tendência consistente de os maridos serem mais velhos nas relações.
```

b)

```
var_age <- var(salarios$age)
desvio_padrao_age <- sd(salarios$age)
coef_variacao_age <- desvio_padrao_age / media_age * 100

var_husage <- var(salarios$husage)
desvio_padrao_husage <- sd(salarios$husage)
coef_variacao_husage <- desvio_padrao_husage / media_husage * 100

# Comparação dos resultados
resultadoB <- data.frame(
  Variavel = c("Idade da Esposa", "Idade do Marido"),
  Variancia = c(var_age, var_husage),
  Desvio_Padrao = c(desvio_padrao_age, desvio_padrao_husage),
  Coeficiente_de_Variacao = c(coef_variacao_age, coef_variacao_husage)
)
print(resultadoB)

  Variavel Variancia Desvio_Padrao Coeficiente_de_variacao
1 Idade da Esposa 99.75234 9.98761 25.33153
2 Idade do Marido 126.67173 11.22817 26.44849
```

```
#É notável que os maridos são mais velhos, mas também exibem maior variabilidade em suas idades. A análise das variâncias e desvios padrões das idades dá uma ideia de como esses valores se espalham ao redor da média, e os resultados são bem interessantes.

#A variância das idades dos maridos é de 126.07, enquanto a das esposas é de 99.75. Isso mostra que as idades dos maridos estão mais espalhadas, indicando uma diversidade maior. O desvio padrão, dá uma ideia da dispersão em relação à média, segue o mesmo padrão: 11.23 para maridos contra 9.99 para esposas.

#Além disso, o coeficiente de variação nos ajuda a comparar a dispersão entre conjuntos de dados com médias diferentes. Os maridos têm um coeficiente de 26.44%, enquanto as esposas têm 25.33%. Isso significa que, relativamente, a idade dos maridos varia mais em torno da média do que a das esposas.

#Esses números revelam a heterogeneidade dentro dos dois grupos. Uma maior variância e coeficiente de variação nos maridos sugerem que existe uma gama mais ampla de idades entre eles, talvez refletindo normas sociais ou culturais que influenciam a idade em que os homens se casam em comparação com as mulheres.
```

Questão 3

```
options(scipen = 999)

# Verificar o tamanho da amostra
n <- nrow(salarios)

# Como a amostra possui mais de 5000 registros utilizaremos os testes lillie.test e JarqueBeraTest
# para verificar a normalidade

# Testar normalidade das variáveis "age" e "husage" usando o teste de Lilliefors
> lillie.test(salarios$age)

> lillie.test(salarios$age)
Lilliefors (Kolmogorov-Smirnov) normality test

data: salarios$age
D = 0.058909, p-value < 0.00000000000000022
```

```
>lillie.test(salarios$husage)
> lillie.test(salarios$husage)
Lilliefors (Kolmogorov-Smirnov) normality test

data: salarios$husage
D = 0.059662, p-value < 0.00000000000000022

> |

# Testar normalidade das variáveis "age" e "husage" usando o teste de Jarque-
Bera
>JarqueBeraTest(salarios$age, robust = TRUE)
>JarqueBeraTest(salarios$husage, robust = TRUE)

# Ambos deram p-value < 0.00000000000000022 então utilizaremos um teste não
paramétrico
# Mann-Whitney "U" test

# Criando uma data frame com o nome "idade_emp"
idade_emp <- data.frame(
  Grupo = rep(c("husage", "age"), each = nrow(salarios)),
  Idade = c(salarios$age, salarios$husage)
)

# Calculando um sumário estatístico
sumario_estatistico <- idade_emp %>%
  group_by(Grupo) %>%
  summarise(
    count = n(),
    median = median(Idade, na.rm = TRUE),
    IQR = IQR(Idade, na.rm = TRUE)
  )

# Vamos fazer o teste se a idade mediana dos homens é igual a
# idade mediana das mulheres

# Hipóteses do teste:

# H0: A idade mediana dos homens é igual estatisticamente a idade
# mediana das mulheres;
# Ha: A idade mediana dos homens NÃO é igual estatisticamente a idade
```

```
# mediana das mulheres;

# Teste de Wilcoxon
test_result <- wilcox.test(Idade ~ Grupo, data = idade_emp, exact = FALSE,
conf.int = TRUE)
print(test_result)

# O p-value do teste é 0.00000000000000022, que é menor que o nível de
# significância 0.05. Podemos concluir que a idade mediana dos
# homens eh estatisticamente diferente da idade mediana das
# mulheres (rejeitamos H0).
# O intervalo de confiança da diferença entre as medianas esta
# entre 2.000033 e 3.000024

#Após análise de normalidade das variáveis "husage" e "age" utilizando o
teste Lilliefors e
#Jarque-Bera, no qual ambos tiveram resultado de p-value <
0.00000000000000022, que é
#menor que o nível de significância 0.05. Podemos concluir que a idade mediana
dos homens
#é estatisticamente diferente da idade mediana das mulheres (rejeitamos a
hipótese nula de
#que as localizações das distribuições são iguais). Então, optou-se pela
escolha do teste não
#paramétrico Mann-Whitney "U" test. O intervalo de confiança da diferença
entre as
#medianas está entre 2.000033 e 3.000024, isso significa que há uma confiança
de 95% de
#que as localizações das distribuições estão entre esse intervalo.

#A estimativa da diferença nas localizações das distribuições é de 2.99966.
Isso indica que
#a média da variável "age" (idade das esposas) é aproximadamente 3 unidades
menor que a
#média da variável "husage" (idade dos maridos).

#Por isso, infere-se que, há uma diferença estatisticamente significativa na
distribuição da
#variável Idade entre os grupos de Maridos ("husage") e Esposas ("age").

#Segue abaixo o resultado do teste.
```

```
> print(test_result)

wilcoxon rank sum test with continuity correction

data: Idade by Grupo
W = 18122044, p-value < 0.00000000000000022
alternative hypothesis: true location shift is not equal to 0
95 percent confidence interval:
2.000033 3.000024
sample estimates:
difference in location
2.999966
```

APÊNDICE E - ESTATÍSTICA APLICADA II

A – ENUNCIADO

Regressões Ridge, Lasso e ElasticNet

(100 pontos) Fazer as regressões Ridge, Lasso e ElasticNet com a variável dependente “lwage” (salário-hora da esposa em logaritmo neperiano) e todas as demais variáveis da base de dados são variáveis explicativas (todas essas variáveis tentam explicar o salário-hora da esposa). No pdf você deve colocar a rotina utilizada, mostrar em uma tabela as estatísticas dos modelos (RMSE e R^2) e concluir qual o melhor modelo entre os três, e mostrar o resultado da predição com intervalos de confiança para os seguintes valores:

husage = 40 (anos – idade do marido)
husunion = 0 (marido não possui união estável)
husearns = 600 (US$ renda do marido por semana)
huseduc = 13 (anos de estudo do marido)
husblck = 1 (o marido é preto)
hushisp = 0 (o marido não é hispânico)
hushrs = 40 (horas semanais de trabalho do marido)
kidge6 = 1 (possui filhos maiores de 6 anos)
age = 38 (anos – idade da esposa)
black = 0 (a esposa não é preta)
educ = 13 (anos de estudo da esposa)
hispanic = 1 (a esposa é hispânica)
union = 0 (esposa não possui união estável)
exper = 18 (anos de experiência de trabalho da esposa)
kidlt6 = 1 (possui filhos menores de 6 anos)

obs: lembre-se de que a variável dependente “lwage” já está em logarítmo, portanto voçê não precisa aplicar o logaritmo nela para fazer as regressões, mas é necessário aplicar o antilog para obter o resultado da predição.

B – RESOLUÇÃO

Questão 1

a)

Foi efetuado o particionamento do dataset“trabalhosalarios.RData” em 80% dos dados para treinamento e 20% para teste.

Considerando a aplicação dos modelos Ridge, Lasso e ElasticNet, após criação de nova matriz com os valores da Tabela 1, o melhor desempenho para a estimativa do valor hora de salário para a esposa foi o do modelo Ridge, pois este modelo apresentou um RMSE (Erro Quadrático Médio) levemente inferior aos demais e apresentou um R2 (R Quadrado) mais próximo de 1, indicando que se ajusta melhor aos dados observados.

Outra métrica considerada para a escolha do modelo Ridge foi o tempo de execução, pois o modelo Ridge levou menos tempo e ainda apresentou resultados melhores. O modelo Lasso foi levemente superior em relação aos intervalos de confiança, porém em um contexto geral o Ridge se mostrou melhor na maioria das métricas avaliadas. Os dados da tabela abaixo foram extraídos dos processamentos dos scripts em R que foram anexados no arquivo .zip da entrega do trabalho e eles foram usados para a análise e escolha do melhor modelo.

Regressão Ridge

```
# Carregar os pacotes necessários
library(plyr)
library(readr)
library(dplyr)
library(ggplot2)
library(repr)
library(glmnet)
library(caret)

# Carregar a base de dados
load("trabalhosalarios.RData")

# Armazenar o dataset em um objeto denominado "dat"
dat <- trabalhosalarios

# Exibir o dataset completo
View(dat)

# Exibir a estrutura e o resumo das variáveis
glimpse(dat)

# Verificar o uso de memória
gc()

# Desativar a notação científica para facilitar a leitura dos resultados
options(scipen = 999)
```

```
# Definir a semente para reprodutibilidade dos resultados
# A semente garante que a amostragem aleatória seja idêntica em diferentes
execuções
set.seed(302)

# Gerar um índice para particionar o dataset (80% treino / 20% teste)
index = sample(1:nrow(dat), 0.8 * nrow(dat))

# Criar subconjuntos de dados para treino e teste
train = dat[index,]
test = dat[-index,]

# Verificar as dimensões dos conjuntos de treino e teste
dim(train)
dim(test)
# O conjunto de treino possui 2059 observações e 17 variáveis
# O conjunto de teste possui 515 observações e 17 variáveis

# Selecionar variáveis numéricas contínuas para padronização
# Variáveis binárias não são incluídas no processo
cols = c('husage', 'husearns', 'huseduc', 'hushrs',
         'age', 'educ', 'exper', 'lwage')

# Aplicar centralização e escalonamento (z-score normalization)
pre_proc_val <- preProcess(train[, cols], method = c("center", "scale"))
train[, cols] = predict(pre_proc_val, train[, cols])
test[, cols] = predict(pre_proc_val, test[, cols])

# Exibir resumo estatístico das variáveis padronizadas
summary(train)
summary(test)

############################################################
#                      REGRESSÃO RIDGE                     #
############################################################

# A regressão Ridge aplica regularização L2, penalizando a magnitude dos
coeficientes

# Definir as variáveis explicativas e a variável dependente
cols_reg = c('husage', 'husunion', 'husearns', 'huseduc',
```

```
             'husblck', 'hushisp', 'hushrs', 'kidge6',
             'age', 'black', 'educ', 'hispanic',
             'union', 'exper', 'kidlt6', 'lwage')

# Criar variáveis dummies para fatores categóricos e converter para matriz
numérica
# O objetivo é estimar o salário-hora da esposa (lwage)
dummies <- dummyVars(lwage ~ husage + husunion + husearns + huseduc + husblck
+
                       hushisp + hushrs + kidge6 + age + black + educ +
                       hispanic + union + exper + kidlt6,
                     data = dat[, cols_reg])

train_dummies = predict(dummies, newdata = train[, cols_reg])
test_dummies = predict(dummies, newdata = test[, cols_reg])
print(dim(train_dummies)); print(dim(test_dummies))

# Converter os dados em matrizes para uso no modelo Ridge
x = as.matrix(train_dummies)
y_train = train$lwage
x_test = as.matrix(test_dummies)
y_test = test$lwage

# Definir a sequência de valores de lambda a serem testados (10^2 até 10^-3)
lambdas <- 10^seq(2, -3, by = -.1)

# Realizar validação cruzada para determinar o lambda ótimo (alpha = 0 →
Ridge)
ridge_lamb <- cv.glmnet(x, y_train, alpha = 0, lambda = lambdas)
best_lambda_ridge <- ridge_lamb$lambda.min
best_lambda_ridge

# Medir o tempo de processamento da estimativa do modelo
start <- Sys.time()
ridge_reg = glmnet(x, y_train, nlambda = 25, alpha = 0,
                   family = 'gaussian',
                   lambda = best_lambda_ridge)
end <- Sys.time()
difftime(end, start, units = "secs")

# Exibir os coeficientes estimados do modelo Ridge
```

```
ridge_reg[["beta"]]

# Função para calcular métricas de desempenho (R² e RMSE)
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE / nrow(df))

  data.frame(
    RMSE = RMSE,
    Rsquare = R_square
  )
}

# Predição e avaliação no conjunto de treino
predictions_train <- predict(ridge_reg,
                             s = best_lambda_ridge,
                             newx = x)
eval_results(y_train, predictions_train, train)

# Predição e avaliação no conjunto de teste
predictions_test <- predict(ridge_reg,
                            s = best_lambda_ridge,
                            newx = x_test)
eval_results(y_test, predictions_test, test)

# Exemplo de predição para um novo perfil de observação
# Os valores devem ser padronizados conforme o pré-processamento aplicado
# Variáveis dummies permanecem inalteradas

# Definir os valores de entrada padronizados
husage      =      (40      -      pre_proc_val[["mean"]][["husage"]])      /
pre_proc_val[["std"]][["husage"]]
husunion = 0
husearns     =     (600     -     pre_proc_val[["mean"]][["husearns"]])     /
pre_proc_val[["std"]][["husearns"]]
huseduc      =      (13      -      pre_proc_val[["mean"]][["huseduc"]])      /
pre_proc_val[["std"]][["huseduc"]]
husblck = 1
hushisp = 0
```

```
hushrs      =      (40      -      pre_proc_val[["mean"]][["hushrs"]])      /
pre_proc_val[["std"]][["hushrs"]]
kidge6 = 1
age = (38 - pre_proc_val[["mean"]][["age"]]) / pre_proc_val[["std"]][["age"]]
black = 0
educ       =       (13       -       pre_proc_val[["mean"]][["educ"]])       /
pre_proc_val[["std"]][["educ"]]
hispanic = 1
union = 0
exper      =      (18      -      pre_proc_val[["mean"]][["exper"]])      /
pre_proc_val[["std"]][["exper"]]
kidlt6 = 1

# Construir matriz de predição
our_pred = as.matrix(data.frame(husage = husage,
                                husunion = husunion,
                                husearns = husearns,
                                huseduc = huseduc,
                                husblck = husblck,
                                hushisp = hushisp,
                                hushrs = hushrs,
                                kidge6 = kidge6,
                                age = age,
                                black = black,
                                educ = educ,
                                hispanic = hispanic,
                                union = union,
                                exper = exper,
                                kidlt6 = kidlt6))

# Realizar a predição
predict_our_ridge <- predict(ridge_reg,
                             s = best_lambda_ridge,
                             newx = our_pred)
predict_our_ridge

# Converter o valor padronizado para a escala original
wage_pred_ridge = (predict_our_ridge * pre_proc_val[["std"]][["lwage"]]) +
  pre_proc_val[["mean"]][["lwage"]]

# Aplicar a transformação inversa do logaritmo para obter o salário em dólares
```

```
exp(wage_pred_ridge)

# O valor obtido representa o salário-hora estimado em US$

# Cálculo do intervalo de confiança para a predição
n <- nrow(train)  # tamanho da amostra
m <- wage_pred_ridge  # média estimada
s <- pre_proc_val[["std"]][["lwage"]]  # desvio padrão
dam <- s / sqrt(n)  # erro padrão da média
CIlwr_ridge <- m + (qnorm(0.025)) * dam  # limite inferior
CIupr_ridge <- m - (qnorm(0.025)) * dam  # limite superior

# Intervalo de confiança na escala original
exp(CIlwr_ridge)
exp(CIupr_ridge)

# O salário-hora estimado é de aproximadamente US$ 8.68,
#   com   intervalo   de   confiança   entre   US$   8.59   e   US$   8.97.
```

Regressão Lasso

```
# Carregar pacotes necessários
library(plyr)
library(readr)
library(dplyr)
library(ggplot2)
library(repr)
library(glmnet)
library(caret)

# Carregar a base de dados
load("trabalhosalarios.RData")

# Armazenar o dataset em um objeto denominado "dat"
dat <- trabalhosalarios

# Exibir o dataset completo
View(dat)

# Exibir estrutura e resumo das variáveis
glimpse(dat)
```

```
# Verificar uso atual de memória
gc()

# Definir semente para reprodutibilidade de resultados
#  A  semente  garante  particionamento  aleatório  idêntico  em  diferentes
execuções
set.seed(302)

# Gerar índice para particionar o dataset (80% treino / 20% teste)
index = sample(1:nrow(dat), 0.8 * nrow(dat))

# Criar subconjuntos de dados para treino e teste
train = dat[index,]
test = dat[-index,]

# Verificar as dimensões dos conjuntos de treino e teste
dim(train)
dim(test)
# O conjunto de treino contém 2059 observações e 17 variáveis
# O conjunto de teste contém 515 observações e 17 variáveis

# Selecionar variáveis contínuas para padronização
# Variáveis binárias não são incluídas nesse processo
cols = c('husage', 'husearns', 'huseduc', 'hushrs',
         'age', 'educ', 'exper', 'lwage')

# Aplicar centralização e escalonamento (z-score normalization)
pre_proc_val <- preProcess(train[, cols], method = c("center", "scale"))
train[, cols] = predict(pre_proc_val, train[, cols])
test[, cols] = predict(pre_proc_val, test[, cols])

# Exibir resumo estatístico das variáveis padronizadas
summary(train)
summary(test)

############################################################
#                      REGRESSÃO LASSO                     #
############################################################
```

```
# A regressão Lasso aplica regularização L1, promovendo seleção automática
de variáveis

# Definir variáveis explicativas e dependente
cols_reg = c('husage', 'husunion', 'husearns', 'huseduc',
             'husblck', 'hushisp', 'hushrs', 'kidge6',
             'age', 'black', 'educ', 'hispanic',
             'union', 'exper', 'kidlt6', 'lwage')

# Criar variáveis dummies e converter os datasets em formato matricial
# O objetivo é estimar o salário-hora da esposa (lwage)
dummies <- dummyVars(lwage ~ husage + husunion + husearns + huseduc +
                       husblck + hushisp + hushrs + kidge6 + age +
                       black + educ + hispanic + union + exper + kidlt6,
                     data = dat[, cols_reg])

train_dummies = predict(dummies, newdata = train[, cols_reg])
test_dummies = predict(dummies, newdata = test[, cols_reg])
print(dim(train_dummies)); print(dim(test_dummies))

# Converter os dados em matrizes para uso no modelo Lasso
x = as.matrix(train_dummies)
y_train = train$lwage
x_test = as.matrix(test_dummies)
y_test = test$lwage

# Definir sequência de valores de lambda (10^2 a 10^-3)
lambdas <- 10^seq(2, -3, by = -.1)

# Realizar validação cruzada para determinar o lambda ótimo (alpha = 1 →
Lasso)
lasso_lamb <- cv.glmnet(x, y_train, alpha = 1,
                        lambda = lambdas,
                        standardize = TRUE,
                        nfolds = 5)

# Obter o valor ótimo de lambda
best_lambda_lasso <- lasso_lamb$lambda.min
best_lambda_lasso

# Medir tempo de processamento da estimativa do modelo
```

```
start <- Sys.time()
lasso_reg <- glmnet(x, y_train, alpha = 1,
                    lambda = best_lambda_lasso,
                    standardize = TRUE)
end <- Sys.time()
difftime(end, start, units = "secs")

# Exibir os coeficientes estimados do modelo Lasso
lasso_reg[["beta"]]

# Função para cálculo das métricas de desempenho (R² e RMSE)
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE / nrow(df))

  data.frame(
    RMSE = RMSE,
    Rsquare = R_square
  )
}

# Predição e avaliação no conjunto de treino
predictions_train <- predict(lasso_reg,
                             s = best_lambda_lasso,
                             newx = x)
eval_results(y_train, predictions_train, train)

# Predição e avaliação no conjunto de teste
predictions_test <- predict(lasso_reg,
                            s = best_lambda_lasso,
                            newx = x_test)
eval_results(y_test, predictions_test, test)

# Exemplo de predição para uma nova observação
# Como o dataset foi padronizado, é necessário aplicar a mesma transformação
# As variáveis dummies não são padronizadas

# Definir os valores de entrada padronizados
```

```
husage = (40 - pre_proc_val[["mean"]][["husage"]]) /
pre_proc_val[["std"]][["husage"]]
husunion = 0
husearns = (600 - pre_proc_val[["mean"]][["husearns"]]) /
pre_proc_val[["std"]][["husearns"]]
huseduc = (13 - pre_proc_val[["mean"]][["huseduc"]]) /
pre_proc_val[["std"]][["huseduc"]]
husblck = 1
hushisp = 0
hushrs = (40 - pre_proc_val[["mean"]][["hushrs"]]) /
pre_proc_val[["std"]][["hushrs"]]
kidge6 = 1
age = (38 - pre_proc_val[["mean"]][["age"]]) / pre_proc_val[["std"]][["age"]]
black = 0
educ = (13 - pre_proc_val[["mean"]][["educ"]]) /
pre_proc_val[["std"]][["educ"]]
hispanic = 1
union = 0
exper = (18 - pre_proc_val[["mean"]][["exper"]]) /
pre_proc_val[["std"]][["exper"]]
kidlt6 = 1

# Construir matriz de predição
our_pred = as.matrix(data.frame(husage = husage,
                                husunion = husunion,
                                husearns = husearns,
                                huseduc = huseduc,
                                husblck = husblck,
                                hushisp = hushisp,
                                hushrs = hushrs,
                                kidge6 = kidge6,
                                age = age,
                                black = black,
                                educ = educ,
                                hispanic = hispanic,
                                union = union,
                                exper = exper,
                                kidlt6 = kidlt6))

# Realizar a predição
predict_our_lasso <- predict(lasso_reg,
```

```
                              s = best_lambda_lasso,
                              newx = our_pred)
predict_our_lasso

# Converter o valor padronizado para a escala original
wage_pred_lasso = (predict_our_lasso *
                     pre_proc_val[["std"]][["lwage"]]) +
  pre_proc_val[["mean"]][["lwage"]]

# Aplicar a transformação inversa do logaritmo
exp(wage_pred_lasso)

# O valor representa o salário-hora estimado (US$),
# com base nas características especificadas

# Cálculo do intervalo de confiança da predição
n <- nrow(train)  # tamanho da amostra
m <- wage_pred_lasso  # média estimada
s <- pre_proc_val[["std"]][["lwage"]]  # desvio padrão
dam <- s / sqrt(n)  # erro padrão da média
CIlwr_lasso <- m + (qnorm(0.025)) * dam  # limite inferior
CIupr_lasso <- m - (qnorm(0.025)) * dam  # limite superior

# Converter limites do intervalo de confiança para a escala original
exp(CIlwr_lasso)
exp(CIupr_lasso)

# O salário-hora estimado é de aproximadamente US$ 8.02,
# com intervalo de confiança entre US$ 7.85 e US$ 8.19.
```

Regressão Elastic Net

```
# Carregar pacotes necessários para manipulação, modelagem e visualização de
dados
library(plyr)
library(readr)
library(dplyr)
library(ggplot2)
library(repr)
library(glmnet)
library(caret)
```

```
# Carregar o dataset a partir do arquivo .RData
load("trabalhosalarios.RData")

# Armazenar o dataset carregado no objeto "dat"
dat <- trabalhosalarios

# Exibir o conjunto de dados completo na interface
View(dat)

# Exibir a estrutura e o tipo das variáveis contidas no dataset
glimpse(dat)

# Verificar o uso de memória atual e liberar espaço, se possível
gc()

# Definir a semente para geração de números aleatórios
# A utilização da mesma semente assegura reprodutibilidade nos resultados
# Essa semente controla a amostragem aleatória utilizada na partição dos
dados
set.seed(302)

# Criar um índice de amostragem para dividir o dataset em 80% treino e 20%
teste
index = sample(1:nrow(dat), 0.8 * nrow(dat))

# Criar subconjunto de dados para treinamento
train = dat[index,]

# Criar subconjunto de dados para teste
test = dat[-index,]

# Verificar as dimensões dos conjuntos de treinamento e teste
dim(train)
dim(test)
# O conjunto de treinamento contém 2059 observações e 17 variáveis
# O conjunto de teste contém 359 observações e 17 variáveis

# Definir as variáveis contínuas que serão padronizadas
# Variáveis binárias são excluídas desse processo
cols = c('husage', 'husearns', 'huseduc', 'hushrs',
         'age', 'educ', 'exper', 'lwage')
```

```
# Aplicar centralização e escalonamento (z-score normalization)
pre_proc_val <- preProcess(train[, cols], method = c("center", "scale"))
train[, cols] = predict(pre_proc_val, train[, cols])
test[, cols] = predict(pre_proc_val, test[, cols])

# Exibir o resumo estatístico das variáveis padronizadas em ambos os datasets
summary(train)
summary(test)

############################################################
#                     REGRESSÃO ELASTIC NET                #
############################################################

# Definir as variáveis preditoras e a variável dependente a serem utilizadas
no modelo
cols_reg = c('husage', 'husunion', 'husearns', 'huseduc',
             'husblck', 'hushisp', 'hushrs', 'kidge6',
             'age', 'black', 'educ', 'hispanic',
             'union', 'exper', 'kidlt6', 'lwage')

# Gerar variáveis dummies e organizar o dataset em formato matricial
# O objetivo é estimar o logaritmo do salário-hora da esposa (lwage)
dummies <- dummyVars(lwage ~ husage + husunion + husearns + huseduc +
                       husblck + hushisp + hushrs + kidge6 + age +
                       black + educ + hispanic + union + exper + kidlt6,
                     data = dat[, cols_reg])

# Aplicar a transformação dummy nos conjuntos de treino e teste
train_dummies = predict(dummies, newdata = train[, cols_reg])
test_dummies = predict(dummies, newdata = test[, cols_reg])
print(dim(train_dummies)); print(dim(test_dummies))

# Converter o conjunto de treinamento das variáveis explicativas em matriz
x = as.matrix(train_dummies)

# Extrair o vetor da variável dependente (lwage) do conjunto de treino
y_train = train$lwage

# Converter o conjunto de teste das variáveis explicativas em matriz
x_test = as.matrix(test_dummies)
```

```
# Extrair o vetor da variável dependente (lwage) do conjunto de teste
y_test = test$lwage

# A regressão Elastic Net combina as penalizações L1 (Lasso) e L2 (Ridge),
# promovendo simultaneamente regularização e seleção de variáveis.
# O pacote "caret" permite a otimização automática dos parâmetros alpha e
lambda.

# Configurar o controle de treino utilizando validação cruzada repetida
# Método: 10-fold cross-validation, com 5 repetições e busca aleatória dos
hiperparâmetros
train_cont <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 5,
                           search = "random",
                           verboseIter = TRUE)

# O parâmetro alpha (entre 0 e 1) é ajustado automaticamente:
# - alpha = 0 corresponde à regressão Ridge
# - alpha = 1 corresponde à regressão Lasso
# O parâmetro lambda também é estimado por validação cruzada.

# Treinar o modelo Elastic Net com base nas variáveis definidas
elastic_reg <- train(lwage ~ husage + husunion + husearns + huseduc + husblck
+
                       hushisp + hushrs + kidge6 + age + black + educ +
                       hispanic + union + exper + kidlt6,
                     data = train,
                     method = "glmnet",
                     tuneLength = 10,
                     trControl = train_cont)

# Exibir o melhor conjunto de hiperparâmetros (alpha e lambda) identificados
elastic_reg$bestTune

# Exibir os coeficientes estimados do modelo final
elastic_reg[["finalModel"]][["beta"]]

# Realizar predições e avaliar o desempenho do modelo
```

```
# Gerar predições no conjunto de treinamento
predictions_train <- predict(elastic_reg, x)

# Definir função para cálculo das métricas de desempenho (R² e RMSE)
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE / nrow(df))

  # Retornar métricas de avaliação do modelo
  data.frame(
    RMSE = RMSE,
    Rsquare = R_square
  )
}

# Avaliar o desempenho do modelo no conjunto de treinamento
eval_results(y_train, predictions_train, train)

# Gerar predições no conjunto de teste
predictions_test <- predict(elastic_reg, x_test)

# Avaliar o desempenho do modelo no conjunto de teste
eval_results(y_test, predictions_test, test)

# Realizar predição para um novo conjunto de valores de entrada

# Como as variáveis contínuas foram padronizadas,
# é necessário aplicar o mesmo processo antes da predição.
# As variáveis binárias (dummies) não são padronizadas.

# Definir valores padronizados para o novo caso
husage      =      (40      -      pre_proc_val[["mean"]][["husage"]])      /
pre_proc_val[["std"]][["husage"]]
husunion = 0
husearns     =     (600     -     pre_proc_val[["mean"]][["husearns"]])     /
pre_proc_val[["std"]][["husearns"]]
huseduc      =      (13      -      pre_proc_val[["mean"]][["huseduc"]])      /
pre_proc_val[["std"]][["huseduc"]]
husblck = 1
```

```
hushisp = 0
hushrs      =      (40      -      pre_proc_val[["mean"]][["hushrs"]])      /
pre_proc_val[["std"]][["hushrs"]]
kidge6 = 1
age = (38 - pre_proc_val[["mean"]][["age"]]) / pre_proc_val[["std"]][["age"]]
black = 0
educ       =       (13       -       pre_proc_val[["mean"]][["educ"]])       /
pre_proc_val[["std"]][["educ"]]
hispanic = 1
union = 0
exper      =      (18      -      pre_proc_val[["mean"]][["exper"]])      /
pre_proc_val[["std"]][["exper"]]
kidlt6 = 1

# Construir a matriz de predição com os valores padronizados
our_pred = as.matrix(data.frame(husage = husage,
                                husunion = husunion,
                                husearns = husearns,
                                huseduc = huseduc,
                                husblck = husblck,
                                hushisp = hushisp,
                                hushrs = hushrs,
                                kidge6 = kidge6,
                                age = age,
                                black = black,
                                educ = educ,
                                hispanic = hispanic,
                                union = union,
                                exper = exper,
                                kidlt6 = kidlt6))

# Realizar a predição com base no modelo ajustado
predict_our_elastic <- predict(elastic_reg, our_pred)
predict_our_elastic

# Converter o resultado padronizado para a escala original (despadronização)
wage_pred_elastic = (predict_our_elastic * pre_proc_val[["std"]][["lwage"]])
+
  pre_proc_val[["mean"]][["lwage"]]

# Aplicar a transformação inversa do logaritmo
```

```
exp(wage_pred_elastic)

# O valor resultante representa o salário-hora estimado da esposa,
# em dólares, com base nas características informadas.

# Calcular intervalo de confiança aproximado para a estimativa
n <- nrow(train)
m <- wage_pred_elastic
s <- pre_proc_val[["std"]][["lwage"]]
dam <- s / sqrt(n)
CIlwr_elastic <- m + (qnorm(0.025)) * dam
CIupr_elastic <- m - (qnorm(0.025)) * dam

# Converter os limites do intervalo de confiança para a escala original
exp(CIlwr_elastic)
exp(CIupr_elastic)

# A predição indica um salário-hora médio estimado de US$ 8.77,
# com intervalo de confiança entre aproximadamente US$ 8.59 e US$ 8.97.
```

TABELA 3 – DESEMPENHO DOS MODELOS DE REGRESSÃO RIDGE, LASSO E ELASTICNET

Modelo	RMSE	R2	CIlwr (Intervalo Inferior)	CIupr (Intervalo Superior)	Predição var. “lwage” (salárioshora esposa)
Ridge Time:1.37s Lambda:0.0316	Treino: 0.8411446 Teste: 0.9893328	Treino: 0.292132 Teste: 0.259084	8.493945	8.87351	8.681654
Lasso Time:1.55s Lambda:0.01	Treino: 0.8420298 Teste: 0.9894877	Treino: 0.2906413 Teste: 0.258852	7.84636	8.196938	8.01971
ElasticNet Time:7.91s Lambda:0.0125	Treino: 0.8410462 Teste: 0.9894186	Treino: 0.2922976 Teste: 0.2589555	8.587557	8.971305	8.777334

FONTE: De autoria própria (2024)

APÊNDICE F - ARQUITETURA DE DADOS

A – ENUNCIADO

1 Construção de Características: Identificador automático de idioma

O problema consiste em criar um modelo de reconhecimento de padrões que dado um texto de entrada, o programa consegue classificar o texto e indicar a língua em que o texto foi escrito.

Parta do exemplo (notebook produzido no Colab) que foi disponibilidade e crie as funções para calcular as diferentes características para o problema da identificação da língua do texto de entrada.

Nessa atividade é para "construir características".

Meta: a acurácia deverá ser maior ou igual a 70%.

Essa tarefa pode ser feita no Colab (Google) ou no Jupiter, em que deverá exportar o notebook e imprimir o notebook para o formato PDF. Envie no UFPR Virtual os dois arquivos.

2 Melhore uma base de dados ruim

Escolha uma base de dados pública para problemas de classificação, disponível ou com origem na UCI Machine Learning.

Use o mínimo de intervenção para rodar a SVM e obtenha a matriz de confusão dessa base.

O trabalho começa aqui, escolha as diferentes tarefas discutidas ao longo da disciplina, para melhorar essa base de dados, até que consiga efetivamente melhorar o resultado.

Considerando a acurácia para bases de dados balanceadas ou quase balanceadas, se o percentual da acurácia original estiver em até 85%, a meta será obter 5%. Para bases com mais de 90% de acurácia, a meta será obter a melhora em pelo menos 2 pontos percentuais (92% ou mais).

Nessa atividade deverá ser entregue o script aplicado (o notebook e o PDF correspondente).

B – RESOLUÇÃO

```
import random
import re
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
       #
       # amostras de texto em diferentes línguas
       #
       ingles = [
       "Hello, how are you?",
       "I love to read books.",
       "The weather is nice today.",
       "Where is the nearest restaurant?",
       "What time is it?",
       "I enjoy playing soccer.",
       "Can you help me with this?",
       "I'm going to the movies tonight.",
       "This is a beautiful place.",
       "I like listening to music.",
       "Do you speak English?",
       "What is your favorite color?",
       "I'm learning to play the guitar.",
       "Have a great day!",
       "I need to buy some groceries.",
       "Let's go for a walk.",
       "How was your weekend?",
       "I'm excited for the concert.",
       "Could you pass me the salt, please?",
       "I have a meeting at 2 PM.",
       "I'm planning a vacation.",
       "She sings beautifully.",
       "The cat is sleeping.",
       "I want to learn French.",
       "I enjoy going to the beach.",
       "Where can I find a taxi?",
       "I'm sorry for the inconvenience.",
```

```
"I'm studying for my exams.",
"I like to cook dinner at home.",
"Do you have any recommendations for restaurants?",
]
espanhol = [
"Hola, ¿cómo estás?",
"Me encanta leer libros.",
"El clima está agradable hoy.",
"¿Dónde está el restaurante más cercano?",
"¿Qué hora es?",
"Voy al parque todos los días.",
"¿Puedes ayudarme con esto?",
"Me gustaría ir de vacaciones.",
"Este es mi libro favorito.",
"Me gusta bailar salsa.",
"¿Hablas español?",
"¿Cuál es tu comida favorita?",
"Estoy aprendiendo a tocar el piano.",
"¡Que tengas un buen día!",
"Necesito comprar algunas frutas.",
"Vamos a dar un paseo.",
"¿Cómo estuvo tu fin de semana?",
"Estoy emocionado por el concierto.",
"¿Me pasas la sal, por favor?",
"Tengo una reunión a las 2 PM.",
"Estoy planeando unas vacaciones.",
"Ella canta hermosamente.",
"El perro está jugando.",
"Quiero aprender italiano.",
"Disfruto ir a la playa.",
"¿Dónde puedo encontrar un taxi?",
"Lamento las molestias.",
"Estoy estudiando para mis exámenes.",
"Me gusta cocinar la cena en casa.",
"¿Tienes alguna recomendación de restaurantes?",
]
portugues = [
"Estou indo para o trabalho agora.",
"Adoro passar tempo com minha família.",
"Preciso comprar leite e pão.",
"Vamos ao cinema no sábado.",
```

```
        "Gosto de praticar esportes ao ar livre.",
        "O trânsito está terrível hoje.",
        "A comida estava deliciosa!",
        "Você já visitou o Rio de Janeiro?",
        "Tenho uma reunião importante amanhã.",
        "A festa começa às 20h.",
        "Estou cansado depois de um longo dia de trabalho.",
        "Vamos fazer um churrasco no final de semana.",
        "O livro que estou lendo é muito interessante.",
        "Estou aprendendo a cozinhar pratos novos.",
        "Preciso fazer exercícios físicos regularmente.",
        "Vou viajar para o exterior nas férias.",
        "Você gosta de dançar?",
        "Hoje é meu aniversário!",
        "Gosto de ouvir música clássica.",
        "Estou estudando para o vestibular.",
        "Meu time de futebol favorito ganhou o jogo.",
        "Quero aprender a tocar violão.",
        "Vamos fazer uma viagem de carro.",
        "O parque fica cheio aos finais de semana.",
        "O filme que assisti ontem foi ótimo.",
        "Preciso resolver esse problema o mais rápido possível.",
        "Adoro explorar novos lugares.",
        "Vou visitar meus avós no domingo.",
        "Estou ansioso para as férias de verão.",
        "Gosto de fazer caminhadas na natureza.",
        "O restaurante tem uma vista incrível.",
        "Vamos sair para jantar no sábado.",
        ]

pre_padroes = []
for frase in ingles:
pre_padroes.append( [frase, 'inglês'])
for frase in espanhol:
pre_padroes.append( [frase, 'espanhol'])
for frase in portugues:
pre_padroes.append( [frase, 'português'])
random.shuffle(pre_padroes)
print(pre_padroes
import pandas as pd
dados = pd.DataFrame(pre_padroes)
```

```
dados
# a entrada é o vetor pre_padroes e a saída desse passo deverá ser "padrões"
import re
import pandas as pd
from collections import Counter
# Função para calcular a frequência de bigramas em uma frase
def frequenciaBigramas(texto):
palavras = re.split(r'\s+', texto)
bigramas = [palavras[i] + ' ' + palavras[i+1] for i in range(len(palavras)-
1) if len(palavras[i]) > 0 and len(palavras[i+1]) > 0]
contagem_bigramas = Counter(bigramas)
total_bigramas = sum(contagem_bigramas.values())
return {bg: count / total_bigramas for bg, count in
contagem_bigramas.items()}
# Função para calcular a frequência de trigramas em uma frase
def frequenciaTrigramas(texto):
palavras = re.split(r'\s+', texto)
trigramas = [palavras[i] + ' ' + palavras[i+1] + ' ' + palavras[i+2] for i
in range(len(palavras)-2) if len(palavras[i]) > 0 and l
contagem_trigramas = Counter(trigramas)
total_trigramas = sum(contagem_trigramas.values())
return {tg: count / total_trigramas for tg, count in
contagem_trigramas.items()}
# Função para calcular a frequência de palavras comuns em inglês, português
e espanhol
def frequenciaPalavrasComuns(texto):
palavras = re.split(r'\s+', texto.lower())
palavras_comuns_pt = set([
'de', 'que', 'não', 'é', 'em', 'o', 'a', 'os', 'as', 'um', 'uma', 'uns',
'umas', 'e', 'ou', 'mas', 'se',
'por', 'com', 'para', 'como', 'no', 'na', 'nos', 'nas', 'ao', 'aos', 'à',
'às', 'do', 'da', 'dos', 'das',
'num', 'numa', 'num', 'nuns', 'numas', 'até', 'então', 'sobre', 'pelos',
'pelas', 'pelo', 'pela'
])
palavras_comuns_es = set([
'de', 'que', 'no', 'es', 'en', 'el', 'la', 'los', 'las', 'un', 'una', 'unos',
'unas', 'y', 'o', 'pero', 'si',
'por', 'con', 'para', 'como', 'del', 'al', 'a', 'ante', 'bajo', 'cabe',
'contra', 'desde', 'durante', 'entre',
```

```
'hacia', 'hasta', 'mediante', 'según', 'sin', 'so', 'sobre', 'tras',
'versus', 'via'
])
palavras_comuns_en = set([
'the', 'is', 'in', 'and', 'it', 'a', 'an', 'of', 'to', 'for', 'on', 'with',
'as', 'by', 'at', 'from', 'that',
'this', 'these', 'those', 'be', 'been', 'being', 'am', 'are', 'was', 'were',
'will', 'would', 'can', 'could',
'shall', 'should', 'may', 'might', 'must', 'do', 'does', 'did', 'done',
'doing', 'has', 'have', 'had', 'having'
])
contagem_pt = sum(1 for palavra in palavras if palavra in palavras_comuns_pt)
contagem_es = sum(1 for palavra in palavras if palavra in palavras_comuns_es)
contagem_en = sum(1 for palavra in palavras if palavra in palavras_comuns_en)
total_palavras = len(palavras) if len(palavras) > 0 else 1
return contagem_pt / total_palavras, contagem_es / total_palavras,
contagem_en / total_palavras
def extraiCaracteristicas(frase):
# frase é um vetor [ 'texto', 'lingua' ]
texto = frase[0]
pattern_regex = re.compile(r'[^\w\s]', re.UNICODE)
texto = re.sub(pattern_regex, ' ', texto)
bigramas = frequenciaBigramas(texto)
trigramas = frequenciaTrigramas(texto)
stopwords_pt, stopwords_es, stopwords_en = frequenciaPalavrasComuns(texto)
# Características nomeadas
caracteristica1 = sum(bigramas.values()) / len(bigramas) if bigramas else 0
caracteristica2 = sum(trigramas.values()) / len(trigramas) if trigramas else
0
caracteristica3 = stopwords_pt
caracteristica4 = stopwords_es
caracteristica5 = stopwords_en
padrao = [caracteristica1, caracteristica2, caracteristica3,
caracteristica4, caracteristica5, frase[1]]
return padrao
def geraPadroes(frases):
padroes = []
for frase in frases:
padrao = extraiCaracteristicas(frase)
padroes.append(padrao)
return padroes
```

```
def geraPadroes(frases):
padroes = []
for frase in frases:
padrao = extraiCaracteristicas(frase)
padroes.append(padrao)
return padroes
#
# apenas para visualizacao
print(padroes)
dados = pd.DataFrame(padroes)
from sklearn.model_selection import train_test_split
import numpy as np
#from sklearn.metrics import confusion_matrix
vet = np.array(padroes)
classes = vet[:,-1] # classes = [p[-1] for p in padroes]
#print(classes)
padroes_sem_classe = vet[:,0:-1]
#print(padroes_sem_classe)
X_train, X_test, y_train, y_test = train_test_split(padroes_sem_classe,
classes, test_size=0.25, stratify=classes)
Com os conjuntos separados, podemos "treinar" o modelo usando a SVM.
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
treinador = svm.SVC() #algoritmo escolhido
modelo = treinador.fit(X_train, y_train)
#
# score com os dados de treinamento
acuracia = modelo.score(X_train, y_train)
print("Acurácia nos dados de treinamento: {:.2f}%".format(acuracia * 100))
#
# melhor avaliar com a matriz de confusão
y_pred = modelo.predict(X_train)
cm = confusion_matrix(y_train, y_pred)
print(cm)
print(classification_report(y_train, y_pred))
#
# com dados de teste que não foram usados no treinamento
print('métricas mais confiáveis')
y_pred2 = modelo.predict(X_test)
cm = confusion_matrix(y_test, y_pred2)
```

```
print(cm)
print(classification_report(y_test, y_pred2))
def identifica_idioma(texto):
"""
Identifica o idioma de um texto dado. Se o texto tiver mais de 500
caracteres, será truncado.
"""
if len(texto) > 500:
texto = texto[:500]
# Extrai características do texto
caracteristicas = extraiCaracteristicas([texto, 'desconhecido'])[:-1] #
Remove a classe ('desconhecido')
# Prediz o idioma usando o modelo treinado
predicao = modelo.predict([caracteristicas])[0]
return predicao
# Solicitar ao usuário a entrada de um texto
texto_usuario = input("Por favor, insira um texto (máximo de 500
caracteres): ")
# Identificar o idioma do texto
idioma_identificado = identifica_idioma(texto_usuario)
# Exibir o resultado
if idioma_identificado in ['inglês', 'espanhol', 'português']:
print(f"O seu texto está escrito no idioma {idioma_identificado}.")
else:
print("Não foi possível identificar o idioma.")
def avalia acuracia identificacao(textos idiomas reais):

Acurácia nos dados de treinamento: 88.41%
[[19  0  3]
 [ 2 21  0]
 [ 3  0 21]]
              precision    recall  f1-score   support

    espanhol       0.79      0.86      0.83        22
      inglês       1.00      0.91      0.95        23
   português       0.88      0.88      0.88        24

    accuracy                           0.88        69
   macro avg       0.89      0.88      0.89        69
weighted avg       0.89      0.88      0.89        69
```

```
métricas mais confiáveis
[[8 0 0]
 [0 6 1]
 [2 0 6]]
              precision    recall  f1-score   support

    espanhol       0.80      1.00      0.89         8
      inglês       1.00      0.86      0.92         7
   português       0.86      0.75      0.80         8

    accuracy                           0.87        23
   macro avg       0.89      0.87      0.87        23
weighted avg       0.88      0.87      0.87        23
```

Questão 2

```
Acurácia original: 62.93% Acurácia melhorada: 79.72% A diferença após o
tratamento do dataset foi de 16.79%
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
# Biblioteca Matplotlib - Criação de gráficos
import matplotlib.pyplot as plt
# Carregando os dados do Titanic usando Seaborn
titanic = sns.load_dataset('titanic')
# Listando o nome das colunas
titanic.columns
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
 'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
 'alive', 'alone'],
 dtype='object')
survived pclass sex age sibsp parch fare embarked class who adult_male deck
embark_town alive alone
0 0 3 male 22.0 1 0 7.2500 S Third man True NaN Southampton no False
1 1 1 female 38.0 1 0 71.2833 C First woman False C Cherbourg yes False
2 1 3 female 26.0 0 0 7.9250 S Third woman False NaN Southampton yes True
3 1 1 female 35.0 1 0 53.1000 S First woman False C Southampton yes False
4 0 3 male 35.0 0 0 8.0500 S Third man True NaN Southampton no True
```

```
# Imprimindo as 5 primeiras linhas
titanic.head()
# Observando número de linhas e colunas
titanic.shape
(891, 15)
# Imprime o tipo de dado em cada coluna.
titanic.dtypes
survived int64
pclass int64
sex object
age float64
sibsp int64
parch int64
fare float64
embarked object
class category
who object
adult_male bool
deck category
embark_town object
alive object
alone bool
dtype: object
# 1) Itera sobre todas as colunas não numéricas e aplica o LabelEncoder
(transformar variáveis categóricas em valores numéricos)
for column in titanic.select_dtypes(include=['object', 'bool',
'category']).columns:
label_encoder = LabelEncoder()
titanic[column] = label_encoder.fit_transform(titanic[column])
# Imprimindo as 5 primeiras linhas
titanic.head()
survived pclass sex age sibsp parch fare embarked class who adult_male
0 0 3 1 22.0 1 0 7.2500 2 2 1 1
1 1 1 0 38.0 1 0 71.2833 0 0 2 0
2 1 3 0 26.0 0 0 7.9250 2 2 2 0
3 1 1 0 35.0 1 0 53.1000 2 0 2 0
4 0 3 1 35.0 0 0 8.0500 2 2 1 1
# Imprime o tipo de dado em cada coluna.
titanic.dtypes
survived int64
pclass int64
```

```
sex int64
age float64
sibsp int64
parch int64
fare float64
embarked int64
class int64
who int64
adult_male int64
deck int64
embark_town int64
alive int64
alone int64
dtype: object
# 2) Remover linhas com valores ausentes do DataFrame titanic
titanic.dropna(inplace=True)
# Depois das correções, verificando se ainda há valores faltantes
titanic.isna().sum()
survived 0
pclass 0
sex 0
age 0
sibsp 0
parch 0
fare 0
embarked 0
class 0
who 0
adult_male 0
deck 0
embark_town 0
alive 0
alone 0
dtype: int64
# Selecionar as features (X) e o target (y)
X = titanic.drop('survived', axis=1)
y = titanic['survived']
pclass sex age sibsp parch fare embarked class who adult_male deck em
0 3 1 22.0 1 0 7.2500 2 2 1 1 7
1 1 0 38.0 1 0 71.2833 0 0 2 0 2
2 3 0 26.0 0 0 7.9250 2 2 2 0 7
```

```
3 1 0 35.0 1 0 53.1000 2 0 2 0 2
4 3 1 35.0 0 0 8.0500 2 2 1 1 7
X.head()
y.head()
0 0
1 1
2 1
3 1
4 0
Name: survived, dtype: int64
# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Inicializar e treinar o classificador SVM
svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)
# Fazer previsões no conjunto de teste
y_pred = svm_classifier.predict(X_test)
# Calcular a acurácia do modelo em porcentagem
accuracy_percentage = accuracy_score(y_test, y_pred) * 100
print("Acurácia do modelo:", accuracy_percentage, "%")
Acurácia do modelo: 62.93706293706294 %
# Verificando a prevalência de sobreviventes
# A porcentagem resultante representa a prevalência de sobreviventes na
base de dados do Titanic.
# Com 40.62% entendemos que os dados estão razoavelmente balanceados.
prevalence = titanic['survived'].mean() * 100 # Calculando a média da
coluna 'survived' e multiplicando por 100 para obter a porcentage
print("Prevalência de sobreviventes: {:.2f}%".format(prevalence))
Prevalência de sobreviventes: 40.62%
# 3) Criar uma nova coluna 'FamilySize' representando o tamanho da família
titanic['FamilySize'] = titanic['sibsp'] + titanic['parch'] + 1
# 4) Agrupamento de idade
idade = [0, 12, 18, 30, 50, 200]
labels = ['Child', 'Teenager', 'Young Adult', 'Adult', 'Senior']
titanic['AgeGroup'] = pd.cut(titanic['age'], bins=idade, labels=labels)
titanic['AgeGroup'] = LabelEncoder().fit_transform(titanic['AgeGroup'])
survived pclass sex age sibsp parch fare embarked class who adult_male
0 0 3 1 22.0 1 0 7.2500 2 2 1 1
1 1 1 0 38.0 1 0 71.2833 0 0 2 0
2 1 3 0 26.0 0 0 7.9250 2 2 2 0
```

```
3 1 1 0 35.0 1 0 53.1000 2 0 2 0
4 0 3 1 35.0 0 0 8.0500 2 2 1 1
# Imprimindo as 5 primeiras linhas
titanic.head()
# 5) Manter apenas as colunas relevantes
#titanic = titanic[['survived', 'pclass', 'sex', 'embarked', 'FamilySize',
'AgeGroup' ]]
# Manter apenas as colunas relevantes
titanic = titanic[['survived', 'pclass', 'sex', 'FamilySize','AgeGroup' ]]
survived pclass sex FamilySize AgeGroup
0 0 3 1 2 4
1 1 1 0 2 0
2 1 3 0 1 4
3 1 1 0 2 0
4 0 3 1 1 0
# Imprimindo as 5 primeiras linhas
titanic.head()
# Selecionar as features (X) e o target (y)
X = titanic.drop('survived', axis=1)
y = titanic['survived']
X.head()
pclass sex FamilySize AgeGroup
0 3 1 2 4
1 1 0 2 0
2 3 0 1 4
3 1 0 2 0
4 3 1 1 0
y.head()
0 0
1 1
2 1
3 1
4 0
Name: survived, dtype: int64
# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Inicializar e treinar o classificador SVM
svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)
# Fazer previsões no conjunto de teste
```

```
y_pred = svm_classifier.predict(X_test)
# Calcular a acurácia do modelo em porcentagem
accuracy_percentage = accuracy_score(y_test, y_pred) * 100
print("Acurácia do modelo:", accuracy_percentage, "%")
Acurácia do modelo: 79.72027972027972 %
# Calcular a matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)
print("Matriz de Confusão:")
print(conf_matrix)
Matriz de Confusão:
[[75 12]
 [17 39]]
```

APÊNDICE G - APRENDIZADO DE MÁQUINA

A – ENUNCIADO

Para cada uma das tarefas abaixo (Classificação, Regressão etc.) e cada base de dados (Veículo, Diabetes etc.), fazer os experimentos com todas as técnicas solicitadas (KNN, RNA etc.) e preencher os quadros com as estatísticas solicitadas, bem como os resultados pedidos em cada experimento.

B – RESOLUÇÃO

Questão 1

Veículo

TABELA 4 – MELHOR TÉCNICA SVM – HOLD-OUT VEICULO

Técnica	Parâmetro	Acurácia	Matriz de Confusão
SVM – Hold-out	C=1 Sigma=0.06	0.784	Prediction,bus,opel,saab,van bus,41,2,1,0 opel,0,20,9,0 saab,0,19,32,1 van,2,1,1,38

FONTE: De autoria própria (2024)

Diabetes

TABELA 5 – MELHOR TÉCNICA RF – CV DIABETES

Técnica	Parâmetro	Acurácia	Matriz de Confusão
RF – CV	mtry=2	0.7451	Prediction,neg,pos neg,82,21 pos,18,32

FONTE: De autoria própria (2024)

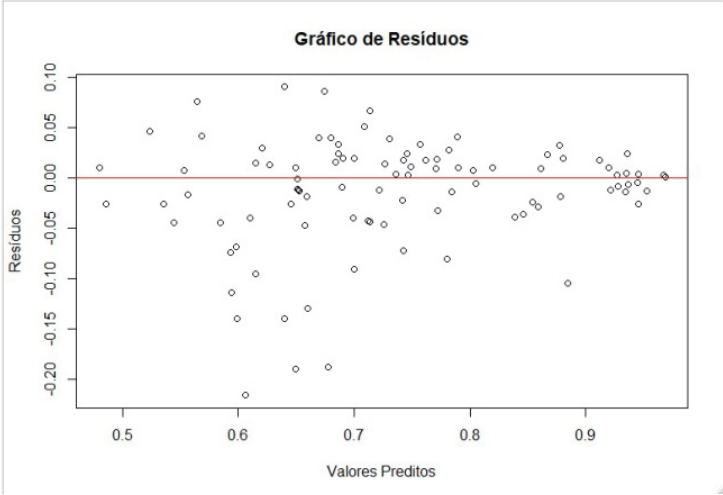
Admissão

TABELA 6 – MELHOR TÉCNICA SVM – HOLDOUT ADMISSÃO

Técnica	Parâmetro	R2	Syx	Pearson	Rmse	MAE
SVM– Holdout	C=1 Sigma=0.11	0.8492141	0.0568319 4	0.928519 6	0.0565412 4	0.0380139 9

FONTE: De autoria própria (2024)

FIGURA 7 – RESÍDUOS SVM – HOLDOUT ADMISSÃO



FONTE: De autoria própria (2024)

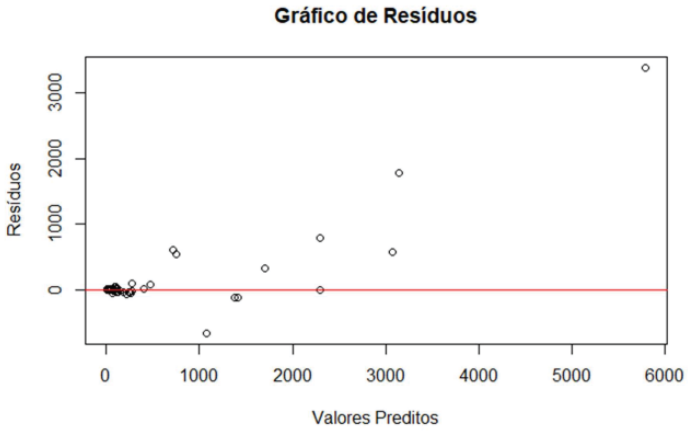
Biomassa

TABELA 8 – MELHOR TÉCNICA SVM - HOLDOUT

Técnica	Parâmetro	R2	Syx	Pearson	Rmse	MAE
RF – Holdout	mtry=2	0.8709691	532.4229	0.9797789	523.474	157.3707

FONTE: De autoria própria (2024)

FIGURA 9 – RESÍDUOS SVM – HOLDOUT BIOMASSA

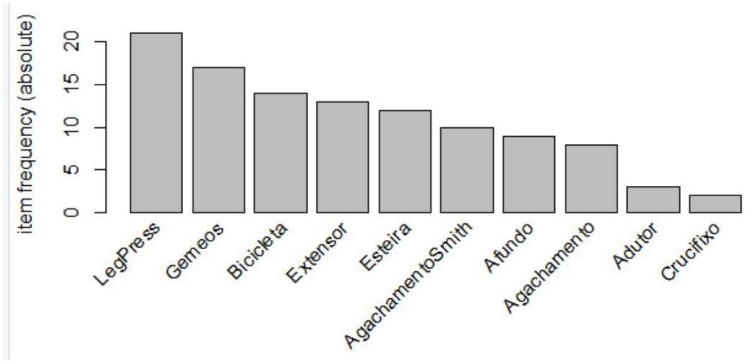


FONTE: De autoria própria (2024)

```
### Pacotes necessários
install.packages("mlbench")
library(mlbench)
install.packages("mice")
library(mice)
### para o kmodes
install.packages("kLar")
library(kLar)
### Leitura dos dados
setwd("c:\\faa\\Trabalho")
dados <- read.csv("4 - Veiculos - Dados.csv")
view(dados)
# Remoção da coluna desnecessária (id)
dados$id <- NULL
## Executa o cluster - Usar 10 Clusters
set.seed(202470)
cluster.results <- kmodes(dados, 10, weighted = FALSE)
cluster.results
### Resultado do agrupamento
resultado <- cbind(dados, cluster.results$cluster)
### Exibição do resultado
head(resultado, 10)
### Instalação dos pacotes necessários
# install.packages("arules", dep=TRUE)
library(arules)
library(datasets)
```

```
### Leitura dos dados
setwd("C:\\laa\\Trabalho")
dados <- read.transactions(file="2 - Musculacao - Dados.csv",
format="basket", sep=";")
# Exibição dos 5 primeiros itens
inspect(dados[1:5])
# Gráfico dos 10 primeiros itens mais frequentes
itemFrequencyPlot(dados, topN=10, type="absolute")
# Visão geral dos dados
summary(dados)
### Extração das regras
# Testes efetuados com suporte entre 0,1 a 0,5 e confiança entre 0,5 e 0,9
# A escolha final foi suporte = 0,3 e confiança = 0,7
# Foram geradas 18 regras
set.seed(202470)
rules <- apriori(dados, parameter = list(supp=0.3, conf=0.7,
target="rules")) # > 18 regras
# Visão geral das regras
summary(rules)
# Exibição das regras por ordem de confiança
inspect(sort(rules, by="confidence"))
```

FIGURA 8 – FREQUENCIA DE ITENS



FONTE: De autoria própria (2024)

APÊNDICE H - DEEP LEARNING

A – ENUNCIADO

1 Classificação de Imagens (CNN)

Implementar o exemplo de classificação de objetos usando a base de dados CIFAR10 e a arquitetura CNN vista no curso.

2 Detector de SPAM (RNN)

Implementar o detector de spam visto em sala, usando a base de dados SMS Spam e arquitetura de RNN vista no curso.

3 Gerador de Dígitos Fake (GAN)

Implementar o gerador de dígitos *fake* usando a base de dados MNIST e arquitetura GAN vista no curso.

4 Tradutor de Textos (Transformer)

Implementar o tradutor de texto do português para o inglês, usando a base de dados e a arquitetura Transformer vista no curso.

B – RESOLUÇÃO

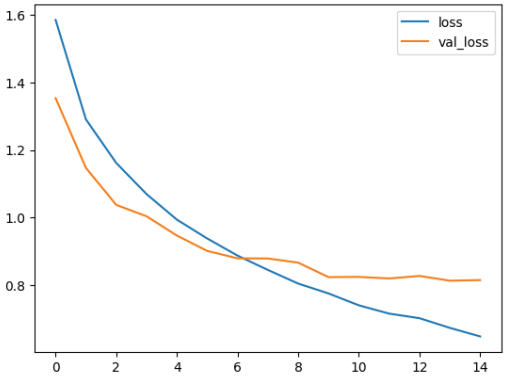
Questão 1

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
# Carga da base
cifar10 = tf.keras.datasets.cifar10
# Já está separado em dados de treino e teste
# Não precisa separar
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# Imagens em pixels de 0 - 255
```

```
# / 255.0 transforma em 0 - 1
x_train, x_test = x_train / 255.0, x_test / 255.0
# O dado y é a classe a qual faz parte
# O flattem torna os dados vetorizados
y_train, y_test = y_train.flatten(), y_test.flatten()
# Dimensão dos dados
print("x_train.shape: ", x_train.shape)
print("y_train.shape: ", y_train.shape)
print("x_test.shape: ", x_test.shape)
print("y_test.shape: ", y_test.shape)
K = len(set(y_train))
# Aqui começa o Estágio 1
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), strides=2, activation="relu")(i)
x = Conv2D(64, (3, 3), strides=2, activation="relu")(x)
x = Conv2D(128, (3, 3), strides=2, activation="relu")(x)
# Todas as imagens são do mesmo tamanho, não precisa de Global Pooling
x = Flatten()(x)
# Aqui começa o Estágio 2
x = Dropout(0.5)(x)
x = Dense(1024, activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(K, activation="softmax")(x)
# Model ( lista entrada, lista saída)
model = Model(i, x)
# Relatório sobre a arquitetura da rede
model.summary()

# Compilar o modelo
model.compile(optimizer="adam",
loss="sparse_categorical_crossentropy", metrics=["accuracy"])
# Treinar o modelo
r = model.fit(x_train, y_train, validation_data=(x_test, y_test),
epochs=15)

# Plotar a função de perda, treino e validação
plt.plot(r.history["loss"], label="loss")
plt.plot(r.history["val_loss"], label="val_loss")
plt.legend()
plt.show()
# Plotar acurácia, treino e validação
```

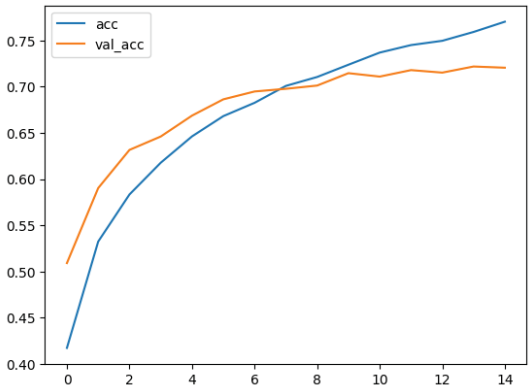
```
plt.plot(r.history["accuracy"], label="acc")
plt.plot(r.history["val_accuracy"], label="val_acc")
plt.legend()
plt.show()
```

FIGURA 9 – FUNÇÃO DE PERDA, TREINO E VALIDAÇÃO



FONTE: De autoria própria (2024)

FIGURA 10 –ACURÁCIA, TREINO E VALIDAÇÃO

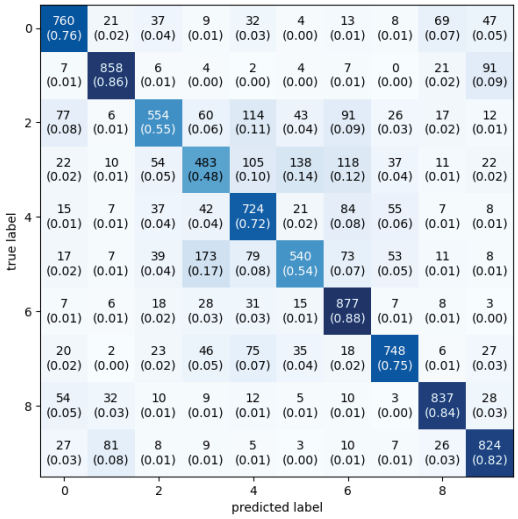


FONTE: De autoria própria (2024)

```
# Efetuar predições na base de teste
# argmax é usado pois a função de ativação da saída é softmax
# argmax pega o neurônio que deu o maior resultado, isto é,
# a maior probabilidade de saída
y_pred = model.predict(x_test).argmax(axis=1)
# Mostrar a matriz de confusão
cm = confusion_matrix(y_test, y_pred)
```

```
plot_confusion_matrix(conf_mat=cm, figsize=(7, 7),
show_normed=True)
```

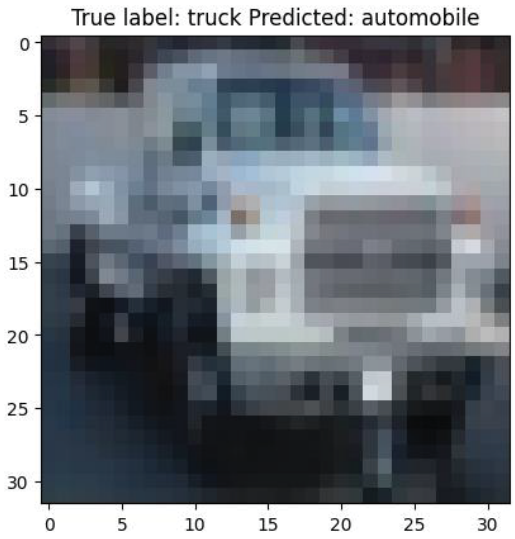
FIGURA 11 – MATRIZ DE CONFUSÃO DE CLASSIFICAÇÃO DE IMAGEM



FONTE: De autoria própria (2024)

```
# Efetuar predições na base de teste
# argmax é usado pois a função de ativação da saída é softmax
# argmax pega o neurônio que deu o maior resultado, isto é,
# a maior probabilidade de saída
y_pred = model.predict(x_test).argmax(axis=1)
# Mostrar a matriz de confusão
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(conf_mat=cm, figsize=(7, 7),
show_normed=True)
Text(0.5, 1.0, 'True label: truck Predicted: automobile')
```

FIGURA 12 – PREDIÇÃO DE AUTOMÓVEL



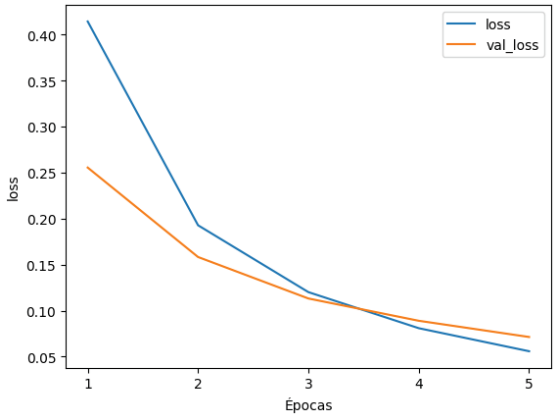
FONTE: De autoria própria (2024)

Questão 2

```
# Importação das Bibliotecas
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense
from tensorflow.keras.layers import GlobalMaxPooling1D
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
# carrega e arruma a base
!wget http://www.razer.net.br/datasets/spam.csv
df = pd.read_csv("spam.csv", encoding="ISO-8859-1")
df.head()
df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
df.columns = ["labels", "data"]
df["b_labels"] = df["labels"].map({ "ham": 0, "spam": 1})
y = df["b_labels"].values
# Número máximo de palavras para considerar
# São consideradas as mais frequentes, as demais são
# ignoradas
num_words = 20000
tokenizer = Tokenizer(num_words=num_words)
```

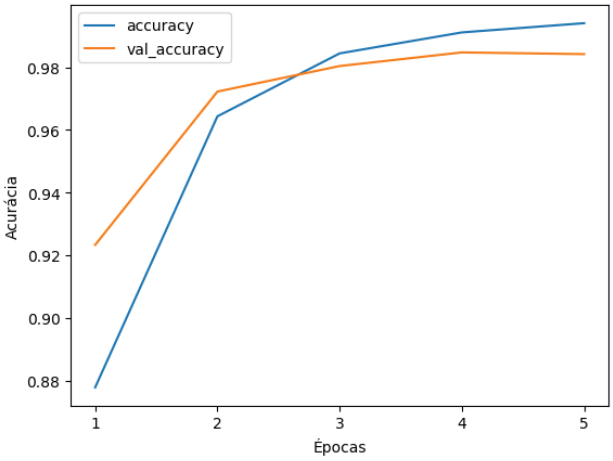
```
tokenizer.fit_on_texts(x_train)
sequences_train = tokenizer.texts_to_sequences(x_train)
sequences_test = tokenizer.texts_to_sequences(x_test)
word2index = tokenizer.word_index
V = len(word2index)
print("%s tokens" % V)
# Acerta o tamanho das sequências (padding)
data_train = pad_sequences(sequences_train) # usa o tamanho da maior seq.
T = data_train.shape[1] # tamanho da sequência
data_test = pad_sequences(sequences_test, maxlen=T)
print("data_train.shape: ", data_train.shape)
print("data_test.shape: ", data_test.shape)
# Define o modelo
D = 20 # tamanho do embedding, hiperparâmetro que pode ser escolhido
M = 5 # tamanho do hidden state, quantidade de unidades LSTM
i = Input(shape=(T,)) # Entra uma frase inteira
x = Embedding(V+1, D)(i)
x = LSTM(M)(x)
x = Dense(1, activation="sigmoid")(x) # Sigmoide pois só tem 2 valores
model = Model(i, x)
model.compile(loss="binary_crossentropy", optimizer="adam",
metrics=["accuracy"])
epochs = 5
r        =        model.fit(data_train,        y_train,        epochs=epochs,
validation_data=(data_test,
y_test))
plt.plot(r.history["loss"], label="loss")
plt.plot(r.history["val_loss"], label="val_loss")
plt.xlabel("Épocas")
plt.ylabel("loss")
plt.xticks(np.arange(0, epochs, step=1), labels=range(1, epochs+1))
plt.legend()
plt.show()
plt.plot(r.history["accuracy"], label="accuracy")
plt.plot(r.history["val_accuracy"], label="val_accuracy")
plt.xlabel("Épocas")
plt.ylabel("Acurácia")
plt.xticks(np.arange(0, epochs, step=1), labels=range(1, epochs+1))
plt.legend()
plt.show()
```

FIGURA 13 – FUNÇÃO DE PERDA DE DETECTOR DE SPAM



FONTE: De autoria própria (2024)

FIGURA 14 – ACURÁCIA DE DETECTOR DE SPAM



FONTE: De autoria própria (2024)

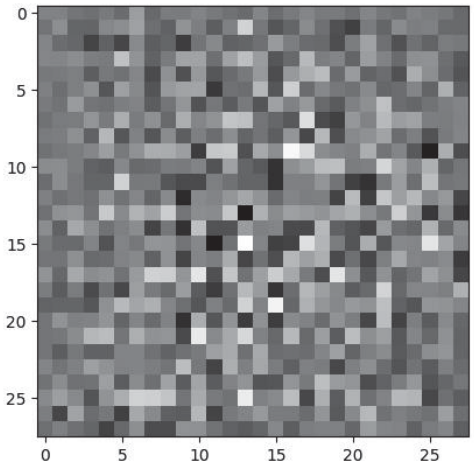
```
# Efetua a predição de um texto novo
texto = "Hi, my name is Razer and want to tell you something."
#texto = "Is your car dirty? Discover our new product. Free for all. Click
the link."
seq_texto = tokenizer.texts_to_sequences([texto]) # Tokeniza
data_texto = pad_sequences(seq_texto, maxlen=T) # Padding
pred = model.predict(data_texto) # Predição
print(pred)
print ("SPAM" if pred >= 0.5 else "OK")
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
[[0.01949643]]
OK
```

Questão 3

```
# Para Gerar os GIFs
!pip install imageio
!pip install git+https://github.com/tensorflow/docs
# Importações
import tensorflow as tf
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time
from IPython import display
# Carregar a base de dados
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
# Normalização
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5 # Normaliza entre [-1, 1]
# Gera o banco em partes e randomiza
BUFFER_SIZE = 60000
BATCH_SIZE = 256
# Cria o dataset (from_tensor_slices)
# Randomiza (shuffle)
# Combina elementos consecutivos em lotes (batch)
train_dataset=tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
```

```
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Conv2DTranspose(64,    (5,    5),    strides=(2,    2),
padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Conv2DTranspose(1,    (5,    5),    strides=(2,    2),
padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)
    return model
# Exemplo de uso da função
model = make_generator_model()
# Teste do GERADOR, ainda não treinado
generator = make_generator_model()
noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

FIGURA 15 – GERADOR DE DIGITOS SEM TREINAMENTO



FONTE: De autoria própria (2024)

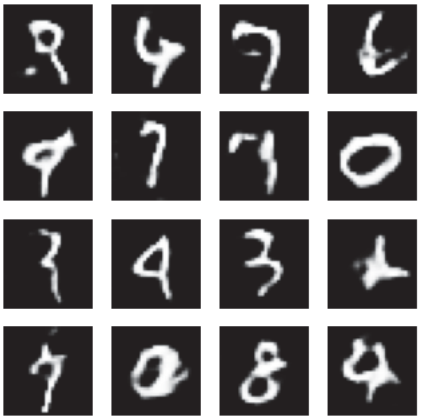
```
# Cria o DISCRIMINADOR
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))
```

```
    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))
    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model

#Teste do DISCRIMINADOR, ainda não treinado
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)

# Define as funções de perda
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

# Perda do DISCRIMINADOR
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

# Perda do GERADOR
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
# Cria os otimizadores para o gerador e discriminador
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
# Cria checkpoints para salvar modelos ao longo do tempo
# Úteis em tarefas longas, para se recuperar de um desligamento
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
discriminator_optimizer=discriminator_optimizer,
generator=generator,
discriminator=discriminator)
# Configura o Loop de treinamento
EPOCHS = 100
noise_dim = 100
num_examples_to_generate = 16
```

```
# You will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])
# Função que faz um passo de treinamento
# É uma `tf.function`, que compila essa função
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

        gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables)
        gradients_of_discriminator = disc_tape.gradient(disc_loss,
discriminator.trainable_variables)

        generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))

discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))
# Treinamento completo/laço
def train(dataset, epochs):
    for epoch in range(epochs):
        start = time.time()
        for image_batch in dataset:
            train_step(image_batch)

        # Produce images for the GIF as you go
        display.clear_output(wait=True)
        generate_and_save_images(generator, epoch + 1, seed)

        # Save the model every 15 epochs
        if (epoch + 1) % 15 == 0:
            checkpoint.save(file_prefix=checkpoint_prefix)
```

```
        print('Time for epoch {} is {} sec'.format(epoch + 1, time.time() -
start))

    # Generate after the final epoch
    display.clear_output(wait=True)
    generate_and_save_images(generator, epochs, seed)
# Gerar e salvar imagens
def generate_and_save_images(model, epoch, test_input):
    # Notice `training` is set to False.
    # This is so all layers run in inference mode (batchnorm).
    predictions = model(test_input, training=False)

    fig = plt.figure(figsize=(4, 4))

    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i + 1)
        plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
        plt.axis('off')

    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
    plt.show()
# Treinar o modelo e restaurar o último ponto de verificação
train(train_dataset, EPOCHS)
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

FIGURA 16 – GERADOR DE DIGITOS TREINADO



FONTE: De autoria própria (2024)

```
# Criar um GIF
# Display a single image using the epoch number
```

```
def display_image(epoch_no):
    return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))

display_image(EPOCHS)

anim_file = 'dcgan.gif'
with imageio.get_writer(anim_file, mode='I') as writer:
    filenames = glob.glob('image*.png')
    filenames = sorted(filenames)
    for filename in filenames:
        image = imageio.imread(filename)
        writer.append_data(image)

import tensorflow_docs.vis.embed as embed
embed.embed_file(anim_file)
```

Questão 4

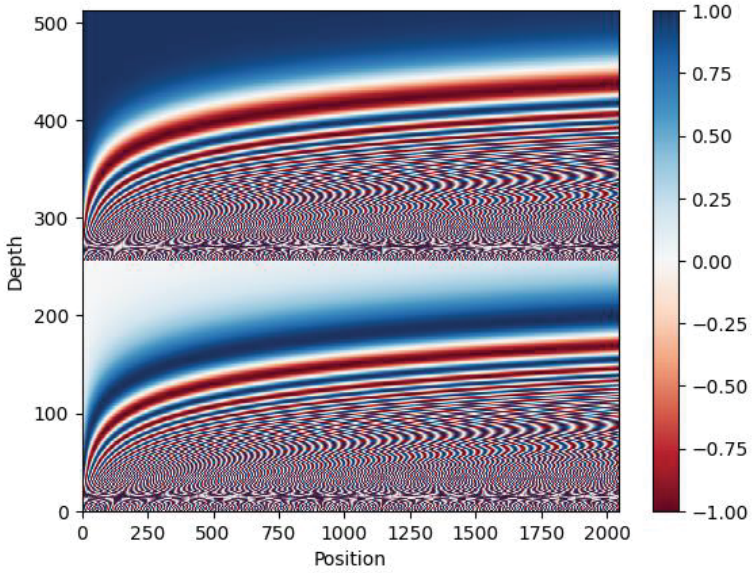
```
# Instalação e importação
!pip uninstall tensorflow
!pip install tensorflow==2.15.0
!pip install tensorflow_datasets
!pip install -U tensorflow-text==2.15.0
import collections
import logging
import os
import pathlib
import re
import string
import sys
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import tensorflow_text as text
import tensorflow as tf
logging.getLogger('tensorflow').setLevel(logging.ERROR) # suppress warnings
# Carregar a base de dados
examples, metadata = tfds.load('ted_hrlr_translate/pt_to_en',
with_info=True, as_supervised=True)
train_examples, val_examples = examples['train'], examples['validation']
# Verificar o dataset
```

```
for pt_examples, en_examples in train_examples.batch(3).take(1):
    for pt in pt_examples.numpy():
        print(pt.decode('utf-8'))
    print()
    for en in en_examples.numpy():
        print(en.decode('utf-8'))
# Tokenização e Destokenização do texto
model_name = "ted_hrlr_translate_pt_en_converter"
tf.keras.utils.get_file(
    f"{model_name}.zip",

f"https://storage.googleapis.com/download.tensorflow.org/models/{model_name
}.zip",
    cache_dir='.',
    cache_subdir='',
    extract=True
)

# Tem 2 tokenizers: um pt outro em en
# tokenizers.en tokeniza e detokeniza
tokenizers = tf.saved_model.load(model_name)
# PIPELINE DE ENTRADA
# Codificar/tokenizar lotes de texto puro
def tokenize_pairs(pt, en):
    pt = tokenizers.pt.tokenize(pt)
    # Converte ragged (irregular, tam variável) para dense
    # Faz padding com zeros.
    pt = pt.to_tensor()
    en = tokenizers.en.tokenize(en)
    # ragged -> dense
    en = en.to_tensor()
    return pt, en
# Pipeline simples: processa, embaralha, agrupa os dados, prefetch
# Datasets de entrada terminam com prefetch
BUFFER_SIZE = 20000
BATCH_SIZE = 64

def make_batches(ds):
    return (
        ds
        .cache()
```

```
        .shuffle(BUFFER_SIZE)
        .batch(BATCH_SIZE)
        .map(tokenize_pairs, num_parallel_calls=tf.data.AUTOTUNE)
        .prefetch(tf.data.AUTOTUNE)
    )

train_batches = make_batches(train_examples)
val_batches = make_batches(val_examples)
# CODIFICAÇÃO POSICIONAL
def get_angles(pos, i, d_model):
    angle_rates = 1 / np.power(10000, (2 * (i // 2)) / np.float32(d_model))
    return pos * angle_rates

def positional_encoding(position, d_model):
    angle_rads = get_angles(
        np.arange(position)[:, np.newaxis],
        np.arange(d_model)[np.newaxis, :],
        d_model
    )
    # sin em índices pares no array; 2i
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
    # cos em índices ímpares no array; 2i+1
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
    # newaxis, aumenta a dimensão [] -> [ [] ]
    pos_encoding = angle_rads[np.newaxis, ...]
    return tf.cast(pos_encoding, dtype=tf.float32)
# CODIFICAÇÃO POSICIONAL
n, d = 2048, 512
pos_encoding = positional_encoding(n, d)
print(pos_encoding.shape)

pos_encoding = pos_encoding[0]
# Arrumar as dimensões
pos_encoding = tf.reshape(pos_encoding, (n, d // 2, 2))
pos_encoding = tf.transpose(pos_encoding, (2, 1, 0))
pos_encoding = tf.reshape(pos_encoding, (d, n))

plt.pcolormesh(pos_encoding, cmap='RdBu')
plt.ylabel('Depth')
plt.xlabel('Position')
plt.colorbar()
```

```
plt.show()
```

FIGURA 17 – TRADUTOR DE TEXTO



FONTE: De autoria própria (2024)

```
# Cria uma máscara de 0 e 1, 0 para quando há valor e 1 quando não há
def create_padding_mask(seq):

    seq = tf.cast(tf.math.equal(seq, 0), tf.float32)
    # Add extra dimensions to add the padding
    # to the attention logits.
    return seq[:, tf.newaxis, tf.newaxis, :]  # (batch_size, 1, 1, seq_len)

# Máscara futura, usada no decoder
def create_look_ahead_mask(size):
    # Zera o triângulo inferior
    mask = 1 - tf.linalg.band_part(tf.ones((size, size)), -1, 0)
    return mask  # (seq_len, seq_len)

# Função de Atenção
def scaled_dot_product_attention(q, k, v, mask):
    # Q K^T
    matmul_qk = tf.matmul(q, k, transpose_b=True)  # (..., seq_len_q,
seq_len_k)

    # Converte matmul_qk para float32
    dk = tf.cast(tf.shape(k)[-1], tf.float32)
```

```
    # Divide por sqrt(d_k)
    scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

    # Soma a máscara, e os valores faltantes serão um número próximo a -inf
se mask for fornecida
    if mask is not None:
        scaled_attention_logits += (mask * -1e9)

    # Softmax normaliza os dados, somando 1. // (..., seq_len_q, seq_len_k)
    attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)

    # Calcula a saída
    output = tf.matmul(attention_weights, v)  # (..., seq_len_q, depth_v)

    return output, attention_weights~

class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads):
        super(MultiHeadAttention, self).__init__()
        self.num_heads = num_heads
        self.d_model = d_model
        assert d_model % self.num_heads == 0
        self.depth = d_model // self.num_heads

        self.wq = tf.keras.layers.Dense(d_model)
        self.wk = tf.keras.layers.Dense(d_model)
        self.wv = tf.keras.layers.Dense(d_model)
        self.dense = tf.keras.layers.Dense(d_model)

    def split_heads(self, x, batch_size):
        """Separa a última dimensão em (num_heads, depth).
        Transpõe o resultado para o shape (batch_size, num_heads, seq_len,
depth).
        """
        x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
        return tf.transpose(x, perm=[0, 2, 1, 3])

    def call(self, v, k, q, mask):
        batch_size = tf.shape(q)[0]
```

```
        q = self.wq(q)  # (batch_size, seq_len, d_model)
        k = self.wk(k)  # (batch_size, seq_len, d_model)
        v = self.wv(v)  # (batch_size, seq_len, d_model)

        q = self.split_heads(q, batch_size)  # (batch_size, num_heads,
seq_len_q, depth)
        k = self.split_heads(k, batch_size)  # (batch_size, num_heads,
seq_len_k, depth)
        v = self.split_heads(v, batch_size)  # (batch_size, num_heads,
seq_len_v, depth)

        # Calcula a atenção para cada cabeça (de forma matricial)
        # scaled_attention.shape == (batch_size, num_heads, seq_len_q,
depth)
        # attention_weights.shape == (batch_size, num_heads, seq_len_q,
seq_len_k)
        scaled_attention, attention_weights =
scaled_dot_product_attention(q, k, v, mask)

        # Troca a dimensão 2 com 1, para acertar o num_heads
        # (batch_size, seq_len_q, num_heads, depth)
        scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1,
3])

        # Concatena os valores em: (batch_size, seq_len_q, d_model)
        concat_attention = tf.reshape(scaled_attention, (batch_size, -1,
self.d_model))

        output = self.dense(concat_attention)  # (batch_size, seq_len_q,
d_model)

        return output, attention_weights

def point_wise_feed_forward_network(d_model, dff):
    return tf.keras.Sequential([
        tf.keras.layers.Dense(dff, activation='relu'),  # (batch_size,
seq_len, dff)
        tf.keras.layers.Dense(d_model)  # (batch_size, seq_len, d_model)
    ])
class EncoderLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads, dff, rate=0.1):
```

```
        super(EncoderLayer, self).__init__()
        self.mha = MultiHeadAttention(d_model, num_heads)
        self.ffn = point_wise_feed_forward_network(d_model, dff)
        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = tf.keras.layers.Dropout(rate)
        self.dropout2 = tf.keras.layers.Dropout(rate)

    def call(self, x, training, mask):
        attn_output, _ = self.mha(x, x, x, mask)  # (batch_size,
input_seq_len, d_model)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(x + attn_output)  # (batch_size,
input_seq_len, d_model)

        ffn_output = self.ffn(out1)  # (batch_size, input_seq_len, d_model)
        ffn_output = self.dropout2(ffn_output, training=training)
        out2 = self.layernorm2(out1 + ffn_output)  # (batch_size,
input_seq_len, d_model)

        return out2
class DecoderLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads, dff, rate=0.1):
        super(DecoderLayer, self).__init__()
        self.mha1 = MultiHeadAttention(d_model, num_heads)
        self.mha2 = MultiHeadAttention(d_model, num_heads)
        self.ffn = point_wise_feed_forward_network(d_model, dff)
        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm3 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = tf.keras.layers.Dropout(rate)
        self.dropout2 = tf.keras.layers.Dropout(rate)
        self.dropout3 = tf.keras.layers.Dropout(rate)

    def call(self, x, enc_output, training, look_ahead_mask, padding_mask):
        # enc_output.shape == (batch_size, input_seq_len, d_model)
        # (batch_size, target_seq_len, d_model)
        attn1, attn_weights_block1 = self.mha1(x, x, x, look_ahead_mask)
        attn1 = self.dropout1(attn1, training=training)
        out1 = self.layernorm1(attn1 + x)
        # (batch_size, target_seq_len, d_model)
```

```
        attn2, attn_weights_block2 = self.mha2(enc_output, enc_output,
out1, padding_mask)
        attn2 = self.dropout2(attn2, training=training)
        out2 = self.layernorm2(attn2 + out1) # (batch_size, target_seq_len,
d_model)
        ffn_output = self.ffn(out2) # (batch_size, target_seq_len, d_model)
        ffn_output = self.dropout3(ffn_output, training=training)
        out3 = self.layernorm3(ffn_output + out2) # (batch_size,
target_seq_len, d_model)
        return out3, attn_weights_block1, attn_weights_block2
class Encoder(tf.keras.layers.Layer):
    def __init__(self, num_layers, d_model, num_heads, dff,
                 input_vocab_size, maximum_position_encoding, rate=0.1):
        super(Encoder, self).__init__()

        self.d_model = d_model
        self.num_layers = num_layers

        self.embedding = tf.keras.layers.Embedding(input_vocab_size,
d_model)
        self.pos_encoding = positional_encoding(maximum_position_encoding,
self.d_model)

        self.enc_layers = [EncoderLayer(d_model, num_heads, dff, rate)
                           for _ in range(num_layers)]

        self.dropout = tf.keras.layers.Dropout(rate)

    def call(self, x, training, mask):
        seq_len = tf.shape(x)[1]

        # adding embedding and position encoding.
        x = self.embedding(x)  # (batch_size, input_seq_len, d_model)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x += self.pos_encoding[:, :seq_len, :]

        x = self.dropout(x, training=training)

        for i in range(self.num_layers):
            x = self.enc_layers[i](x, training, mask)
```

```
        return x  # (batch_size, input_seq_len, d_model)
class Decoder(tf.keras.layers.Layer):
    def __init__(self, num_layers, d_model, num_heads, dff,
target_vocab_size,
                 maximum_position_encoding, rate=0.1):
        super(Decoder, self).__init__()

        self.d_model = d_model
        self.num_layers = num_layers

        self.embedding = tf.keras.layers.Embedding(target_vocab_size,
d_model)
        self.pos_encoding = positional_encoding(maximum_position_encoding,
d_model)

        self.dec_layers = [DecoderLayer(d_model, num_heads, dff, rate)
                           for _ in range(num_layers)]

        self.dropout = tf.keras.layers.Dropout(rate)

    def call(self, x, enc_output, training, look_ahead_mask, padding_mask):
        seq_len = tf.shape(x)[1]
        attention_weights = {}

        x = self.embedding(x)  # (batch_size, target_seq_len, d_model)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x += self.pos_encoding[:, :seq_len, :]
        x = self.dropout(x, training=training)

        for i in range(self.num_layers):
            x, block1, block2 = self.dec_layers[i](x, enc_output, training,
                                                   look_ahead_mask,
padding_mask)
            attention_weights[f'decoder_layer{i+1}_block1'] = block1
            attention_weights[f'decoder_layer{i+1}_block2'] = block2

        # x.shape == (batch_size, target_seq_len, d_model)
        return x, attention_weights
class Transformer(tf.keras.Model):
    def __init__(self, num_layers, d_model, num_heads, dff,
input_vocab_size,
```

```
                 target_vocab_size, pe_input, pe_target, rate=0.1):
        super().__init__()
        self.encoder = Encoder(num_layers, d_model, num_heads, dff,
input_vocab_size,
                               pe_input, rate)
        self.decoder = Decoder(num_layers, d_model, num_heads, dff,
target_vocab_size,
                               pe_target, rate)
        self.final_layer = tf.keras.layers.Dense(target_vocab_size)

    def call(self, inputs, training):
        # Keras models prefer if you pass all your inputs in the first
argument
        inp, tar = inputs
        enc_padding_mask, look_ahead_mask, dec_padding_mask =
self.create_masks(inp, tar)

        # (batch_size, inp_seq_len, d_model)
        enc_output = self.encoder(inp, training, enc_padding_mask)

        # dec_output.shape == (batch_size, tar_seq_len, d_model)
        dec_output, attention_weights = self.decoder(
            tar, enc_output, training, look_ahead_mask, dec_padding_mask)

        # (batch_size, tar_seq_len, target_vocab_size)
        final_output = self.final_layer(dec_output)
        return final_output, attention_weights

    def create_masks(self, inp, tar):
        # Encoder padding mask
        enc_padding_mask = create_padding_mask(inp)

        # Used in the 2nd attention block in the decoder.
        # This padding mask is used to mask the encoder outputs.
        dec_padding_mask = create_padding_mask(inp)

        # Used in the 1st attention block in the decoder.
        # It is used to pad and mask future tokens in the input received by
        # the decoder.
        look_ahead_mask = create_look_ahead_mask(tf.shape(tar)[1])
        dec_target_padding_mask = create_padding_mask(tar)
```

```
        look_ahead_mask = tf.maximum(dec_target_padding_mask,
look_ahead_mask)

        return enc_padding_mask, look_ahead_mask, dec_padding_mask
# Hiperparâmetros
num_layers = 4
d_model = 128
dff = 512
num_heads = 8
dropout_rate = 0.1
class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):
    def __init__(self, d_model, warmup_steps=4000):
        super(CustomSchedule, self).__init__()
        self.d_model = d_model
        self.d_model = tf.cast(self.d_model, tf.float32)
        self.warmup_steps = warmup_steps

    def __call__(self, step):
        step = tf.cast(step, tf.float32)  # Adicionado para evitar ERRO
        arg1 = tf.math.rsqrt(step)
        arg2 = step * (self.warmup_steps ** -1.5)
        return tf.math.rsqrt(self.d_model) * tf.math.minimum(arg1, arg2)

learning_rate = CustomSchedule(d_model)
optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.9,
beta_2=0.98, epsilon=1e-9)
# Define a função de perda
loss_object =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)
    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask
    return tf.reduce_sum(loss_) / tf.reduce_sum(mask)

def accuracy_function(real, pred):
    accuracies = tf.equal(real, tf.argmax(pred, axis=2))
    mask = tf.math.logical_not(tf.math.equal(real, 0))
```

```
    accuracies = tf.math.logical_and(mask, accuracies)
    accuracies = tf.cast(accuracies, dtype=tf.float32)
    mask = tf.cast(mask, dtype=tf.float32)
    return tf.reduce_sum(accuracies) / tf.reduce_sum(mask)

# Define as métricas
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.Mean(name='train_accuracy')

transformer = Transformer(
num_layers=num_layers,
d_model=d_model,
num_heads=num_heads,
dff=dff,
input_vocab_size=tokenizers.pt.get_vocab_size().numpy(),
target_vocab_size=tokenizers.en.get_vocab_size().numpy(),
pe_input=1000,
pe_target=1000,
rate=dropout_rate)
# Checkpoint
checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(transformer=transformer, optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path,
max_to_keep=5)

# if a checkpoint exists, restore the latest checkpoint.
if ckpt_manager.latest_checkpoint:
    ckpt.restore(ckpt_manager.latest_checkpoint)
    print('Latest checkpoint restored!!')

EPOCHS = 20
train_step_signature = [
    tf.TensorSpec(shape=(None, None), dtype=tf.int64),
    tf.TensorSpec(shape=(None, None), dtype=tf.int64),
]

@tf.function(input_signature=train_step_signature)
def train_step(inp, tar):
    tar_inp = tar[:, :-1]
    tar_real = tar[:, 1:]
```

```
    with tf.GradientTape() as tape:
        predictions, _ = transformer([inp, tar_inp], training=True)
        loss = loss_function(tar_real, predictions)
        gradients = tape.gradient(loss, transformer.trainable_variables)
        optimizer.apply_gradients(zip(gradients,
transformer.trainable_variables))

    train_loss(loss)
    train_accuracy(accuracy_function(tar_real, predictions))

for epoch in range(EPOCHS):
    start = time.time()
    train_loss.reset_state()
    train_accuracy.reset_state()

    # inp -> português, tar -> inglês
    for batch, (inp, tar) in enumerate(train_batches):
        train_step(inp, tar)

        if batch % 50 == 0:
            print(f'Epoch {epoch + 1} Batch {batch} Loss
{train_loss.result():.4f} Accuracy {train_accuracy.result():.4f}')

    if (epoch + 1) % 5 == 0:
        ckpt_save_path = ckpt_manager.save()
        print(f'Saving checkpoint for epoch {epoch + 1} at
{ckpt_save_path}')

    print(f'Epoch {epoch + 1} Loss {train_loss.result():.4f} Accuracy
{train_accuracy.result():.4f}')
    print(f'Time taken for 1 epoch: {time.time() - start:.2f} secs\n')

Epoch 1 Batch 0 Loss 8.8537 Accuracy 0.0000
Epoch 1 Batch 50 Loss 8.7983 Accuracy 0.0050
Epoch 1 Batch 100 Loss 8.7028 Accuracy 0.0247
Epoch 1 Batch 150 Loss 8.5897 Accuracy 0.0316
Epoch 1 Batch 200 Loss 8.4496 Accuracy 0.0352
Epoch 1 Batch 250 Loss 8.2822 Accuracy 0.0373
Epoch 1 Batch 300 Loss 8.0950 Accuracy 0.0387
Epoch 1 Batch 350 Loss 7.8998 Accuracy 0.0427
Epoch 1 Batch 400 Loss 7.7112 Accuracy 0.0507
```

```
Epoch 1 Batch 450 Loss 7.5412 Accuracy 0.0597
Epoch 1 Batch 500 Loss 7.3905 Accuracy 0.0691
Epoch 1 Batch 550 Loss 7.2529 Accuracy 0.0783
Epoch 1 Batch 600 Loss 7.1254 Accuracy 0.0869
Epoch 1 Batch 650 Loss 7.0069 Accuracy 0.0947
Epoch 1 Batch 700 Loss 6.8964 Accuracy 0.1019
Epoch 1 Batch 750 Loss 6.7959 Accuracy 0.1086
Epoch 1 Batch 800 Loss 6.7021 Accuracy 0.1148
Epoch 1 Loss 6.6863 Accuracy 0.1159
Time taken for 1 epoch: 168.15 secs

class Translator(tf.Module):
    def __init__(self, tokenizers, transformer):
        self.tokenizers = tokenizers
        self.transformer = transformer

    def __call__(self, sentence, max_length=20):
        # Verifica se o input é um tensor
        assert isinstance(sentence, tf.Tensor)

        if len(sentence.shape) == 0:
            sentence = sentence[tf.newaxis]

        # Tokeniza a sentença de entrada em português
        sentence = self.tokenizers.pt.tokenize(sentence).to_tensor()
        encoder_input = sentence

        # Obtém os tokens de início e fim em inglês
        start_end = self.tokenizers.en.tokenize([''])[0]
        start = start_end[0][tf.newaxis]
        end = start_end[1][tf.newaxis]

        # Inicializa o array de saída
        output_array = tf.TensorArray(dtype=tf.int64, size=0,
dynamic_size=True)
        output_array = output_array.write(0, start)

        for i in tf.range(max_length):
            output = tf.transpose(output_array.stack())

            # Gera previsões com o transformer
```

```
            predictions, _ = self.transformer([encoder_input, output],
training=False)
            predictions = predictions[:, -1:, :]  # (batch_size, 1,
vocab_size)
            predicted_id = tf.argmax(predictions, axis=-1)

            # Adiciona o token previsto ao array de saída
            output_array = output_array.write(i + 1, predicted_id[0])

            if predicted_id == end:
                break

        # Converte a saída em texto e tokens
        output = tf.transpose(output_array.stack())
        text = self.tokenizers.en.detokenize(output)[0]
        tokens = self.tokenizers.en.lookup(output)[0]

        # Obtém os pesos de atenção
        _, attention_weights = self.transformer([encoder_input, output[:,
:-1]], training=False)

        return text, tokens, attention_weights

translator = Translator(tokenizers, transformer)
sentence = "Eu li sobre triceratops na enciclopédia."
translated_text, translated_tokens, attention_weights =
translator(tf.constant(sentence))
print(f'{"Prediction":15s}: {translated_text}')
```

APÊNDICE I - BIG DATA

A – ENUNCIADO

Enviar um arquivo PDF contendo uma descrição breve (2 páginas) sobre a implementação de uma aplicação ou estudo de caso envolvendo Big Data e suas ferramentas (NoSQL e NewSQL). Caracterize os dados e Vs envolvidos, além da modelagem necessária dependendo dos modelos de dados empregados.

B – RESOLUÇÃO

Contexto do estudo de caso
Este estudo de caso baseia-se na rotina de um integrante da equipe de TI de uma rede de supermercados com 29 lojas no Paraná. Para entender o comportamento dos clientes, foi proposto o desenvolvimento de um escopo de Big Data. O objetivo é analisar o perfil dos clientes cadastrados via aplicativo da rede, segmentando-os de acordo com seus hábitos de compra. Para isso, serão utilizados dados extraídos de sistemas CRM e ERP, como histórico de compras e cadastro de clientes e produtos, para realizar uma análise comportamental, identificar padrões de consumo e auxiliar na personalização de campanhas de marketing.

Ferramentas e tecnologias utilizadas
Para o projeto, serão utilizadas diversas ferramentas e tecnologias. Nos sistemas de Log Management, o IBM Qradar será empregado para correlação de eventos de segurança em tempo real. A plataforma de Big Data será o Apache Hadoop, que é o núcleo do processamento de dados devido à sua capacidade de lidar com grandes volumes de dados distribuídos. Utilizando o MapReduce, os dados serão processados de forma distribuída, permitindo análises rápidas e eficientes, como a classificação de perfis de clientes.

Para o armazenamento de dados não estruturados e semiestruturados, como histórico de compras e preferências, serão usados bancos de dados NoSQL como Cassandra e HBase. Esses sistemas são ideais para lidar com grandes volumes de dados com velocidade e flexibilidade.

Para a análise comportamental, serão aplicadas técnicas de Machine Learning com bibliotecas como Mahout e Spark MLlib. Elas permitirão a criação de algoritmos de clustering para segmentar os clientes em três grupos: clientes que gastam muito, clientes com gastos moderados e clientes que gastam pouco. Além disso, modelos preditivos serão desenvolvidos para recomendar produtos com base no perfil de cada cliente, auxiliando nas campanhas de marketing personalizadas.

Organização dos dados

A organização dos dados será feita através de diferentes abordagens de modelagem, conforme os tipos de dados existentes nos sistemas CRM e ERP.

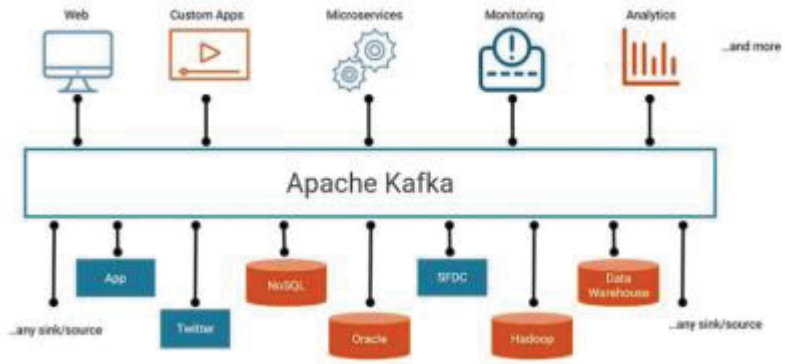
Modelo Colunar: O HBase, como banco de dados NoSQL colunar, será utilizado para armazenar dados do histórico de compras e cadastros de clientes, proporcionando um modelo eficiente para leituras rápidas.

Modelo Chave-Valor: Sistemas como o Redis podem ser usados para armazenar preferências instantâneas dos clientes, como produtos visualizados recentemente.

Spark: Será empregado para processar grandes volumes de dados, rodar algoritmos de clustering (K-means) para segmentar os clientes e aplicar modelos de aprendizado de máquina para prever comportamentos de consumo e recomendar produtos. Essa arquitetura permite um fluxo contínuo de dados entre diferentes componentes.

A captura de dados em tempo real do PDV (ponto de venda) será realizada com Kafka e Storm. O Kafka terá o papel de gerir em tempo real os dados de CRM/ERP, consolidando o fluxo de transações e o comportamento dos clientes. O Storm pode ser usado para processar esses dados conforme chegam, com uma arquitetura distribuída que garante alta disponibilidade, semelhante ao que é feito no pipeline de análise da Figura 18.

FIGURA 18 – APACHE KAFKA



FONTE: Evandro Souza (2018)

Para a visualização, o Grafana será utilizado para análises em tempo real das requisições e informações relacionadas ao desempenho geral do produto. O Power BI será usado para a criação de dashboards personalizados para as equipes de marketing, e-commerce e outros setores administrativos da rede.

A arquitetura do projeto pode ser resumida em cinco etapas: 1. Ingestão de Dados: Kafka para coleta de dados em tempo real dos sistemas CRM/ERP. 2. Processamento de Dados: Storm processando streams de dados, seguido de persistência no Hadoop HDFS e bancos NoSQL (HBase ou Cassandra). 3. Análise: Spark para análise distribuída e segmentação de clientes com aprendizado de máquina. 4. Visualização: Dashboards interativos com Grafana e Power BI para análise do comportamento dos clientes. 5. Recomendação de Produtos: Sistemas baseados em aprendizado de máquina para recomendar produtos em tempo real, de acordo com o comportamento dos clientes.

Especificação dos Vs

Volume: Espera-se um volume considerável de dados a serem processados, provenientes do histórico de compras dos clientes das 29 lojas e dos cadastros no aplicativo.

Variedade: O conjunto de dados é altamente heterogêneo, incluindo dados estruturados (cadastros de clientes e produtos) e não estruturados (logs de navegação e preferências). A utilização de sistemas NoSQL é crucial para lidar com essa variedade.

Velocidade: A análise em tempo real é necessária para recomendações instantâneas de produtos e personalização de campanhas. Para isso, será utilizado o processamento de dados em movimento com ferramentas como o Apache Storm e Kafka, que fornecem capacidades de streaming. O Kafka possui ótima performance no envio de mensagens que serão consumidas via tópicos.

Veracidade: Garantir a integridade e a qualidade dos dados é fundamental. Sistemas distribuídos como o Hadoop têm mecanismos para garantir a consistência dos dados, mesmo em grandes volumes e alta diversidade, o que reforça sua escolha.

Valor: A análise comportamental trará valor ao negócio ao identificar padrões de compra, permitindo a segmentação precisa dos clientes e gerando insights valiosos para campanhas de marketing personalizadas e auxílio na compra de produtos em geral.

APÊNDICE J - VISÃO COMPUTACIONAL

A – ENUNCIADO

1) Extração de Características

Os bancos de imagens fornecidos são conjuntos de imagens de 250x250 pixels de imuno-histoquímica (biópsia) de câncer de mama. No total são 4 classes (0, 1+, 2+ e 3+) que estão divididas em diretórios. O objetivo é classificar as imagens nas categorias correspondentes. Uma base de imagens será utilizada para o treinamento e outra para o teste do treino.

As imagens fornecidas são recortes de uma imagem maior do tipo WSI *(Whole Slide Imaging*) disponibilizada pela Universidade de Warwick (link). A nomenclatura das imagens segue o padrão XX_HER_YYYY.png, onde XX é o número do paciente e YYYY é o número da imagem recortada. Separe a base de treino em 80% para treino e 20% para validação. **Separe por pacientes (XX), não utilize a separação randômica! Pois, imagens do mesmo paciente não podem estar na base de treino e de validação, pois isso pode gerar um viés.** No caso da CNN VGG16 remova a última camada de classificação e armazene os valores da penúltima camada como um vetor de características. Após o treinamento, os modelos treinados devem ser validados na base de teste.

Tarefas:

a) Carregue a base de dados de **Treino.**
b) Crie partições contendo 80% para treino e 20% para validação (atenção aos pacientes).
c) Extraia características utilizando LBP e a CNN VGG16 (gerando um csv para cada extrator).
d) Treine modelos Random Forest, SVM e RNA para predição dos dados extraídos.
e) Carregue a base de **Teste** e execute a tarefa 3 nesta base.
f) Aplique os modelos treinados nos dados de treino
g) Calcule as métricas de Sensibilidade, Especificidade e F1-Score com base em suas matrizes de confusão.
h) Indique qual modelo dá o melhor o resultado e a métrica utilizada

2) Redes Neurais

Utilize as duas bases do exercício anterior para treinar as Redes Neurais Convolucionais VGG16 e a Resnet50. Utilize os pesos pré-treinados (*Transfer Learning*), refaça as camadas *Fully Connected* para o problema de 4 classes. Compare os treinos de 15 épocas com e sem *Data Augmentation*. Tanto a VGG16 quanto a Resnet50 têm como camada de entrada uma imagem 224x224x3, ou seja, uma imagem de 224x224 pixels coloridos (3 canais de cores). Portanto, será necessário fazer uma transformação de 250x250x3 para 224x224x3. Ao fazer o *Data Augmentation* **cuidado** para não alterar demais as cores das imagens e atrapalhar na classificação.

Tarefas:

a) Utilize a base de dados de **Treino** já separadas em treino e validação do exercício anterior
b) Treine modelos VGG16 e Resnet50 adaptadas com e sem *Data Augmentation*
c) Aplique os modelos treinados nas imagens da base de **Teste**
d) Calcule as métricas de Sensibilidade, Especificidade e F1-Score com base em suas matrizes de confusão.
e) Indique qual modelo dá o melhor o resultado e a métrica utilizada

B – RESOLUÇÃO

Questão 1

```
       from google.colab import drive
       drive.mount('/content/drive')

       file_path                    =                    '/content/drive/My
Drive/VisaoComputacional/Train_Warwick.zip'
       import zipfile
       with zipfile.ZipFile(file_path, 'r') as zip_ref:
           zip_ref.extractall('/content/')

       file_path                    =                    '/content/drive/My
Drive/VisaoComputacional/Test_Warwick.zip'
       import zipfile
       with zipfile.ZipFile(file_path, 'r') as zip_ref:
           zip_ref.extractall('/content/')

       """# Importando as bibliotecas necessárias"""

       import os
       import numpy as np
       import cv2
       import pandas as pd
       from    tensorflow.keras.preprocessing.image    import    load_img,
img_to_array
       from sklearn.model_selection import train_test_split
       from sklearn.preprocessing import LabelEncoder
       from skimage import feature

       from keras.applications import VGG16
       from keras.layers import GlobalAveragePooling2D, Dense
       from keras.models import Model
```

```
        from keras.preprocessing.image import load_img, img_to_array

        from sklearn.ensemble import RandomForestClassifier
        from sklearn.svm import SVC
        from sklearn.neural_network import MLPClassifier
        from sklearn.metrics import confusion_matrix, classification_report

        """Tarefa 1: Carregar a Base de Dados de Treino"""

        # Função para carregar as imagens e rótulos
        def carregar_imagens(diretorio):
            imagens = []
            rotulos = []
            pacientes = []

            for subdir, dirs, files in os.walk(diretorio):
                # Extrair o rótulo da classe a partir do nome da pasta
                classe = os.path.basename(subdir)  # Nome da pasta atual

                for file in files:
                    if file.endswith('.png'):
                        # Extrair ID do paciente
                        partes = file.split('_')
                        id_paciente = partes[0]  # Parte XX (ID do paciente)
                        # Adicionar o rótulo da classe
                        rotulo = classe  # A classe é o nome da pasta

                        # Carregar a imagem com cv2
                        img = cv2.imread(os.path.join(subdir, file))
                         img_array = img.astype('float32')  # Converter para
float32

                        # Adicionar aos arrays
                        imagens.append(img_array)
                        rotulos.append(rotulo)  # Adiciona o rótulo da classe
                        pacientes.append(id_paciente)

            return np.array(imagens), np.array(rotulos), np.array(pacientes)

        # Diretório onde as imagens foram extraídas
        train_dir = '/content/Train_4cls_amostra'
```

```
        # Carregar as imagens de treino
        X, y, pacientes = carregar_imagens(train_dir)

        # Verificar se as imagens foram carregadas corretamente
        print(f"Número de imagens carregadas: {len(X)}")
        print(f"Número de rótulos carregados: {len(y)}")
        print(f"Número de pacientes: {len(np.unique(pacientes))}")

        # Codificar os rótulos de classe em números
        label_encoder = LabelEncoder()
        y_treino_encoded = label_encoder.fit_transform(y)  # Transforma as
classes em números

        """Tarefa 2: Criar Partições de Treino e Validação"""

        # Função para dividir manualmente os dados de treino e validação por
pacientes, garantindo a presença de todas as classes
        def dividir_por_paciente_manual(X, y, pacientes,
proporcao_treino=0.8, n_classes=4):

            idx_treino = []  # Lista para armazenar os índices do conjunto de
treino
            idx_val = []     # Lista para armazenar os índices do conjunto de
validação

            # Para cada classe (0, 1, 2, ..., n_classes-1)
            for classe in range(n_classes):
                # Obter os índices das imagens que pertencem à classe atual
                idx_classe = np.where(y == classe)[0]

                # Obter os IDs únicos dos pacientes que possuem imagens nesta
classe
                pacientes_classe = np.unique(pacientes[idx_classe])

                 # Determinar o número de pacientes que irão para o conjunto
de treino
                      n_pacientes_treino = int(len(pacientes_classe) *
proporcao_treino)

                # Dividir os pacientes dessa classe entre treino e validação
```

```
                                        pacientes_treino_classe =
pacientes_classe[:n_pacientes_treino]  # Primeira parte vai para treino
                                          pacientes_val_classe =
pacientes_classe[n_pacientes_treino:]     # O restante vai para validação

               # Identificar os índices no array original de pacientes que
pertencem ao conjunto de treino e validação
                              idx_treino_classe    =    np.isin(pacientes,
pacientes_treino_classe)  # Índices de treino para a classe atual
                                idx_val_classe    =    np.isin(pacientes,
pacientes_val_classe)        # Índices de validação para a classe atual

               # Adicionar esses índices às listas principais
               idx_treino.extend(np.where(idx_treino_classe)[0])
               idx_val.extend(np.where(idx_val_classe)[0])

           # Dividir os dados de entrada (X) e rótulos (y) com base nos
índices de treino e validação
           X_treino, X_val = X[idx_treino], X[idx_val]
           y_treino, y_val = y[idx_treino], y[idx_val]

           # Separar os IDs dos pacientes correspondentes aos conjuntos de
treino e validação
           pacientes_treino = pacientes[idx_treino]
           pacientes_val = pacientes[idx_val]

           # Retornar os dados divididos
             return  X_treino,  X_val,  y_treino,  y_val,  pacientes_treino,
pacientes_val

       # Realizar a divisão manual
       X_treino, X_val, y_treino, y_val, pacientes_treino, pacientes_val =
dividir_por_paciente_manual(X, y_treino_encoded, pacientes)

       # Verificar a divisão
       print(f"Imagens de treino: {len(X_treino)}, Imagens de validação:
{len(X_val)}")
       print(f"Pacientes de treino: {np.unique(pacientes_treino)}, Pacientes
de validação: {np.unique(pacientes_val)}")

       # Verificar se há interseção entre os pacientes
```

```
       pacientes_em_comum = np.intersect1d(pacientes_treino, pacientes_val)

       # Mostrar os resultados
       if len(pacientes_em_comum) == 0:
           print("Não há pacientes repetidos entre os conjuntos de treino e
validação.")
       else:
            print(f"Os seguintes pacientes estão em ambos os conjuntos:
{pacientes_em_comum}")

       # Verificar a distribuição de classes nos conjuntos de treino e
validação
       def verificar_distribuicao_classes(y_treino, y_val, n_classes=4):
           # Contar quantas amostras existem de cada classe no conjunto de
treino
           for classe in range(n_classes):
               count_treino = np.sum(y_treino == classe)
               count_val = np.sum(y_val == classe)
                print(f"Classe {classe}: {count_treino} imagens no treino,
{count_val} imagens na validação.")

           # Verificar se todas as classes estão presentes
           classes_treino = np.unique(y_treino)
           classes_val = np.unique(y_val)

            if len(classes_treino) == n_classes and len(classes_val) ==
n_classes:
                print("Todas as classes estão presentes nos conjuntos de
treino e validação.")
           else:
               print("Alguma classe está faltando no conjunto de treino ou
validação.")

       # Chamar a função para verificar a distribuição das classes
       verificar_distribuicao_classes(y_treino, y_val, n_classes=4)

       """Tarefa 3: Extrair Características com LBP e CNN VGG16

       Função LBP
       """
```

```
        def calcular_lbp(imagem):
            # Converter a imagem para escala de cinza
            imagem_gray = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)

            # Calcular o LBP usando 8 pontos em um raio de 1 pixel
             # 'uniform' considera apenas padrões uniformes, o que reduz a
complexidade
              lbp  =  feature.local_binary_pattern(imagem_gray,  P=8,  R=1,
method='uniform')

            # Criar um histograma dos valores LBP
            # ravel() transforma a matriz LBP em um vetor unidimensional
            # bins cria 10 bins para o histograma, de 0 a 10
             (hist,  _)  =  np.histogram(lbp.ravel(),  bins=np.arange(0,  11),
range=(0, 10))

            # Converter o histograma para float para normalização
            hist = hist.astype("float")

            # Normalizar o histograma para que a soma total seja 1
            hist /= hist.sum()

            # Retornar o histograma normalizado
            return hist

        # Extrair características LBP conjunto 80% - Treino
        lbp_caracteristicas = []
        for img in X_treino:
            lbp_hist = calcular_lbp(img)
            lbp_caracteristicas.append(lbp_hist)

        df_lbp = pd.DataFrame(lbp_caracteristicas)
        df_lbp['classe'] = y_treino
        df_lbp.to_csv('caracteristicas_lbpTreino.csv', index=False)

        # Extrair características LBP conjunto 20% - Treino
        lbp_caracteristicas = []
        for img in X_val:
            lbp_hist = calcular_lbp(img)
            lbp_caracteristicas.append(lbp_hist)
```

```
        df_lbp_val = pd.DataFrame(lbp_caracteristicas)
        df_lbp_val['classe'] = y_val
        df_lbp_val.to_csv('caracteristicas_lbpVal.csv', index=False)

        """CNN VGG16

        VGG16 80% Treino
        """

        # Carregar o modelo VGG16 sem a última camada de classificação
        base_model     =     VGG16(weights='imagenet',     include_top=False,
input_shape=(250, 250, 3))

        # Adicionar uma camada de pooling global para transformar as saídas
em vetores
        x = base_model.output
        x = GlobalAveragePooling2D()(x)

        # Criar um novo modelo que retorna as características da penúltima
camada
        model = Model(inputs=base_model.input, outputs=x)

        # Normalizar as imagens para o intervalo [0, 1]
        X_treino_normalizado = X_treino / 255.0

        # Extrair características usando a CNN VGG16
        features_vgg = model.predict(X_treino_normalizado)  # Aqui os valores
da penúltima camada são armazenados

        # Criar um DataFrame com as características extraídas
        df_vgg  =  pd.DataFrame(features_vgg)   #  Os  valores  estão  agora  em
df_vgg
        df_vgg['classe'] = y_treino  # Adicionar rótulos às características
        df_vgg.to_csv('caracteristicas_vgg16Treino.csv',   index=False)    #
Salvar em um CSV

        """    VGG16 20% Validação"""

        # Carregar o modelo VGG16 sem a última camada de classificação
        base_model     =     VGG16(weights='imagenet',     include_top=False,
input_shape=(250, 250, 3))
```

```
        # Adicionar uma camada de pooling global para transformar as saídas
em vetores
        x = base_model.output
        x = GlobalAveragePooling2D()(x)

        # Criar um novo modelo que retorna as características da penúltima
camada
        model = Model(inputs=base_model.input, outputs=x)

        # Normalizar as imagens para o intervalo [0, 1]
        X_val_normalizado = X_val / 255.0

        # Extrair características usando a CNN VGG16
        features_vgg = model.predict(X_val_normalizado)  # Aqui os valores da
penúltima camada são armazenados

        # Criar um DataFrame com as características extraídas
        df_vgg_val = pd.DataFrame(features_vgg)  # Os valores estão agora em
df_vgg
        df_vgg_val['classe'] = y_val  # Adicionar rótulos às características
        df_vgg_val.to_csv('caracteristicas_vgg16Val.csv',   index=False)    #
Salvar em um CSV

        """Tarefa 4 - Treinar Modelos Random Forest, SVM e RNA"""

        # Definir semente de aleatoriedade
        seed = 123

        # Treinar Random Forest lbp com semente
        rf_model_lbp = RandomForestClassifier(random_state=seed)
        rf_model_lbp.fit(df_lbp.drop('classe', axis=1), df_lbp['classe'])

        # Treinar SVM lbp com semente
        svm_model_lbp = SVC(random_state=seed)
        svm_model_lbp.fit(df_lbp.drop('classe', axis=1), df_lbp['classe'])

        # Treinar RNA lbp com semente
        rna_model_lbp = MLPClassifier(random_state=seed)
        rna_model_lbp.fit(df_lbp.drop('classe', axis=1), df_lbp['classe'])
```

```
        # Definir semente de aleatoriedade
        seed = 123

        # Treinar Random Forest vgg16 com semente
        rf_model_vgg16 = RandomForestClassifier(random_state=seed)
        rf_model_vgg16.fit(df_vgg.drop('classe', axis=1), df_vgg['classe'])

        # Treinar SVM vgg16 com semente
        svm_model_vgg16 = SVC(random_state=seed)  # Verificar se a versão do
SVC suporta random_state
        svm_model_vgg16.fit(df_vgg.drop('classe', axis=1), df_vgg['classe'])

        # Treinar RNA vgg16 com semente
        rna_model_vgg16 = MLPClassifier(random_state=seed)
        rna_model_vgg16.fit(df_vgg.drop('classe', axis=1), df_vgg['classe'])

        # Fazer previsões com os modelos treinados 20% validação lbp

        pred_rf_lbp = rf_model_lbp.predict(df_lbp_val.drop('classe', axis=1))
        pred_svm_lbp    =    svm_model_lbp.predict(df_lbp_val.drop('classe',
axis=1))
        pred_rna_lbp    =    rna_model_lbp.predict(df_lbp_val.drop('classe',
axis=1))

        # Para Random Forest
        print("Random Forest:")
        print(confusion_matrix(y_val, pred_rf_lbp))
        print(classification_report(y_val, pred_rf_lbp))

        # Para SVM
        print("SVM:")
        print(confusion_matrix(y_val, pred_svm_lbp))
        print(classification_report(y_val, pred_svm_lbp))

        # Para RNA
        print("RNA:")
        print(confusion_matrix(y_val, pred_rna_lbp))
        print(classification_report(y_val, pred_rna_lbp))

        # Fazer previsões com os modelos treinados 20% validação vgg16
```

```
        pred_rf_vgg16   =   rf_model_vgg16.predict(df_vgg_val.drop('classe',
axis=1))
        pred_svm_vgg16  =  svm_model_vgg16.predict(df_vgg_val.drop('classe',
axis=1))
        pred_rna_vgg16  =  rna_model_vgg16.predict(df_vgg_val.drop('classe',
axis=1))

        # Para Random Forest
        print("Random Forest:")
        print(confusion_matrix(y_val, pred_rf_vgg16))
        print(classification_report(y_val, pred_rf_vgg16))

        # Para SVM
        print("SVM:")
        print(confusion_matrix(y_val, pred_svm_vgg16))
        print(classification_report(y_val, pred_svm_vgg16))

        # Para RNA
        print("RNA:")
        print(confusion_matrix(y_val, pred_rna_vgg16))
        print(classification_report(y_val, pred_rna_vgg16))

        """Tarefa 5: Carregar a Base de Teste e Extrair Características"""

        # Diretório onde as imagens de teste estão localizadas
        test_dir = '/content/Test_4cl_amostra'

        # Carregar as imagens de teste
        X_teste, y_teste, pacientes_teste = carregar_imagens(test_dir)

        # Verificar se as imagens foram carregadas corretamente
        print(f"Número de imagens carregadas para teste: {len(X_teste)}")
        print(f"Número de rótulos carregados para teste: {len(y_teste)}")
        print(f"Número         de         pacientes         no         teste:
{len(np.unique(pacientes_teste))}")

        # Para codificar as classes em números, se necessário
        from sklearn.preprocessing import LabelEncoder

        # Codificar os rótulos de classe em números
        label_encoder = LabelEncoder()
```

```
        y_teste_encoded = label_encoder.fit_transform(y_teste)  # Isso
transforma as classes em números

        """LBP Base Teste"""

        # Extrair características LBP do conjunto de teste
        lbp_caracteristicas_teste = []
        for img in X_teste:
            lbp_hist = calcular_lbp(img)
            lbp_caracteristicas_teste.append(lbp_hist)

        # Criar DataFrame e salvar como CSV
        df_lbp_teste = pd.DataFrame(lbp_caracteristicas_teste)

        # Converter os nomes das colunas para strings
        #df_lbp_teste.columns = df_lbp_teste.columns.astype(str)

        df_lbp_teste['classe'] = y_teste  # Usar y_teste para rótulos
        df_lbp_teste.to_csv('caracteristicas_lbp_teste.csv', index=False)

        """VGG16 Base teste"""

        # Carregar o modelo VGG16 sem a última camada de classificação
        base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(250, 250, 3))

        # Adicionar uma camada de pooling global para transformar as saídas
em vetores
        x = base_model.output
        x = GlobalAveragePooling2D()(x)

        # Criar um novo modelo que retorna as características da penúltima
camada
        model = Model(inputs=base_model.input, outputs=x)

        # Normalizar as imagens do conjunto de teste para o intervalo [0, 1]
        X_teste_normalizado = X_teste / 255.0  # Alterado para X_teste

        # Extrair características usando a CNN VGG16
        features_vgg = model.predict(X_teste_normalizado)  # Aqui os valores
da penúltima camada são armazenados
```

```
        # Criar um DataFrame com as características extraídas
        df_vgg_teste = pd.DataFrame(features_vgg)  # Os valores estão agora
em df_vgg
        df_vgg_teste['classe'] = y_teste  # Alterado para y_teste para
adicionar rótulos às características
        df_vgg_teste.to_csv('caracteristicas_vgg16_teste.csv',
index=False)  # Salvar em um CSV com um nome diferente

        """Previsões LBP"""

        # Fazer previsões com os modelos treinados lbp
        pred_rf_teste_lbp = rf_model_lbp.predict(df_lbp_teste.drop('classe',
axis=1))
        pred_svm_teste_lbp                                                =
svm_model_lbp.predict(df_lbp_teste.drop('classe', axis=1))
        pred_rna_teste_lbp                                                =
rna_model_lbp.predict(df_lbp_teste.drop('classe', axis=1))

        y_teste = label_encoder.transform(y_teste)

        # Para Random Forest
        print("Random Forest:")
        print(confusion_matrix(y_teste, pred_rf_teste_lbp))
        print(classification_report(y_teste, pred_rf_teste_lbp))

        # Para SVM
        print("SVM:")
        print(confusion_matrix(y_teste, pred_svm_teste_lbp))
        print(classification_report(y_teste, pred_svm_teste_lbp))

        # Para RNA
        print("RNA:")
        print(confusion_matrix(y_teste, pred_rna_teste_lbp))
        print(classification_report(y_teste, pred_rna_teste_lbp))

        """Previsões VGG16"""

        # Fazer previsões com os modelos treinados vgg16
        pred_rf_teste_vgg                                                 =
rf_model_vgg16.predict(df_vgg_teste.drop('classe', axis=1))
```

```
       pred_svm_teste_vgg                                                    =
svm_model_vgg16.predict(df_vgg_teste.drop('classe', axis=1))
       pred_rna_teste_vgg                                                    =
rna_model_vgg16.predict(df_vgg_teste.drop('classe', axis=1))

       # Para Random Forest
       print("Random Forest:")
       print(confusion_matrix(y_teste, pred_rf_teste_vgg))
       print(classification_report(y_teste, pred_rf_teste_vgg))

       # Para SVM
       print("SVM:")
       print(confusion_matrix(y_teste, pred_svm_teste_vgg))
       print(classification_report(y_teste, pred_svm_teste_vgg))
       # Para RNA
       print("RNA:")
       print(confusion_matrix(y_teste, pred_rna_teste_vgg))
       print(classification_report(y_teste, pred_rna_teste_vgg))
```

TABELA 9 – DESEMPENHO DOS MODELOS COM CARACTERÍSTICAS LBP

Desempenho dos Modelos com Características LBP

Classe	Modelo	Acurácia	Sensibilidade	Especificidade	F1-Score
0	Random Forest	0.66	0.50	0.80	0.56
1	Random Forest	0.66	0.63	0.81	0.56
2	Random Forest	0.66	0.56	0.87	0.55
3	Random Forest	0.66	0.97	0.98	0.97
0	SVM	0.51	0.02	0.75	0.04
1	SVM	0.51	0.72	0.98	0.51
2	SVM	0.51	0.38	0.84	0.37
3	SVM	0.51	0.98	0.98	0.90
0	RNA	0.50	0.00	0.65	0.00
1	RNA	0.50	0.77	0.68	0.51
2	RNA	0.50	0.31	0.83	0.34
3	RNA	0.50	1.00	0.97	0.87

FONTE: De autoria própria (2024)

TABELA 10 – DESEMPENHO DOS MODELOS COM CARACTERÍSTICAS VGG16

Desempenho dos Modelos com Características VGG16

Classe	Modelo	Acurácia	Sensibilidade	Especificidade	F1-Score
0	SVM	0.91	0.97	0.92	0.93
1	SVM	0.91	0.91	0.88	0.90
2	SVM	0.91	0.84	0.83	0.90
3	SVM	0.91	0.92	0.97	0.93
0	Random Forest	0.87	0.95	0.85	0.91
1	Random Forest	0.87	0.83	0.88	0.88
2	Random Forest	0.87	0.81	0.85	0.82
3	Random Forest	0.87	0.86	0.93	0.84
0	RNA	0.89	0.94	0.93	0.89
1	RNA	0.89	0.79	0.88	0.82
2	RNA	0.89	0.90	0.96	0.91
3	RNA	0.89	0.93	0.96	0.95

FONTE: De autoria própria (2024)

Melhor Modelo por Característica

LBP: O Random Forest apresentou uma Acurácia de 66% com um F1-Score elevado na classe 3 (0.97).

VGG16: O SVM teve a melhor Acurácia de 91% com um F1-Score alto em todas as classes.

Conclusão

O modelo SVM com características VGG16 teve o melhor desempenho geral, superando o Random Forest e o RNA em termos de Acurácia e F1-Score, evidenciando sua eficácia na classificação.

Questão 2

```
from google.colab import drive
drive.mount('/content/drive')


# file_path = '/content/drive/My Drive/VisaoComputacional/Train_Warwick.zip'
file_path = '/content/drive/My Drive/VC/Train_Warwick.zip'
import zipfile
with zipfile.ZipFile(file_path, 'r') as zip_ref:
    zip_ref.extractall('/content/')


#file_path = '/content/drive/My Drive/VisaoComputacional/Test_Warwick.zip'
file_path = '/content/drive/My Drive/VC/Test_Warwick.zip'
import zipfile
```

```
with zipfile.ZipFile(file_path, 'r') as zip_ref:
    zip_ref.extractall('/content/')

"""# Importando as bibliotecas necessárias"""

import os
import numpy as np
import cv2
import pandas as pd
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from skimage import feature

from keras.applications import VGG16
from keras.layers import GlobalAveragePooling2D, Dense
from keras.models import Model
from keras.preprocessing.image import load_img, img_to_array

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report

from keras.utils import to_categorical
from keras.layers import Flatten
from tensorflow.keras.layers import Flatten

# Avaliar o desempenho dos modelos
from sklearn.metrics import confusion_matrix, classification_report

!pip install livelossplot

"""Tarefa 1: Carregar a Base de Dados de Treino"""

# Função para carregar as imagens e rótulos
def carregar_imagens(diretorio):
#def carregar_imagens(diretorio,target_size=(224, 224)):
    imagens = []
    rotulos = []
    pacientes = []
```

```
    for subdir, dirs, files in os.walk(diretorio):
        # Extrair o rótulo da classe a partir do nome da pasta
        classe = os.path.basename(subdir)  # Nome da pasta atual

        for file in files:
            if file.endswith('.png'):
                # Extrair ID do paciente
                partes = file.split('_')
                id_paciente = partes[0]  # Parte XX (ID do paciente)
                # Adicionar o rótulo da classe
                rotulo = classe  # A classe é o nome da pasta

                # Carregar a imagem com cv2
                img = cv2.imread(os.path.join(subdir, file))
                 # Redimensionar imagem >>>> VERIFICAR SE É NECESSÁRIO NESTE
TRECHO
                #img_resized = cv2.resize(img, target_size=(224, 224))

                img_array = img.astype('float32')  # Converter para float32

                # Adicionar aos arrays
                imagens.append(img_array)
                #imagens.append(img_resized.astype('float32'))
                rotulos.append(rotulo)  # Adiciona o rótulo da classe
                pacientes.append(id_paciente)

    return np.array(imagens), np.array(rotulos), np.array(pacientes)

# Diretório onde as imagens foram extraídas
train_dir = '/content/Train_4cls_amostra'

# Carregar as imagens de treino
X, y, pacientes = carregar_imagens(train_dir)

# Verificar se as imagens foram carregadas corretamente
print(f"Número de imagens carregadas: {len(X)}")
print(f"Número de rótulos carregados: {len(y)}")
print(f"Número de pacientes: {len(np.unique(pacientes))}")

# Codificar os rótulos de classe em números
```

```
label_encoder = LabelEncoder()
y_treino_encoded = label_encoder.fit_transform(y)  # Transforma as classes em números

# Verificação do tipo e dimensões dos dados
print(f"Formato de X: {X.shape}")  # Esperado: (n_imagens, 224, 224, 3)
y_treino_encoded = label_encoder.fit_transform(y)  # Transforma as classes em números
print(f"Formato de y_treino_encoded: {y_treino_encoded.shape}")  # Esperado: (n_imagens,)

"""Tarefa 2: Criar Partições de Treino e Validação"""

# Função para dividir manualmente os dados de treino e validação por pacientes, garantindo a presença de todas as classes
def dividir_por_paciente_manual(X, y, pacientes, proporcao_treino=0.8, n_classes=4):

    idx_treino = []  # Lista para armazenar os índices do conjunto de treino
    idx_val = []     # Lista para armazenar os índices do conjunto de validação

    # Para cada classe (0, 1, 2, ..., n_classes-1)
    for classe in range(n_classes):
        # Obter os índices das imagens que pertencem à classe atual
        idx_classe = np.where(y == classe)[0]

        # Obter os IDs únicos dos pacientes que possuem imagens nesta classe
        pacientes_classe = np.unique(pacientes[idx_classe])

        # Determinar o número de pacientes que irão para o conjunto de treino
        n_pacientes_treino = int(len(pacientes_classe) * proporcao_treino)

        # Dividir os pacientes dessa classe entre treino e validação
         pacientes_treino_classe = pacientes_classe[:n_pacientes_treino]  # Primeira parte vai para treino
        pacientes_val_classe = pacientes_classe[n_pacientes_treino:]     # O restante vai para validação

        # Identificar os índices no array original de pacientes que pertencem ao conjunto de treino e validação
```

```
        idx_treino_classe = np.isin(pacientes, pacientes_treino_classe)  # Índices de treino para a classe atual
        idx_val_classe = np.isin(pacientes, pacientes_val_classe)        # Índices de validação para a classe atual

        # Adicionar esses índices às listas principais
        idx_treino.extend(np.where(idx_treino_classe)[0])
        idx_val.extend(np.where(idx_val_classe)[0])

    # Dividir os dados de entrada (X) e rótulos (y) com base nos índices de treino e validação
    X_treino, X_val = X[idx_treino], X[idx_val]
    y_treino, y_val = y[idx_treino], y[idx_val]

    # Separar os IDs dos pacientes correspondentes aos conjuntos de treino e validação
    pacientes_treino = pacientes[idx_treino]
    pacientes_val = pacientes[idx_val]

    # Retornar os dados divididos
    return X_treino, X_val, y_treino, y_val, pacientes_treino, pacientes_val

# Realizar a divisão manual
X_treino, X_val, y_treino, y_val, pacientes_treino, pacientes_val = dividir_por_paciente_manual(X, y_treino_encoded, pacientes)

# Verificar a divisão
print(f"Imagens de treino: {len(X_treino)}, Imagens de validação: {len(X_val)}")
print(f"Pacientes de treino: {np.unique(pacientes_treino)}, Pacientes de validação: {np.unique(pacientes_val)}")

# Verificar se há interseção entre os pacientes
pacientes_em_comum = np.intersect1d(pacientes_treino, pacientes_val)

# Mostrar os resultados
if len(pacientes_em_comum) == 0:
     print("Não há pacientes repetidos entre os conjuntos de treino e validação.")
else:
```

```
      print(f"Os seguintes pacientes estão em ambos os conjuntos:
{pacientes_em_comum}")

# Verificar a distribuição de classes nos conjuntos de treino e validação
def verificar_distribuicao_classes(y_treino, y_val, n_classes=4):
    # Contar quantas amostras existem de cada classe no conjunto de treino
    for classe in range(n_classes):
        count_treino = np.sum(y_treino == classe)
        count_val = np.sum(y_val == classe)
       print(f"Classe {classe}: {count_treino} imagens no treino, {count_val}
imagens na validação.")

    # Verificar se todas as classes estão presentes
    classes_treino = np.unique(y_treino)
    classes_val = np.unique(y_val)

    if len(classes_treino) == n_classes and len(classes_val) == n_classes:
         print("Todas as classes estão presentes nos conjuntos de treino e
validação.")
    else:
           print("Alguma classe está faltando no conjunto de treino ou
validação.")

# Chamar a função para verificar a distribuição das classes
verificar_distribuicao_classes(y_treino, y_val, n_classes=4)

# Função para redimensionar um conjunto de imagens
def redimensionar_imagens(imagens, target_size=(224, 224)):
    imagens_redimensionadas = []

    for img in imagens:
        # Redimensionar a imagem para o tamanho alvo usando cv2
        img_resized = cv2.resize(img, target_size)
        imagens_redimensionadas.append(img_resized)

    # Retornar as imagens redimensionadas como um array NumPy
    return np.array(imagens_redimensionadas)

# Redimensionar imagens
X_treino_redimensionado = redimensionar_imagens(X_treino, target_size=(224,
224))
```

```
X_val_redimensionado = redimensionar_imagens(X_val, target_size=(224, 224))

### VERIFICAR
#X_teste_redimensionado = redimensionar_imagens(X_teste, target_size=(224,
224))

print("Formato de X_treino_redimensionado:", X_treino_redimensionado.shape)
print("Formato de X_val_redimensionado:", X_val_redimensionado.shape)
#print("Formato de X_teste_redimensionado:", X_teste_redimensionado.shape)

# Diretório onde estão as imagens de teste
test_dir = '/content/Test_4cl_amostra'

# Carregar as imagens de TESTE
X_teste, y_teste, pacientes_teste = carregar_imagens(test_dir)

# Codificar os rótulos de classe em números
label_encoder = LabelEncoder()

### ACHO QUE NÃO É NECESSÁRIO ESSA CONVERSÃO PARA OS DADOS DE TESTE
#y_teste_encoded = label_encoder.fit_transform(y_teste)  # Transforma as
classes em números

#### REDIMENSIONAR IMAGENS DE TESTE
X_teste_redimensionado = redimensionar_imagens(X_teste, target_size=(224,
224))

"""DATA AUGMENTATION"""

### parâmetros da aula 21

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import preprocess_input

# Criar geradores de treino e validação a partir dos arrays X e y
train_generator = ImageDataGenerator(
                                    rescale=1./255,
                                   rotation_range=90,  # Rotação das imagens
em até 90 graus
                                     brightness_range=[0.1, 0.7],  # Ajuste
do brilho
```

```
                                        width_shift_range=0.5,  # Deslocamento
horizontal
                                       height_shift_range=0.5,  # Deslocamento
vertical
                                            horizontal_flip=True,  # Inversão
horizontal
                                     vertical_flip=True,  # Inversão vertical
                                    # validation_split=0.2,  # Não necessário,
pois a validação já está separada
                                  preprocessing_function=preprocess_input)  #
Função de normalização

test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)

#### ESTE CÓDIGO É NECESSÁRIO PARA A CORRETA CONVERSÃO

y_treino_one_hot = to_categorical(y_treino, num_classes=4)
y_val_one_hot = to_categorical(y_val, num_classes=4)

#### VERIFICAR SE É NECESSÁRIO
#y_teste_one_hot = to_categorical(y_teste, num_classes=4)

#BATCH_SIZE = 64
BATCH_SIZE = 32

# Criar geradores de treino e validação a partir dos arrays X e y
traingen = train_generator.flow(X_treino_redimensionado, y_treino_one_hot,
batch_size=BATCH_SIZE, shuffle=True, seed=42)
validgen   =   train_generator.flow(X_val_redimensionado,   y_val_one_hot,
batch_size=BATCH_SIZE, shuffle=False, seed=42)

testgen = test_generator.flow(X_teste_redimensionado, batch_size=BATCH_SIZE,
shuffle=False, seed=42)

"""# RESNET50"""

from tensorflow.keras.applications.resnet50 import ResNet50

### O EXERCÍCIO PEDE PARA USAR PESOS PRE-TREINADOS
resnet_0                =                ResNet50(input_shape=(224,224,3),
weights='imagenet',  include_top=False)
```

```
# treinar os pesos existentes
for layer in resnet_0.layers:
  layer.trainable = False

"""Camadas Fully Conect"""

from keras.layers import Flatten
from tensorflow.keras.layers import Flatten

# camadas próprias - você pode colocar mais se quiser
# A saída da resnet será a entrada da camada criada
x = Flatten()(resnet_0.output)

# camada de classificação com as 04 classes utilizadas
prediction = Dense(4, activation='softmax')(x)

# Criação do Objeto Modelo
model_0 = Model(inputs=resnet_0.input, outputs=prediction)

model_0.summary() # Impressão das arquitetura da rede

"""# TREINAMENTO PARA RESNET COM DATA AUGMENTATION"""

# Commented out IPython magic to ensure Python compatibility.
#
# %%time
# # Otimizador propagação da Raiz quadrada da média ao quadrado (Root Mean
Squared Propagation)
# from keras.optimizers import RMSprop
# from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
# from livelossplot import PlotLossesKeras
# from tensorflow.keras.utils import to_categorical
#
# ### SAMPLES não é necessário para flow
#
# steps_per_epoch = len(X_treino_redimensionado) // BATCH_SIZE
# val_steps = len(X_val_redimensionado) // BATCH_SIZE
#
# # TESTE DO NÚMERO DE DIMENSÕES
# # Verificar as dimensões originais de y_treino
```

```
# print("Dimensões originais de y_treino:", y_treino.shape)
# print("Primeiros valores de y_treino:", y_treino[:5])
#
# # Transformar os rótulos em one-hot encoding apenas se ainda não estiverem
nesse formato
# if len(y_treino.shape) == 1:  # Se for um vetor unidimensional
#     y_treino = to_categorical(y_treino, num_classes=4)
# if len(y_val.shape) == 1:  # Se for um vetor unidimensional
#     y_val = to_categorical(y_val, num_classes=4)
#
#
# n_epochs = 10
#
# optimizer = RMSprop(learning_rate=0.0001)
#
#   model_0.compile(loss='categorical_crossentropy',   optimizer=optimizer,
metrics=['accuracy'])
#
# # Treinamento do Modelo
#
# # Salva o modelo Keras após cada época, porém só o de melhor resultado
# checkpointer = ModelCheckpoint(filepath='img_model_0.weights.best.keras',
#                                verbose=1,
#                                save_best_only=True)
#
#
#
# # Para o treinamento para prevenir o overfitting
# # Não utilizei aqui, pois queria que rodasse todas as 30 épocas
# early_stop = EarlyStopping(monitor='val_loss',
#                            patience=10,
#                            restore_best_weights=True,
#                            mode='min')
#
# history_0 = model_0.fit(traingen,
#                         epochs=n_epochs,
#                         steps_per_epoch=steps_per_epoch,
#                         validation_data=validgen,
#                         validation_steps=val_steps,
#                         verbose=True)
```

```
"""# TREINAMENTO PARA RESNET SEM DATA AUGMENTATION"""

#### RESNET SEM DATA AUGMENTATION

resnet_no_aug              =              ResNet50(input_shape=(224,224,3),
weights='imagenet',  include_top=False)

# treinar os pesos existentes
for layer in resnet_no_aug.layers:
  layer.trainable = False

# camadas próprias - você pode colocar mais se quiser
# A saída da resnet será a entrada da camada criada
x_no_aug = Flatten()(resnet_no_aug.output)

# camada de classificação com as 04 classes utilizadas
prediction = Dense(4, activation='softmax')(x_no_aug)

# Criação do Objeto Modelo
model_no_aug = Model(inputs=resnet_no_aug.input, outputs=prediction)

model_no_aug.summary() # Impressão das arquitetura da rede

# Commented out IPython magic to ensure Python compatibility.
# #### TREINAMENTO SEM DATA AUGMENTAION
#
# %%time
# # Otimizador propagação da Raiz quadrada da média ao quadrado (Root Mean
Squared Propagation)
# from keras.optimizers import RMSprop
# from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
# from livelossplot import PlotLossesKeras
# from tensorflow.keras.utils import to_categorical
#
# ### SAMPLES não é necessário para flow
#
# steps_per_epoch = len(X_treino_redimensionado) // BATCH_SIZE
# val_steps = len(X_val_redimensionado) // BATCH_SIZE
#
# # Transformar os rótulos em one-hot encoding apenas se ainda não estiverem
nesse formato
```

```
# if len(y_treino.shape) == 1:  # Se for um vetor unidimensional
#     y_treino = to_categorical(y_treino, num_classes=4)
# if len(y_val.shape) == 1:  # Se for um vetor unidimensional
#     y_val = to_categorical(y_val, num_classes=4)
#
#
# n_epochs = 10
#
# optimizer = RMSprop(learning_rate=0.0001)
#
# model_no_aug.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
#
# # Treinamento do Modelo
#
# # Salva o modelo Keras após cada época, porém só o de melhor resultado
#                                 checkpointer                                =
ModelCheckpoint(filepath='img_model_no_aug.weights.best.keras',
#                                  verbose=1,
#                                  save_best_only=True)
#
#
#
# # Para o treinamento para prevenir o overfitting
# # Não utilizei aqui, pois queria que rodasse todas as 30 épocas
# early_stop = EarlyStopping(monitor='val_loss',
#                            patience=10,
#                            restore_best_weights=True,
#                            mode='min')
#
#
# ### TREINAMENTO SEM DATA AUGMENTATION
#
# ### "GERADORES DE IMAGEM" SEM DATA AUGMENTATION
#                          no_aug_train_generator                             =
ImageDataGenerator(preprocessing_function=preprocess_input).flow(
#     x=np.array(X_treino_redimensionado),
#     y=y_treino_one_hot,
#     batch_size=BATCH_SIZE,
#     shuffle=True,
#     seed=42
```

```
# )
#
#                            no_aug_val_generator                                =
ImageDataGenerator(preprocessing_function=preprocess_input).flow(
#     x=np.array(X_val_redimensionado),
#     y=y_val_one_hot,
#     batch_size=BATCH_SIZE,
#     shuffle=True,
#     seed=42
# )
#
# history_no_aug = model_no_aug.fit(no_aug_train_generator,
#                           epochs=n_epochs,
#                           steps_per_epoch=steps_per_epoch,
#                           validation_data=no_aug_val_generator,
#                           validation_steps=val_steps,
#                           verbose=True)
#

from tensorflow.keras.layers import AveragePooling2D

"""#### VGG16 COM DATA AUGMENTATION"""

### O EXERCÍCIO PEDE PARA USAR PESOS PRE-TREINADOS
vgg_aug                    =                    VGG16(input_shape=(224,224,3),
weights='imagenet',  include_top=False)

# treinar os pesos existentes
for layer in vgg_aug.layers:
  layer.trainable = False

# Camadas Fully Conect

# camadas próprias - você pode colocar mais se quiser
# A saída da resnet será a entrada da camada criada
x_vgg = Flatten()(vgg_aug.output)

# camada de classificação com as 04 classes utilizadas
prediction = Dense(4, activation='softmax')(x_vgg)

# Criação do Objeto Modelo
```

```
vgg_aug = Model(inputs=vgg_aug.input, outputs=prediction)

vgg_aug.summary() # Impressão das arquitetura da rede

# Commented out IPython magic to ensure Python compatibility.
#
# %%time
# # Otimizador propagação da Raiz quadrada da média ao quadrado (Root Mean
Squared Propagation)
# from keras.optimizers import RMSprop
# from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
# from livelossplot import PlotLossesKeras
# from tensorflow.keras.utils import to_categorical
#
# ### SAMPLES não é necessário para flow
#
# steps_per_epoch = len(X_treino_redimensionado) // BATCH_SIZE
# val_steps = len(X_val_redimensionado) // BATCH_SIZE
#
# # TESTE DO NÚMERO DE DIMENSÕES
# # Verificar as dimensões originais de y_treino
# print("Dimensões originais de y_treino:", y_treino.shape)
# print("Primeiros valores de y_treino:", y_treino[:5])
#
# # Transformar os rótulos em one-hot encoding apenas se ainda não estiverem
nesse formato
# if len(y_treino.shape) == 1:  # Se for um vetor unidimensional
#     y_treino = to_categorical(y_treino, num_classes=4)
# if len(y_val.shape) == 1:  # Se for um vetor unidimensional
#     y_val = to_categorical(y_val, num_classes=4)
#
#
# n_epochs = 10
#
# optimizer = RMSprop(learning_rate=0.0001)
#
#   vgg_aug.compile(loss='categorical_crossentropy',   optimizer=optimizer,
metrics=['accuracy'])
#
# # Treinamento do Modelo
#
```

```
# # Salva o modelo Keras após cada época, porém só o de melhor resultado
# checkpointer = ModelCheckpoint(filepath='img_vgg_aug.weights.best.keras',
#                                verbose=1,
#                                save_best_only=True)
#
#
#
# # Para o treinamento para prevenir o overfitting
# # Não utilizei aqui, pois queria que rodasse todas as 30 épocas
# early_stop = EarlyStopping(monitor='val_loss',
#                            patience=10,
#                            restore_best_weights=True,
#                            mode='min')
#
#
# history_0 = vgg_aug.fit(traingen,
#                         epochs=n_epochs,
#                         steps_per_epoch=steps_per_epoch,
#                         validation_data=validgen,
#                         validation_steps=val_steps,
#                         verbose=True)

"""### VGG SEM DATA AUGMENTATION"""

### O EXERCÍCIO PEDE PARA USAR PESOS PRE-TREINADOS
vgg_no_aug                  =                 VGG16(input_shape=(224,224,3),
weights='imagenet',  include_top=False)

# treinar os pesos existentes
for layer in vgg_no_aug.layers:
  layer.trainable = False

# Camadas Fully Conect

# camadas próprias - você pode colocar mais se quiser
# A saída da resnet será a entrada da camada criada
x_vgg_no_aug = Flatten()(vgg_no_aug.output)

# camada de classificação com as 04 classes utilizadas
prediction = Dense(4, activation='softmax')(x_vgg_no_aug)
```

```
# Criação do Objeto Modelo
vgg_no_aug = Model(inputs=vgg_no_aug.input, outputs=prediction)

# Commented out IPython magic to ensure Python compatibility.
# #### TREINAMENTO SEM DATA AUGMENTAION
#
# %%time
# # Otimizador propagação da Raiz quadrada da média ao quadrado (Root Mean
Squared Propagation)
# from keras.optimizers import RMSprop
# from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
# from livelossplot import PlotLossesKeras
# from tensorflow.keras.utils import to_categorical
#
# ### SAMPLES não é necessário para flow
#
# steps_per_epoch = len(X_treino_redimensionado) // BATCH_SIZE
# val_steps = len(X_val_redimensionado) // BATCH_SIZE
#
# # Transformar os rótulos em one-hot encoding apenas se ainda não estiverem
nesse formato
# if len(y_treino.shape) == 1:  # Se for um vetor unidimensional
#     y_treino = to_categorical(y_treino, num_classes=4)
# if len(y_val.shape) == 1:  # Se for um vetor unidimensional
#     y_val = to_categorical(y_val, num_classes=4)
#
#
# n_epochs = 10
#
# optimizer = RMSprop(learning_rate=0.0001)
#
#  vgg_no_aug.compile(loss='categorical_crossentropy',  optimizer=optimizer,
metrics=['accuracy'])
#
# # Treinamento do Modelo
#
# # Salva o modelo Keras após cada época, porém só o de melhor resultado
#                                checkpointer                                =
ModelCheckpoint(filepath='img_model_vgg_no_aug.weights.best.keras',
#                                verbose=1,
#                                save_best_only=True)
```

```
#
#
#
# # Para o treinamento para prevenir o overfitting
# early_stop = EarlyStopping(monitor='val_loss',
#                            patience=10,
#                            restore_best_weights=True,
#                            mode='min')
#
#
# ### TREINAMENTO SEM DATA AUGMENTATION
# ### JÁ FORAM DEFINIDOS ANTERIORMENTE, PODE NÃO SER NECESSÁRIO AQUI NOVAMENTE
# ### "GERADORES DE IMAGEM" SEM DATA AUGMENTATION
# """
#                          no_aug_train_generator                          =
ImageDataGenerator(preprocessing_function=preprocess_input).flow(
#     x=np.array(X_treino_redimensionado),
#     y=y_treino_one_hot,
#     batch_size=BATCH_SIZE,
#     shuffle=True,
#     seed=42
# )
#
#                            no_aug_val_generator                          =
ImageDataGenerator(preprocessing_function=preprocess_input).flow(
#     x=np.array(X_val_redimensionado),
#     y=y_val_one_hot,
#     batch_size=BATCH_SIZE,
#     shuffle=True,
#     seed=42
# )
# """
# history_vgg_no_aug = vgg_no_aug.fit(no_aug_train_generator,
#                          epochs=n_epochs,
#                          steps_per_epoch=steps_per_epoch,
#                          validation_data=no_aug_val_generator,
#                          validation_steps=val_steps,
#                          verbose=True)
#

"""#Predições dos modelos"""
```

```
#### PREDIÇÕES
# RESNET COM DATA AUGMENTATION
pred_resnet_aug = model_0.predict(testgen)
#print(pred_resnet_aug) # Depois excluir
pred_resnet_aug_class = np.argmax(pred_resnet_aug, axis=1)
#print("Classe prevista:", pred_resnet_aug_class)

# RESNET SEM DATA AUGMENTATION
pred_resnet_no_aug = model_no_aug.predict(testgen)
#print(pred_resnet_no_aug) # Depois excluir
pred_resnet_no_aug_class = np.argmax(pred_resnet_no_aug, axis=1)
#print("Classe prevista:", pred_resnet_no_aug_class)

# VGG COM DATA AUGMENTATION
pred_vgg_aug = vgg_aug.predict(testgen)
#print(pred_vgg_aug) # Depois excluir
pred_vgg_aug_class = np.argmax(pred_vgg_aug, axis=1)
#print("Classe prevista:", pred_vgg_aug_class)

# VGG SEM DATA AUGMENTATION
pred_vgg_no_aug = vgg_no_aug.predict(testgen)
#print(pred_vgg_no_aug) # Depois excluir
pred_vgg_no_aug_class = np.argmax(pred_vgg_no_aug, axis=1)
#print("Classe prevista:", pred_vgg_no_aug_class)

"""#Resultados"""

# IMPRIMIR ACURÁCIAS
from sklearn.metrics import accuracy_score

acc_resnet_aug = accuracy_score(y_teste.astype(int), pred_resnet_aug_class)
print("Acurácia     Modelo     ResNet50     COM     Data     Augmentation:
{:.2f}%".format(acc_resnet_aug * 100))

acc_resnet_no_aug           =           accuracy_score(y_teste.astype(int),
pred_resnet_no_aug_class)
print("Acurácia     Modelo     ResNet50     SEM     Data     Augmentation:
{:.2f}%".format(acc_resnet_no_aug * 100))

acc_vgg_aug = accuracy_score(y_teste.astype(int), pred_vgg_aug_class)
```

```
print("Acurácia Modelo VGG16 COM Data Augmentation:
{:.2f}%".format(acc_vgg_aug * 100))

acc_vgg_no_aug = accuracy_score(y_teste.astype(int), pred_vgg_no_aug_class)
print("Acurácia Modelo VGG16 SEM Data Augmentation:
{:.2f}%".format(acc_vgg_no_aug * 100))

#### Matriz de confusão

cm_resnet_aug = confusion_matrix(y_teste.astype(int), pred_resnet_aug_class)
cm_resnet_no_aug = confusion_matrix(y_teste.astype(int),
pred_resnet_no_aug_class)
cm_vgg_aug = confusion_matrix(y_teste.astype(int), pred_vgg_aug_class)
cm_vgg_no_aug = confusion_matrix(y_teste.astype(int), pred_vgg_no_aug_class)

#### Função para calcular métricas e imprimir os resultados
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report

# Função para calcular métricas
def calcular_metricas(cm, model_name):
    true_positive = np.diag(cm)
    false_positive = cm.sum(axis=0) - true_positive
    false_negative = cm.sum(axis=1) - true_positive
      true_negative = cm.sum() - (false_positive + false_negative +
true_positive)

    sensibilidade = true_positive / (true_positive + false_negative)
    especificidade = true_negative / (true_negative + false_positive)
     f1_scores = 2 * (sensibilidade * especificidade) / (sensibilidade +
especificidade)

    print(f"Matriz de Confusão para {model_name}:")
    print()
    print(cm)
    print()

    # Criação de um DataFrame para visualizar as métricas
    metrics_df = pd.DataFrame({
        'Classe': [f'Classe {i}' for i in range(len(sensibilidade))],
```

```
        'Sensibilidade': sensibilidade,
        'Especificidade': especificidade,
        'F1-Score': f1_scores
    })

    print(f"Métricas para {model_name}:")
    print(metrics_df.set_index('Classe'))
    print()  # Linha em branco para melhor legibilidade

    return sensibilidade, especificidade, f1_scores

#### CALCULANDO AS MÉTRICAS E IMPRIMINDO OS RESULTADOS
calcular_metricas(cm_resnet_aug, "ResNet50 COM Data Augmentation")
calcular_metricas(cm_resnet_no_aug, "ResNet50 SEM Data Augmentation")
calcular_metricas(cm_vgg_aug, "VGG16 COM Data Augmentation")
calcular_metricas(cm_vgg_no_aug, "VGG16 SEM Data Augmentation")
```

TABELA 11 – DESEMPENHO DOS MODELOS RESNET50 E VGG16

Desempenho dos Modelos RESNET50 e VGG16

Classe	Modelo	Acurácia	Sensibilidade	Especificidade	F1-Score
0	Resnet50 COM Data Augmentation	24.26%	0.059406	0.985185	0.112055
1	Resnet50 COM Data Augmentation	24.26%	0.022222	0.971530	0.043451
2	Resnet50 COM Data Augmentation	24.26%	0.155556	0.697509	0.254380
3	Resnet50 COM Data Augmentation	24.26%	0.944444	0.405694	0.567579
0	Resnet50 SEM Data Augmentation	81.13%	0.930693	1.000000	0.964103
1	Resnet50 SEM Data Augmentation	81.13%	0.700000	0.889680	0.783524
2	Resnet50 SEM Data Augmentation	81.13%	0.688889	0.882562	0.773791
3	Resnet50 SEM Data Augmentation	81.13%	0.933333	0.985765	0.958833
0	VGG16 COM Data Augmentation	44.47%	0.029703	1.000000	0.057692

1	VGG16 COM Data Augmentation	44.47%	0.000000	1.000000	0.000000
2	VGG16 COM Data Augmentation	44.47%	0.011111	1.000000	0.021978
3	VGG16 COM Data Augmentation	44.47%	1.000000	0.014235	0.028070
0	VGG16 SEM Data Augmentation	63.07%	0.811881	0.951852	0.876312
1	VGG16 SEM Data Augmentation	63.07%	0.466667	0.882562	0.610515
2	VGG16 SEM Data Augmentation	63.07%	0.466667	0.797153	0.588699
3	VGG16 SEM Data Augmentation	63.07%	0.833333	0.903915	0.867190

FONTE: De autoria própria (2024)

Conclusão

Com base nas métricas encontradas em todos os modelos treinados, o modelo ResNet50 SEM Data Augmentation é a melhor escolha. Ele combina uma alta acurácia com boas métricas de sensibilidade e F1-Score, sugerindo que ele generaliza bem e é eficaz em classificar corretamente as diferentes classes.

APÊNDICE K - ASPECTOS FILOSÓFICOS E ÉTICOS DA IA

A – ENUNCIADO

Título do Trabalho: "Estudo de Caso: Implicações Éticas do Uso do ChatGPT"

Trabalho em Grupo: O trabalho deverá ser realizado em grupo de alunos de no máximo seis (06) integrantes.

Objetivo do Trabalho: Investigar as implicações éticas do uso do ChatGPT em diferentes contextos e propor soluções responsáveis para lidar com esses dilemas.
Parâmetros para elaboração do Trabalho:

1. Relevância Ética: O trabalho deve abordar questões éticas significativas relacionadas ao uso da inteligência artificial, especialmente no contexto do ChatGPT. Os alunos devem identificar dilemas éticos relevantes e explorar como esses dilemas afetam diferentes partes interessadas, como usuários, desenvolvedores e a sociedade em geral.
2. Análise Crítica: Os alunos devem realizar uma análise crítica das implicações éticas do uso do ChatGPT em estudos de caso específicos. Eles devem examinar como o algoritmo pode influenciar a disseminação de informações, a privacidade dos usuários e a tomada de decisões éticas. Além disso, devem considerar possíveis vieses algorítmicos, discriminação e questões de responsabilidade.
3. Soluções Responsáveis: Além de identificar os desafios éticos, os alunos devem propor soluções responsáveis e éticas para lidar com esses dilemas. Isso pode incluir sugestões para políticas, regulamentações ou práticas de design que promovam o uso responsável da inteligência artificial. Eles devem considerar como essas soluções podem equilibrar os interesses de diferentes partes interessadas e promover valores éticos fundamentais, como transparência, justiça e privacidade.
4. Colaboração e Discussão: O trabalho deve envolver discussões em grupo e colaboração entre os alunos. Eles devem compartilhar ideias, debater diferentes pontos de vista e chegar a conclusões informadas através do diálogo e da reflexão mútua. O estudo de caso do ChatGPT pode servir como um ponto de partida para essas discussões, incentivando os alunos a aplicar conceitos éticos e legais aprendidos ao analisar um caso concreto.
5. Limite de Palavras: O trabalho terá um limite de 6 a 10 páginas teria aproximadamente entre 1500 e 3000 palavras.
6. Estruturação Adequada: O trabalho siga uma estrutura adequada, incluindo introdução, desenvolvimento e conclusão. Cada seção deve ocupar uma parte proporcional do total de páginas, com a introdução e a conclusão ocupando menos espaço do que o desenvolvimento.

7. Controle de Informações: Evitar incluir informações desnecessárias que possam aumentar o comprimento do trabalho sem contribuir significativamente para o conteúdo. Concentre-se em informações relevantes, argumentos sólidos e evidências importantes para apoiar sua análise.
8. Síntese e Clareza: O trabalho deverá ser conciso e claro em sua escrita. Evite repetições desnecessárias e redundâncias. Sintetize suas ideias e argumentos de forma eficaz para transmitir suas mensagens de maneira sucinta.
9. Formatação Adequada: O trabalho deverá ser apresentado nas normas da ABNT de acordo com as diretrizes fornecidas, incluindo margens, espaçamento, tamanho da fonte e estilo de citação. Deve-se seguir o seguinte template de arquivo: hfps://bibliotecas.ufpr.br/wp-content/uploads/2022/03/template-artigo-de-periodico.docx

B – RESOLUÇÃO

A inteligência artificial (IA) tem mudado muitos aspectos da vida moderna, desde a automação de processos industriais até a personalização de serviços ao consumidor e questões básicas do dia a dia. Uma das inovações mais notáveis é o desenvolvimento de modelos de linguagem avançados, como o ChatGPT da OpenAI. Esses modelos são capazes de gerar textos coerentes e relevantes, interagindo com os usuários de maneira cada vez mais sofisticada. No entanto, o uso crescente dessas tecnologias levanta importantes questões éticas que precisam ser cuidadosamente examinadas e tratadas.

Este estudo de caso tem como objetivo investigar as implicações éticas do uso do ChatGPT em diferentes contextos, explorando como esses desafios afetam desenvolvedores, usuários e a sociedade em geral. Abordaremos questões cruciais, como a privacidade dos dados dos usuários, o potencial de disseminação de desinformação, os vieses algorítmicos e a responsabilidade dos desenvolvedores. Além disso, discutiremos possíveis soluções para mitigar esses dilemas éticos e promover um uso mais responsável e transparente da IA.

A importância deste estudo está na necessidade de equilibrar o avanço tecnológico com a manutenção de princípios éticos fundamentais. À medida que a IA continua a evoluir, é essencial que seu desenvolvimento e uso sejam guiados por normas éticas rigorosas que protejam os direitos e a privacidade das pessoas, promovam a equidade e garantam a transparência. Com este trabalho, esperamos contribuir para um debate mais informado e construtivo sobre as melhores práticas no uso da IA, com foco específico no ChatGPT.

Relevância Ética

Os desafios éticos relacionados ao uso do ChatGPT são amplos e complexos. Alguns dos principais dilemas, percebemos que, incluem a privacidade dos usuários, a disseminação de desinformação e a discriminação algorítmica e questões relacionadas a dificuldade de avaliar as responsabilidades efeitos do seu uso e a necessidade de análise humana antes de usar as informações em outros contextos.

A privacidade dos dados é uma preocupação central, especialmente considerando que o ChatGPT processa grandes volumes de informações pessoais. De acordo com a Lei Geral de Proteção de Dados (LGPD) no Brasil, a privacidade e a proteção dos dados pessoais são direitos fundamentais (Brasil, 2018).

A disseminação de desinformação é outro problema crítico, que vem tomando grandes proporções com os avanços da tecnologia, uma vez que o ChatGPT pode gerar informações imprecisas ou enganosas e que são expostas de uma forma persuasiva, porém muitas vezes sem fontes consideradas confiáveis ou que possam ser checadas de forma independente.

Além disso, há preocupações sobre a discriminação algorítmica, onde vieses presentes nos dados de treinamento podem resultar em respostas prejudiciais ou injustas. Outra questão relevante, é necessidade de um debate maior na sociedade para elencar responsabilidades quando existe alguma consequência no uso das informações recebidas via ChatGPT.

Análise Crítica

Analisando alguns casos específicos, o uso do ChatGPT revelou várias implicações éticas significativas. Em um estudo recente, Floridi (2023) discute os desafios éticos emergentes relacionados ao uso de IA em geral, destacando a necessidade de frameworks robustos para lidar com a transparência e a responsabilidade.

No contexto brasileiro, a pesquisadora Virgínia Dignum (2020) ressalta que a IA deve ser desenvolvida e utilizada com responsabilidade social, promovendo a justiça e a inclusão.

Além disso, um relatório recente da OpenAI (2023) identificou riscos associados à geração de desinformação, onde o ChatGPT foi capaz de criar conteúdo falso de maneira convincente, potencialmente influenciando negativamente a opinião pública e decisões políticas.

No Brasil, a questão da desinformação é particularmente relevante, dada a recente onda de fake news que tem impactado processos eleitorais e a confiança pública nas instituições (Ruediger, 2023).

Os vieses algorítmicos representam um dos maiores desafios éticos no desenvolvimento e uso de inteligência artificial. Estes vieses podem surgir de diversas maneiras, incluindo dados de treinamento enviesados, design inadequado do algoritmo e falta de diversidade nas equipes de desenvolvimento. Esses vieses podem resultar em discriminação e injustiça, afetando negativamente grupos minoritários e perpetuando desigualdades existentes na sociedade.

Por exemplo, estudos demonstram que modelos de IA treinados em dados históricos podem reproduzir e até amplificar preconceitos presentes nos dados, resultando em decisões enviesadas (Bender et al., 2021).

No contexto do ChatGPT, esses vieses podem se manifestar na geração de respostas que reforçam estereótipos ou discriminam certos grupos, como aconteceu em um wokshop promovido pela Gradio, uma empresa especializada em testes de machine learning, onde preconceitos foram reproduzidos na maioria dos cenários de testes apresentados durante uma demonstração do uso do chatGPT (JOHNSON, 2021).

Para mitigar os vieses algorítmicos, é fundamental adotar práticas de design e desenvolvimento inclusivas, como a auditoria constante dos dados de treinamento e dos modelos, e a incorporação de diversas perspectivas na criação e avaliação dos algoritmos (Nascimento, 2021).

Ferramentas de auditoria de IA, como as desenvolvidas por Raji et al. (2020), são essenciais para identificar e corrigir esses vieses antes que causem impactos significativos.

A responsabilidade dos desenvolvedores de IA é um aspecto crucial da ética em tecnologia. Desenvolvedores têm o dever de garantir que seus sistemas sejam seguros, justos e transparentes. Isso inclui não apenas a criação de algoritmos éticos, mas também a implementação de práticas de monitoramento contínuo e a manutenção de um diálogo aberto com a sociedade sobre os usos e limitações dessas tecnologias.

Conforme destacado por Floridi (2023), a responsabilidade ética na IA envolve a necessidade de accountability, ou seja, a capacidade de os desenvolvedores e as empresas de tecnologia serem responsabilizados pelas consequências de seus sistemas.

No caso do ChatGPT, isso pode incluir garantir que as respostas geradas sejam verificáveis e que os dados dos usuários sejam tratados com o máximo respeito à privacidade e segurança (Silva & Souza, 2021).

Além disso, os desenvolvedores devem promover a transparência em seus processos, permitindo que usuários e reguladores compreendam como as decisões são tomadas pelos algoritmos.

Iniciativas como a criação de comitês de ética e a publicação de relatórios de impacto de IA são passos importantes para assegurar essa responsabilidade (Dignum, 2020). Os autores Rossetti e Angeluci (2021) também destacam a necessidade e importância da transparência algorítmica para possibilitar a correção dos processos que envolvem a compressão a avaliação das informações pelos algoritmos.

Soluções Responsáveis

Para abordar esses dilemas éticos, é importante se pensar em implementar soluções responsáveis que garantam a transparência, justiça e privacidade. Algumas das propostas incluem a adoção de políticas rigorosas de privacidade de dados, conforme estabelecido pela LGPD, o desenvolvimento de algoritmos de mitigação de vieses e a implementação de práticas de design responsável que priorizem a ética. Sugere-se também a criação de regulamentos específicos que guiem o desenvolvimento e uso da IA, como o regulamento da União Europeia para IA, que estabelece diretrizes claras para a transparência e a responsabilização no uso de tecnologias de IA.

No Brasil, a Estratégia Brasileira de Inteligência Artificial (EBIA) destaca a importância de desenvolver IA com responsabilidade, ética e respeito aos direitos humanos (Ministério da Ciência, Tecnologia e Inovações, 2023).

Colaboração e Discussão

A colaboração entre desenvolvedores, pesquisadores, formuladores de políticas e a sociedade em geral é crucial para abordar as questões éticas associadas ao ChatGPT. Discussões abertas e inclusivas permitem a troca de diferentes perspectivas, enriquecendo o debate e promovendo soluções mais equilibradas e justas. A reflexão contínua e a adaptação das práticas de desenvolvimento de IA são necessárias para garantir que essas tecnologias beneficiem a sociedade como um todo.

Propostas de Ação Concreta para Garantir a Privacidade e Ética no Uso do ChatGPT.
1. Criação de Ferramentas de Auditoria e Transparência: Uma proposta concreta é o desenvolvimento de ferramentas de auditoria que permitam monitorar e verificar o comportamento dos sistemas de IA, como o ChatGPT. Essas ferramentas poderiam incluir:

Auditoria Algorítmica: Ferramentas que permitam a análise dos algoritmos utilizados pelo ChatGPT para identificar e corrigir vieses e discriminações. Essas ferramentas poderiam ser disponibilizadas como software open source para promover a transparência e a colaboração entre diferentes partes interessadas.

Transparência de Dados: Plataformas que permitam aos usuários visualizar e controlar os dados que são coletados e utilizados pelos modelos de IA. Isso poderia incluir dashboards que mostram quais dados estão sendo usados e como, além de opções para que os usuários possam editar ou deletar suas informações.

2. Implementação de Políticas de Privacidade Rigorosas:
Para garantir a privacidade dos usuários, é essencial implementar políticas de privacidade rigorosas e assegurar que estas sejam comunicadas de maneira clara e acessível aos usuários. Medidas específicas podem incluir:

Consentimento Informado: Garantir que os usuários forneçam consentimento explícito e informado antes que seus dados sejam coletados ou utilizados. Isso pode ser feito através de interfaces amigáveis que expliquem de maneira clara como os dados serão usados.

Anonimização de Dados: Utilização de técnicas avançadas de anonimização e pseudonimização para proteger a identidade dos usuários. Isso ajuda a minimizar os riscos associados ao vazamento de dados pessoais.

Dados sensíveis: Desenvolver/aperfeiçoar ferramentas que consigam identificar quando vários dados sensíveis são utilizados e assim evitar que esses dados sejam armazenados ou mesmo processados.

3. Desenvolvimento de Práticas de Design Responsável:
A integração de princípios éticos no processo de design e desenvolvimento de IA pode ajudar a mitigar riscos e promover o uso responsável da tecnologia. Isso pode incluir:

Design Centrado no Usuário: Adotar práticas de design que priorizem as necessidades e direitos dos usuários, garantindo que os sistemas de IA sejam seguros e respeitosos da privacidade.

Inclusão e Diversidade: Envolver diversos grupos de stakeholders no processo de design e desenvolvimento para garantir que diferentes perspectivas sejam consideradas e que os sistemas de IA sejam inclusivos e justos.

4. Fomento à Educação e Conscientização:
Promover a educação e a conscientização sobre as implicações éticas da IA é fundamental para criar uma cultura de responsabilidade e ética. Isso pode ser alcançado através de:

Programas de Capacitação: Oferecer programas de capacitação e treinamento para desenvolvedores, formuladores de políticas e o público em geral sobre os princípios éticos da IA e as melhores práticas para seu desenvolvimento e uso.

Campanhas de Conscientização: Lançar campanhas de conscientização pública sobre os direitos dos usuários e as implicações éticas da IA incentivando uma participação ativa e informada da sociedade no debate sobre essas tecnologias.

O estudo das implicações éticas do uso do ChatGPT mostra a necessidade urgente de abordagens responsáveis na implementação de tecnologias de inteligência artificial. É essencial identificar e resolver dilemas éticos, como privacidade, disseminação de desinformação e discriminação algorítmica, para garantir que o desenvolvimento e uso da IA estejam alinhados com valores de justiça, transparência e respeito aos direitos humanos.

Propostas concretas, como a criação de ferramentas de auditoria e transparência, a implementação de políticas de privacidade rigorosas, o desenvolvimento de práticas de design responsável e a promoção da educação e conscientização, são fundamentais para garantir um uso ético e responsável da IA. A adoção dessas medidas pode ajudar a mitigar riscos, promover a confiança dos usuários e assegurar que a IA seja utilizada de maneira justa e benéfica.

Além disso, é crucial promover a colaboração entre desenvolvedores, pesquisadores, formuladores de políticas e a sociedade em geral. Discussões abertas permitem a troca de diferentes perspectivas, enriquecendo o debate e promovendo soluções mais equilibradas e justas. A reflexão contínua e a adaptação das práticas de desenvolvimento de IA são necessárias para garantir que essas tecnologias beneficiem a sociedade como um todo.

Portanto, integrar princípios éticos no desenvolvimento e uso do ChatGPT e outras tecnologias de IA não é apenas necessário, mas uma responsabilidade coletiva. Somente através de uma abordagem ética e colaborativa podemos garantir que a inteligência artificial seja uma força positiva na sociedade, promovendo o bem-estar e respeitando os direitos de todos.

APÊNDICE L - GESTÃO DE PROJETOS DE IA

A – ENUNCIADO

1 Objetivo

Individualmente, ler e resumir – seguindo o *template* fornecido – **um** dos artigos abaixo:

AHMAD, L.; ABDELRAZEK, M.; ARORA, C.; BANO, M; GRUNDY, J. Requirements practices and gaps when engineering human-centered Artificial Intelligence systems. Applied Soft Computing. 143. 2023. DOI https://doi.org/10.1016/j.asoc.2023.110421

NAZIR, R.; BUCAIONI, A.; PELLICCIONE, P.; Architecting ML-enabled systems: Challenges, best practices, and design decisions. The Journal of Systems & Software. 207. 2024. DOI https://doi.org/10.1016/j.jss.2023.111860

SERBAN, A.; BLOM, K.; HOOS, H.; VISSER, J. Software engineering practices for machine learning – Adoption, effects, and team assessment. The Journal of Systems & Software. 209. 2024. DOI https://doi.org/10.1016/j.jss.2023.111907

STEIDL, M.; FELDERER, M.; RAMLER, R. The pipeline for continuous development of artificial intelligence models – Current state of research and practice. The Journal of Systems & Software. 199. 2023. DOI https://doi.org/10.1016/j.jss.2023.111615

XIN, D.; WU, E. Y.; LEE, D. J.; SALEHI, N.; PARAMESWARAN, A. Whither AutoML? Understanding the Role of Automation in Machine Learning Workflows. In CHI Conference on Human Factors in Computing Systems (CHI'21), Maio 8-13, 2021, Yokohama, Japão. DOI https://doi.org/10.1145/3411764.3445306

2 Orientações adicionais

Escolha o artigo que for mais interessante para você. Utilize tradutores e o Chat GPT para entender o conteúdo dos artigos – caso precise, mas escreva o resumo em língua portuguesa e nas suas palavras.

Não esqueça de preencher, no trabalho, os campos relativos ao seu nome e ao artigo escolhido.

No *template*, você deverá responder às seguintes questões:

- Qual o objetivo do estudo descrito pelo artigo?
- Qual o problema/oportunidade/situação que levou a necessidade de realização deste estudo?
- Qual a metodologia que os autores usaram para obter e analisar as informações do estudo?

- Quais os principais resultados obtidos pelo estudo?

Responda cada questão utilizando o espaço fornecido no *template*, sem alteração do tamanho da fonte (Times New Roman, 10), nem alteração do espaçamento entre linhas (1.0).

Não altere as questões do template.

Utilize o editor de textos de sua preferência para preencher as respostas, mas entregue o trabalho em PDF.

B – RESOLUÇÃO

QUADRO 1 – UNDERSTANDING THE ROLE OF AUTOMATION IN MACHINE LEARNING WORKFLOWS

Whither AutoML? Understanding the Role of Automation in Machine Learning Workflows			
Qual o objetivo do estudo descrito pelo artigo?	Qual o problema/oportunidade/situação que levou à necessidade de realização desse estudo?	Qual a metodologia que os autores usaram para obter e analisar as informações do estudo?	Quais os principais resultados obtidos pelo estudo?
O objetivo do estudo é compreender como as ferramentas de AutoML (Automated Machine Learning) são aplicadas na prática	A necessidade de realizar este estudo surge da rápida ascensão e diversificação das ferramentas de AutoML, que prometem	Os autores conduziram um estudo qualitativo baseado em entrevistas semiestruturadas com 16 participantes de diversos níveis de	O estudo revelou que, embora as ferramentas de AutoML tragam benefícios claros, como a redução de tempo de desenvolvimento e a

por diferentes perfis de usuários e analisar seus benefícios e limitações. Os autores buscam identificar o papel que a automação deve desempenhar nos fluxos de trabalho de aprendizado de máquina, destacando a importância da interação entre humanos e sistemas automatizados. Além disso, o estudo propõe recomendações de design para o desenvolvimento de futuras ferramentas de AutoML, com o foco em maximizar a colaboração e a eficiência.	facilitar e democratizar o uso de aprendizado de máquina, mas que enfrentam desafios em termos de transparência, controle e aplicabilidade em diferentes contextos. Muitos usuários relatam que, embora as ferramentas automatizadas agilizem processos, elas não substituem a intuição e o conhecimento contextual humano. Portanto, o estudo busca explorar como equilibrar a automação com a intervenção humana para otimizar resultados.	experiência e áreas de atuação. As entrevistas foram realizadas entre outubro de 2019 e março de 2020, tanto presencialmente quanto de forma remota. Os dados foram analisados por meio de codificação indutiva, permitindo que os pesquisadores extraíssem temas e padrões recorrentes nas práticas e percepções dos usuários. A análise colaborativa entre os autores ajudou a consolidar as categorias e garantir a profundidade na compreensão dos desafios	democratização do uso de aprendizado de máquina por usuários menos experientes, elas ainda apresentam deficiências importantes. Entre as principais limitações destacam-se a falta de transparência e a dificuldade de personalização, fatores que podem comprometer a confiança e a eficácia em cenários de aplicação complexos. Os autores também enfatizaram que o papel humano é indispensável em muitas etapas do processo, como a interpretação dos

		enfrentados e das oportunidades percebidas no uso do AutoML.	resultados, ajuste de parâmetros e supervisão geral. Por fim, o estudo sugere que o futuro do AutoML deve priorizar uma abordagem colaborativa, onde a automação complementa e potencializa a expertise humana, em vez de tentar substitui-la completamente.

FONTE: De autoria própria (2024)

APÊNDICE M - FRAMEWORKS DE INTELIGÊNCIA ARTIFICIAL

A – ENUNCIADO

1 Classificação (RNA)

Implementar o exemplo de Classificação usando a base de dados Fashion MNIST e a arquitetura RNA vista na aula **FRA - Aula 10 - 2.4 Resolução de exercício de RNA - Classificação**. Além disso, fazer uma breve explicação dos seguintes resultados:

- Gráficos de perda e de acurácia;
- Imagem gerada na seção "**Mostrar algumas classificações erradas"**, apresentada na aula prática.
  Informações:

- **Base de dados:** Fashion MNIST Dataset
- **Descrição**: Um dataset de imagens de roupas, onde o objetivo é classificar o tipo de vestuário. É semelhante ao famoso dataset MNIST, mas com peças de vestuário em vez de dígitos.
- **Tamanho**: 70.000 amostras, 784 features (28x28 pixels).
- **Importação do dataset**: Copiar código abaixo.

```
data = tf.keras.datasets.fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

2 Regressão (RNA)

Implementar o exemplo de Classificação usando a base de dados Wine Dataset e a arquitetura RNA vista na aula **FRA - Aula 12 - 2.5 Resolução de exercício de RNA - Regressão**. Além disso, fazer uma breve explicação dos seguintes resultados:

- Gráficos de avaliação do modelo (loss);
- Métricas de avaliação do modelo (pelo menos uma entre MAE, MSE, R^2).

Informações:

- **Base de dados:** Wine Quality
- **Descrição**: O objetivo deste dataset prever a qualidade dos vinhos com base em suas características químicas. A variável target (y) neste exemplo será o score de qualidade do vinho, que varia de 0 (pior qualidade) a 10 (melhor qualidade)
- **Tamanho**: 1599 amostras, 12 features.
- **Importação**: Copiar código abaixo.

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
```

```
data = pd.read_csv(url, delimiter=';')
```

```
Dica 1. Para facilitar o trabalho, renomeie o nome das colunas para português, dessa forma:
```

```
data.columns = [
    'acidez_fixa',            # fixed acidity
    'acidez_volatil',         # volatile acidity
    'acido_citrico',          # citric acid
    'acucar_residual',        # residual sugar
    'cloretos',               # chlorides
    'dioxido_de_enxofre_livre', # free sulfur dioxide
    'dioxido_de_enxofre_total', # total sulfur dioxide
    'densidade',              # density
    'pH',                     # pH
    'sulfatos',               # sulphates
    'alcool',                 # alcohol
    'score_qualidade_vinho'              # quality
]
```

```
Dica 2. Separe os dados (x e y) de tal forma que a última coluna (índice -1), chamada score_qualidade_vinho, seja a variável target (y)
```

3 Sistemas de Recomendação

Implementar o exemplo de Sistemas de Recomendação usando a base de dados Base_livos.csv e a arquitetura vista na aula **FRA - Aula 22 - 4.3 Resolução do Exercício de Sistemas de Recomendação**. Além disso, fazer uma breve explicação dos seguintes resultados:

- Gráficos de avaliação do modelo (loss);
- Exemplo de recomendação de livro para determinado Usuário.

Informações:

- **Base de dados:** Base_livros.csv
- **Descrição**: Esse conjunto de dados contém informações sobre avaliações de livros (Notas), nomes de livros (Titulo), ISBN e identificação do usuário (ID_usuario)
- **Importação:** Base de dados disponível no Moodle (UFPR Virtual), chamada Base_livros (formato .csv).

4 Deepdream

Implementar o exemplo de implementação mínima de Deepdream usando uma imagem de um felino - retirada do site Wikipedia - e a arquitetura Deepdream vista na aula **FRA - Aula 23 - Prática Deepdream**. Além disso, fazer uma breve explicação dos seguintes resultados:

- Imagem onírica obtida por *Main Loop*;
- Imagem onírica obtida ao levar o modelo até uma oitava;
- Diferenças entre imagens oníricas obtidas com *Main Loop* e levando o modelo até a oitava.

Informações:

- **Base de dados:** https://commons.wikimedia.org/wiki/File:Felis_catus-cat_on_snow.jpg
- **Importação da imagem**: Copiar código abaixo.

```
url = "https://commons.wikimedia.org/wiki/Special:FilePath/Felis_catus-cat_on_snow.jpg"
```

```
Dica: Para exibir a imagem utilizando display (display.html) use o link https://commons.wikimedia.org/wiki/File:Felis_catus-cat_on_snow.jpg
```

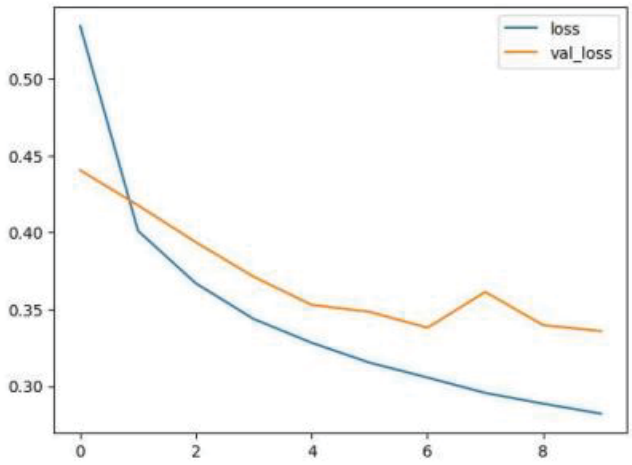
B – RESOLUÇÃO

Questão 1

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
from tensorflow.keras.datasets import fashion_mnist
data = tf.keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
print("x_train.shape: ", x_train.shape)
print("y_train.shape: ", y_train.shape)
print("x_test.shape: ", y_test.shape)
print("y_test.shape: ", y_test.shape)
display(x_train)
x_train, x_test = x_train/255.0, x_test/255.0
i = tf.keras.layers.Input(shape=(28, 28))
x = tf.keras.layers.Flatten()(i)
x = tf.keras.layers.Dense(128, activation="relu")(x)
```

```
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(10, activation="softmax")(x)

model = tf.keras.models.Model(i, x)

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

r = model.fit(x_train,
              y_train,
              validation_data=(x_test, y_test),
              epochs=10)

Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.7636
- loss: 0.6768 - val_accuracy: 0.8408 - val_loss: 0.4404
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8939
- loss: 0.2867 - val_accuracy: 0.8823 - val_loss: 0.3357

# Plotar a função de perda
plt.plot(r.history["loss"], label="loss")
plt.plot(r.history["val_loss"], label="val_loss")
plt.legend()
```

FIGURA 19 – FUNÇÃO DE PERDA DE CLASSIFICAÇÃO COM RNA

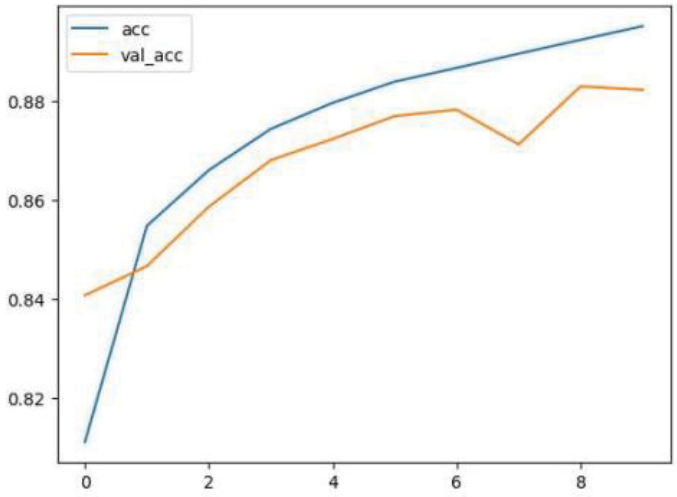


FONTE: De autoria própria (2024)

O gráfico mostra que as perdas de treinamento (loss) e validação (val_loss) diminuem com as épocas, indicando aprendizado do modelo.
Primeiras épocas: Queda acentuada em ambas as perdas, mostrando rápido aprendizado inicial. Últimas épocas: loss continua caindo, mas val_loss estabiliza com pequenas flutuações.

```
# Plotar a acurácia
plt.plot(r.history["accuracy"], label="acc")
plt.plot(r.history["val_accuracy"], label="val_acc")
plt.legend()
```

FIGURA 20 – ACURÁCIA DE CLASSIFICAÇÃO COM RNA



FONTE: De autoria própria (2024)

```
# Avaliar o modelo com a base de teste
print( model.evaluate(x_test, y_test) )
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8826 -
loss: 0.3329
[0.33573681116104126, 0.8823000192642212]
```

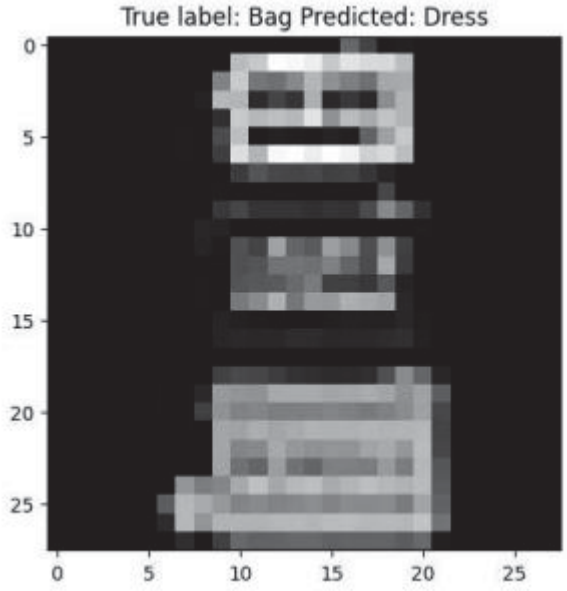
Função de Acurácia: 0.8826

O gráfico mostra que, ao longo das épocas, a acurácia no conjunto de treinamento (acc) cresce de forma consistente, enquanto a acurácia no conjunto de validação (val_acc) também aumenta, mas apresenta pequenas flutuações.
Primeiras épocas: Melhorias rápidas em ambas as acurácias, indicando aprendizado inicial eficiente.
Últimas épocas: acc continua crescendo, mas val_acc se estabiliza e começa a divergir.

```
# Nomes das categorias (no mesmo formato que y_test e y_pred)
category_names = [
    "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
    "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"
]

# Exibir imagem do conjunto de teste com rótulo verdadeiro e predito como
categoria
plt.imshow(x_test[i].reshape(28, 28), cmap="gray")
plt.title("True label: %s Predicted: %s" % (category_names[y_test[i]],
category_names[y_pred[i]]))
plt.show()
```

Mostrar algumas classificações erradas: A predição aponta que esta imagem é um Dress mas na verdade é um Bag.

FIGURA 21 – PREDIÇÃO ERRADA DE VESTIDO



FONTE: De autoria própria (2024)

Questão 2

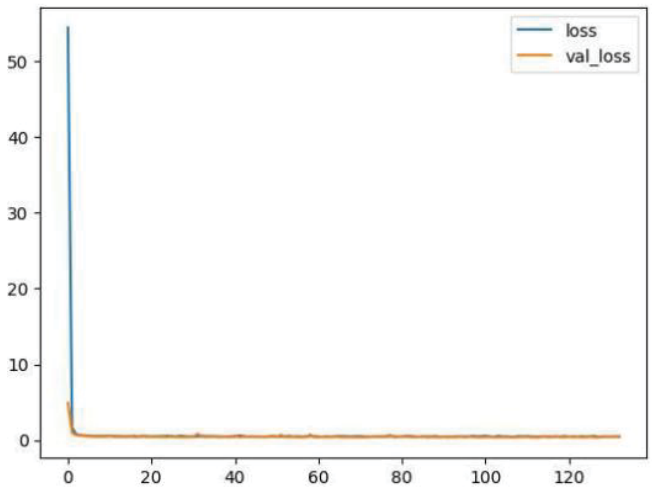
```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from tensorflow.python.keras import backend
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt

url    =    "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality-red.csv"
import pandas as pd

data = pd.read_csv(url, delimiter=';')

data.columns = [
'acidez_fixa',    # fixed acidity
'acidez_volatil', # volatile acidity
'acido_citrico',  # citric acid 'acucar_residual',    # residual sugar
'acucar_residual',      # residual sugar 'cloretos',  # chlorides
'cloretos', # chlorides
'dioxido_de_enxofre_livre', # free sulfur dioxide
'dioxido_de_enxofre_total', # total sulfur dioxide
'densidade',      # density
'pH', # pH
'sulfatos', # sulphates
'alcool',   # alcohol
'score_qualidade_vinho'       # quality
]
# Selecionar as colunas de 1 a 11 (índices 0 a 10)
X = data.iloc[:, 0:11].astype(float)

# Selecionar a coluna 12 (índice 11) como Y
Y = data.iloc[:, 11].astype(float)
x_train, x_test, y_train, y_test = train_test_split(X, Y,
                                        test_size=0.25)
i = tf.keras.layers.Input(shape=(11,))
x = tf.keras.layers.Dense(50, activation="relu")(i)
x = tf.keras.layers.Dense(1)(x)

model = tf.keras.models.Model(i, x)
# Criação de funções para as métricas R2 e RMSE serem inseridas no modelo
def rmse(y_true, y_pred):
  return backend.sqrt(backend.mean( backend.square(y_pred - y_true), axis=-
1) )
```

```
def r2(y_true, y_pred):
  media = backend.mean(y_true)
  num   = backend.sum (backend.square(y_true - y_pred))
  den   = backend.sum (backend.square(y_true - media))
  return (1.0 - num/den)

# Compilação
optimizer=tf.keras.optimizers.Adam(learning_rate=0.05)
# optimizer=tf.keras.optimizers.SGD(learning_rate=0.2, momentum=0.5)
# optimizer=tf.keras.optimizers.RMSprop(0.01)

model.compile(optimizer=optimizer,
              loss="mse",
              metrics=[rmse, r2])

model.summary()
# Early stop para epochs
early_stop = tf.keras.callbacks.EarlyStopping(
                            monitor='val_loss',
                            patience=20,
                            restore_best_weights=True)
r = model.fit(x_train, y_train,
              epochs=1500,
              validation_data=(x_test, y_test),
              callbacks=[early_stop])

Epoch 1/1500
38/38 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 113.8945 - r2:
-182.7078 - rmse: 7.6008 - val_loss: 4.9176 - val_r2: -7.5493 - val_rmse:
1.9644
Epoch 132/1500
38/38 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.4787 - r2:
0.2548 - rmse: 0.5367 - val_loss: 0.4720 - val_r2: 0.1926 - val_rmse: 0.5414
Epoch 133/1500
38/38 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5064 - r2:
0.2255 - rmse: 0.5559 - val_loss: 0.3949 - val_r2: 0.3209 - val_rmse: 0.5138

plt.plot( r.history["loss"], label="loss" )
plt.plot( r.history["val_loss"], label="val_loss" )
```

```
plt.legend()
```

FIGURA 22 – APRENDIZADO DO MODELO COM REGRESSÃO

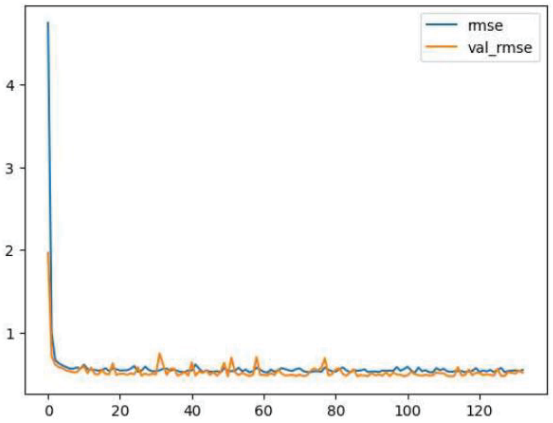


FONTE: De autoria própria (2024)

O modelo demonstrou um aprendizado eficiente e consistente, alcançando boa generalização e evitando problemas de sobreajuste ou subajuste. Esses resultados indicam que a arquitetura e os parâmetros utilizados foram adequados para a tarefa de regressão.

```
plt.plot( r.history["rmse"], label="rmse" )
plt.plot( r.history["val_rmse"], label="val_rmse" )
plt.legend()
```

FIGURA 23 – RESULTADOS DE REGRESSÃO COM RMSE

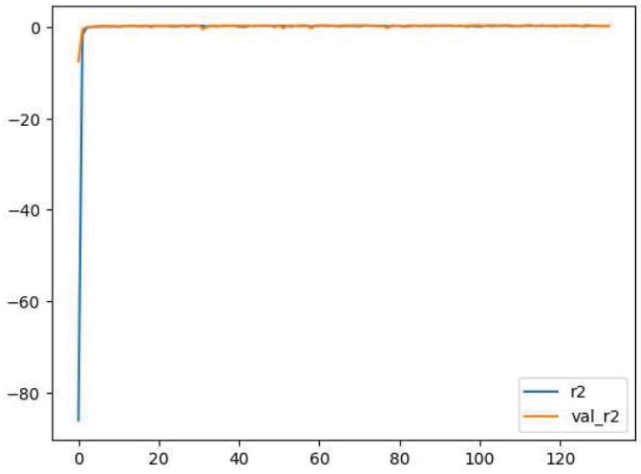


FONTE: De autoria própria (2024)

O gráfico de RMSE indica um bom aprendizado do modelo, com rápida redução do erro inicial e estabilização em valores baixos após cerca de 10-15 épocas. As curvas de treino e validação são próximas, mostrando boa generalização. Não há sinais de overfiţng, e o desempenho foi consistente nos dados de validação. O modelo é eficaz para a tarefa proposta.

```
plt.plot( r.history["r2"], label="r2" )
plt.plot( r.history["val_r2"], label="val_r2" )
plt.legend()
```

FIGURA 24 – RESULTADOS DE REGREÇÃO COM R2



FONTE: De autoria própria (2024)

```
# Predição
y_pred = model.predict(x_test).flatten()
# Cálculo das métricas de acurácia: mse, r2 e rmse
mse  = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
r2   = r2_score(y_test, y_pred)
# Resultados das métricas de acurácia
print("mse     = ", mse)
print("rmse    = ", rmse)
print("r2      = ", r2)
mse     =  0.35148531462085997
rmse    =  0.59286196928194
r2      =  0.43409223213514747
```

O gráfico do coeficiente R2 mostra uma rápida melhoria inicial, alcançando valores estáveis após poucas épocas. As curvas de treino e validação são muito próximas, indicando boa generalização do modelo. O R2 próximo de 0 ou levemente negativo sugere que o modelo ainda não explica completamente a variação dos dados. Ajustes adicionais podem melhorar a qualidade do ajuste.
Conclusão: Os resultados indicam um MSE de 0.35 e um RMSE de 0.59, representando um erro moderado na previsão. O R2 de 0.43 mostra que o modelo explica cerca de 43% da variância dos dados, sugerindo espaço para melhorias no desempenho preditivo.

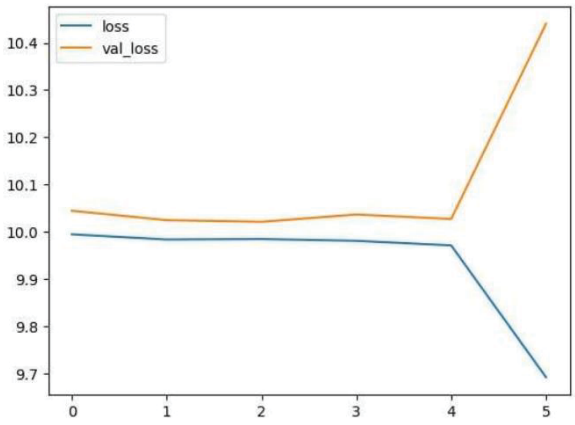
Questão 3

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Embedding, Flatten,
Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.utils import shuffle

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
df=pd.read_csv('/content/drive/MyDrive/FRAMEWORKS/Base_livos.csv',
low_memory=False)
df.ID_usuario = pd.Categorical(df.ID_usuario)
df['new_ID_usuario'] = df.ID_usuario.cat.codes

df.ISBN = pd.Categorical(df.ISBN)
df['new_ISBN'] = df.ISBN.cat.codes
# Dimensões dos embeddings
N = len(set(df.ID_usuario))  # Número de usuários únicos
M = len(set(df.new_ISBN))    # Número de livros únicos

# Dimensão do embedding (hiperparâmetro)
K = 10

print(f"Número de usuários (N): {N}")
print(f"Número de itens (M): {M}")
print(f"Dimensão do embedding (K): {K}")
Número de usuários (N): 11987
```

```
Número de itens (M): 128894
Dimensão do embedding (K): 10
from tensorflow.keras.layers import Dropout

# Usuário
u = Input(shape=(1,))
u_emb = Embedding(N, K)(u)  # saída : num_samples, 1, K
u_emb = Flatten()(u_emb)   # saída : num_samples, K

# Filme
m = Input(shape=(1,))
m_emb = Embedding(M, K)(m)  # saída : num_samples, 1, K
m_emb = Flatten()(m_emb)    # saída : num_samples, K

# Concatenando as camadas de usuário e livro
x = Concatenate()([u_emb, m_emb])

# Camada densa com ReLU
x = Dense(1024, activation="relu")(x)

# Aplicando Dropout
x = Dropout(0.1)(x)  # Desligando 10% dos neurônios

# Camada de saída
x = Dense(1)(x)

# Modelo
model = Model(inputs=[u, m], outputs=x)
model.compile(
    loss="mse",
    optimizer=SGD(learning_rate=0.05, momentum=0.9)
)
# Embaralhar os dados de forma sincronizada
usuario_ids,  ISBN_ids,  notas  =  shuffle(df.new_ID_usuario,  df.new_ISBN,
df.Notas, random_state=42)

# Divisão 80% - 20%
Ntrain = int(0.8 * len(notas))  # 80% para treinamento

train_usuario = usuario_ids[:Ntrain]
train_ISBN = ISBN_ids[:Ntrain]
```

```
train_notas = notas[:Ntrain]

test_usuario = usuario_ids[Ntrain:]
test_ISBN = ISBN_ids[Ntrain:]
test_notas = notas[Ntrain:]

# Centralizar as notas
avg_notas = train_notas.mean()
train_notas = train_notas - avg_notas
test_notas = test_notas - avg_notas  # Subtrair a média do treinamento das
notas de teste
epochs = 25

early_stopping     =     EarlyStopping(monitor='val_loss',     patience=3,
restore_best_weights=True)

r = model.fit(
    x=[train_usuario, train_ISBN],
    y=train_notas,
    epochs=epochs,
    batch_size=512,
    verbose=2,
    validation_data=([test_usuario, test_ISBN], test_notas),
    callbacks=[early_stopping]  # Usando EarlyStopping
)

Epoch 1/25
202/202 - 5s - 25ms/step - loss: 9.9943 - val_loss: 10.0443
Epoch 6/25
202/202 - 0s - 2ms/step - loss: 9.6917 - val_loss: 10.4402

plt.plot(r.history["loss"], label="loss")
plt.plot(r.history["val_loss"], label="val_loss")
plt.legend()
plt.show()
```

FIGURA 25 – FUNÇÃO DE PERDA DO SISTEMA DE RECOMENDAÇÃO



FONTE: De autoria própria (2024)

Análise e Observações: O gráfico mostra que durante as primeiras épocas do treinamento do modelo, o aprendizado ainda é baixo, por isso os valores de loss e val_loss praticamente não sofrem alteração, porém a partir da quinta época, o valor de loss diminuiu de forma agressiva, mas o val_loss chega a aumentar.

Devido a função early stopping configurada para monitorar o val_loss (função de perda dos dados de validação), o treinamento foi interrompido, pois o val_loss não estava melhorando. Essa parada antecipada mostra que a continuidade do treinamento resultaria em overfiţng, visto que os dados de validação não estavam tendo evolução no aprendizado e por isso o modelo não está generalizando bem os novos dados. Justificativa dos resultados e sugestão de Intervenção: Alguns dos parâmetros utilizados podem ser ajustados para buscar um modelo mais eficiente, além disso, outras questões podem ser verificadas. Algumas ações que podem tornar o modelo melhor seriam a redução da taxa de aprendizado (foi usado 0,05) visando evitar oscilações e a redução do número de neurônios (foi usado 1024) na camada densa, visto que a redução do número de pesos pode auxiliar na generalização. O número de épocas pode ser aumentado para um número entre 50 e 100, pois desta maneira o modelo poderá ver os dados mais vezes e consequentemente pode ajustar os pesos de uma forma mais gradual para chegar de uma convergência com melhor precisão. A alteração do batch_size também poderá melhorar o modelo, pois o número usado (512) acabou não capturando bem a variação dos dados usados. Outros testes que podem ser feitos são a alteração do algoritmo otimizador para verificação se o resultado tem um ganho na generalização e testar outros valores mais altos no dropout na camada densa (neste caso, também pode ser necessário ajustar o número de épocas para uma avaliação mais eficiente).

```
# Gerar o array com o usuário único
# repete a quantidade de livros
input_usuario = np.repeat(a=20584, repeats=M)
livro = np.array(list(set(ISBN_ids)))
```

```
preds = model.predict([input_usuario, livro])

# descentraliza as predições
rat = preds.flatten() + avg_notas

# índice da maior nota
idx = np.argmax(rat)

print("Recomendação: Livro - ", livro[idx], " / ", rat[idx], "*")

4028/4028 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step
Recomendação: Livro - 77837 / 5.0071154 *
```

No exemplo acima, é feita uma sugestão de livro para o usuário 20584. Além de apontar o id do livro sugerido, é feita a predição de uma nota aproximada que o usuário deve atribuir ao livro, considerando todos as notas dadas ao livro por outros usuários e as notas que o próprio usuário tende a atribuir aos livros que ele já fez a leitura.

Questão 4

```
import tensorflow as tf
import numpy as np

import matplotlib as mpl

import IPython.display as display
import PIL.Image
url   =   'https://commons.wikimedia.org/wiki/Special:FilePath/Felis_catus-
cat_on_snow.jpg'
# Download da imagem e gravação em array Numpy
def download(url, max_dim=None):
  name = url.split('/')[-1]
  image_path = tf.keras.utils.get_file(name, origin=url)
  img = PIL.Image.open(image_path)
  if max_dim:
    img.thumbnail((max_dim, max_dim))
  return np.array(img)

# Normalização da imagem
def deprocess(img):
  img = 255*(img + 1.0)/2.0
```

```
  return tf.cast(img, tf.uint8)

# Mostra a imagem
def show(img):
  display.display(PIL.Image.fromarray(np.array(img)))



# Redução do tamanho da imagem para facilitar o trabalho da RNN
original_img = download(url, max_dim=500)
show(original_img)
display.display(display.HTML('Image                cc-by:                <a
"https://commons.wikimedia.org/wiki/File:Felis_catus-
cat_on_snow.jpg">Von.grzanka</a>'))
base_model      =      tf.keras.applications.InceptionV3(include_top=False,
weights='imagenet')
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Model

# Carregando o modelo base pré-treinado
base_model = InceptionV3(weights='imagenet', include_top=False)

# Maximizando as ativações das camadas
names = ['mixed3', 'mixed5']
layers = [base_model.get_layer(name).output for name in names]

# Criação do modelo
dream_model = Model(inputs=base_model.input, outputs=layers)

# Resumo do modelo criado
dream_model.summary()
def calc_loss(img, model):
  # Passe a imagem pelo modelo para recuperar as ativações.
  # Converte a imagem em um batch de tamanho 1.
  img_batch = tf.expand_dims(img, axis=0)
  layer_activations = model(img_batch)
  if len(layer_activations) == 1:
    layer_activations = [layer_activations]

  losses = []
  for act in layer_activations:
    loss = tf.math.reduce_mean(act)
```

```
    losses.append(loss)

  return  tf.reduce_sum(losses)
class DeepDream(tf.Module):
  def __init__(self, model):
    self.model = model

  @tf.function(
      input_signature=(
        tf.TensorSpec(shape=[None,None,3], dtype=tf.float32),
        tf.TensorSpec(shape=[], dtype=tf.int32),
        tf.TensorSpec(shape=[], dtype=tf.float32),)
  )
  def __call__(self, img, steps, step_size):
      print("Tracing")
      loss = tf.constant(0.0)

      for n in tf.range(steps):
        with tf.GradientTape() as tape:
          # Gradientes relativos a img
          tape.watch(img)
          loss = calc_loss(img, self.model)

        # Calculo do gradiente da perda em relação aos pixels da imagem de
entrada.
        gradients = tape.gradient(loss, img)

        # Normalizacao dos gradintes
        gradients /= tf.math.reduce_std(gradients) + 1e-8

        # Na subida gradiente, a "perda" é maximizada.
        # Você pode atualizar a imagem adicionando diretamente os gradientes
(porque eles têm o mesmo formato!)
        img = img + gradients*step_size
        img = tf.clip_by_value(img, -1, 1)

      return loss, img

deepdream = DeepDream(dream_model)

def run_deep_dream_simple(img, steps=100, step_size=0.01):
```

```
  img = tf.keras.applications.inception_v3.preprocess_input(img)
  img = tf.convert_to_tensor(img)
  step_size = tf.convert_to_tensor(step_size)
  steps_remaining = steps
  step = 0
  while steps_remaining:
    if steps_remaining>100:
      run_steps = tf.constant(100)
    else:
      run_steps = tf.constant(steps_remaining)
    steps_remaining -= run_steps
    step += run_steps

    loss, img = deepdream(img, run_steps, tf.constant(step_size))

    display.clear_output(wait=True)
    show(deprocess(img))
    print ("Step {}, loss {}".format(step, loss))


  result = deprocess(img)
  display.clear_output(wait=True)
  show(result)

  return result

dream_img = run_deep_dream_simple(img=original_img,
                                  steps=100, step_size=0.01)
```

FIGURA 25 – IMAGEM ONÍRICA

FONTE: De autoria própria (2024)

A imagem onírica obtida no Main Loop resulta do processo de amplificação de padrões que o modelo pré-treinado reconhece nos dados de entrada. Esse efeito é gerado utilizando a técnica conhecida como Deep Dream.

O Deep Dream é uma técnica baseada em redes neurais convolucionais (como IncepΘonV3) que maximiza as ativações de camadas específicas. Isso amplifica os padrões que a rede reconhece na imagem, criando formas "surreais" e texturas complexas.

A imagem de entrada é processada iterativamente. O modelo tenta aumentar as ativações dos filtros das camadas especificadas no dream_model. Em cada passo, o modelo ajusta os valores dos pixels da imagem de entrada para amplificar os padrões.

Elementos visuais, como curvas, olhos ou formas geométricas, são amplificados, criando um aspecto "onírico". Padrões vibrantes: As cores e os detalhes tornam-se mais pronunciados à medida que os filtros são maximizados. Integração dos padrões: As texturas não são adicionadas aleatoriamente; elas emergem de áreas da imagem onde os filtros da rede detectaram elementos significativos.

O modelo identificou características específicas do gato (como olhos, pelos e contornos) e amplificou esses padrões. Camadas diferentes da rede contribuem para texturas e formas mais complexas à medida que os filtros interpretam a imagem.

```
import time
```

```
start = time.time()

OCTAVE_SCALE = 1.30

img = tf.constant(np.array(original_img))
base_shape = tf.shape(img)[:-1]
float_base_shape = tf.cast(base_shape, tf.float32)

for n in range(-2, 3):
  new_shape = tf.cast(float_base_shape*(OCTAVE_SCALE**n), tf.int32)

  img = tf.image.resize(img, new_shape).numpy()

  img = run_deep_dream_simple(img=img, steps=50, step_size=0.01)

display.clear_output(wait=True)
img = tf.image.resize(img, base_shape)
img = tf.image.convert_image_dtype(img/255.0, dtype=tf.uint8)
show(img)

end = time.time()
end-start
```

FIGURA 26 – IMAGEM ONÍRICA COM USO DE OITAVAS

FONTE: De autoria própria (2024)

A imagem onírica obtida com o uso de oitavas apresenta padrões mais refinados e complexos, pois combina informações em múltiplas escalas de resolução. Detalhes maiores, como o corpo do gato, são

destacados em resoluções menores, enquanto texturas finas, como olhos e contornos, são refinadas em resoluções maiores. Esse processo multiescala reduz o ruído e cria uma integração mais harmoniosa dos padrões amplificados. No código, a imagem é redimensionada iterativamente com o fator OCTAVE_SCALE, passando pelo *Deep Dream* em cada nível antes de ser retornada ao tamanho original. O resultado é uma obra visual rica, detalhada e surreal. As diferenças entre as imagens oníricas obtidas com o Main Loop e levando o modelo até uma oitava são as seguintes:

Detalhamento Multiescala: No Main Loop, os padrões são amplificados em uma única resolução, resultando em detalhes uniformes e, às vezes, caóticos. Com oitavas, o modelo processa a imagem em diferentes escalas, refinando padrões em níveis globais e locais.

Redução de Ruído: A imagem do Main Loop tende a ser mais ruidosa devido à amplificação direta. Usando oitavas, o redimensionamento multiescala suaviza transições e integra melhor os padrões.

Complexidade Visual: No Main Loop, os detalhes podem parecer menos harmoniosos. Já com oitavas, os padrões são mais ricos e detalhados, devido ao refinamento em várias resoluções.

Resolução e Granularidade: A imagem do Main Loop é processada diretamente na resolução original, enquanto com oitavas, a granularidade dos padrões melhora em várias escalas de tamanho.

Harmonia Visual: A técnica com oitavas, cria uma composição mais equilibrada, combinando detalhes finos e amplos, enquanto o Main Loop é limitado à resolução única do processamento.

APÊNDICE N - VISUALIZAÇÃO DE DADOS E STORYTELLING

A – ENUNCIADO

Escolha um conjunto de dados brutos (ou uma visualização de dados que você acredite que possa ser melhorada) e faça uma visualização desses dados (de acordo com os dados escolhidos e com a ferramenta de sua escolha)

Desenvolva uma narrativa/storytelling para essa visualização de dados considerando os conceitos e informações que foram discutidas nesta disciplina. Não esqueça de deixar claro para seu possível público alvo qual **o objetivo dessa visualização de dados, o que esses dados significam, quais possíveis ações podem ser feitas com base neles.**

Entregue em um PDF:

- O **conjunto de dados brutos** (**ou uma visualização de dados** que você acredite que possa ser **melhorada**);

- Explicação do **contexto e o publico-alvo** da visualização de dados e do storytelling que será desenvolvido;

- A **visualização desses dados** (de acordo com os dados escolhidos e com a ferramenta de sua escolha) **explicando a escolha do tipo de visualização e da ferramenta usada**; (**50 pontos**)

B – RESOLUÇÃO

Conjunto de Dados Brutos
Para este trabalho, foi escolhido um conjunto de dados sobre o crescimento populacional urbano em diferentes cidades ao longo do tempo. Esses dados permitem analisar padrões de urbanização, impactos ambientais e desafios para o planejamento urbano.

Cidade: São Paulo
Ano: 2020
População: 12.325.232
Área (km²): 1.521

Cidade: Campinas
Ano: 2020

População: 1.213.792
Área (km²): 795

Cidade: Ribeirão Preto
Ano: 2020
População: 720.116
Área (km²): 650

Contexto e Público-Alvo
Contexto:
A urbanização acelerada é um fenômeno global, trazendo desafios relacionados à infraestrutura, mobilidade, poluição e qualidade de vida. A compreensão desses dados é essencial para gestores públicos e planejadores urbanos.
Público-Alvo:
Este trabalho se destina a gestores públicos, urbanistas e pesquisadores interessados em compreender a evolução da população urbana e seus impactos.
Visualização de Dados

Ferramenta Utilizada:
Foi utilizado Python como ferramenta devido à sua flexibilidade e às bibliotecas
poderosas para análise e visualização de dados.

Matplotlib e Seaborn foram usadas para gráficos estáticos de linhas e barras, ideais para mostrar a evolução populacional e comparações entre homens e mulheres.
Plotly foi escolhida para gráficos interativos, como a comparação de homens e mulheres em 2023, permitindo que o público explore os dados dinamicamente.
Geopandas e Folium foram aplicadas para criar um heatmap interativo, visualizando a distribuição da população nos estados de São Paulo, destacando áreas com maior densidade populacional.

Tipos de Visualização Escolhidos:
Gráficos de Linhas (Evolução Populacional e Razão Sexo): A escolha dos gráficos de linhas é adequada para mostrar a mudança ao longo do tempo, como a evolução da população de Indaiatuba e a razão sexo. Gráficos de linhas são eficazes para destacar tendências e padrões temporais.

Gráfico de Barras (Comparação entre Homens e Mulheres em 2023): Os gráficos de barras são ótimos para comparações entre categorias distintas. Ao comparar os valores absolutos de homens e mulheres em 2023, esse tipo de gráfico facilita a visualização das diferenças entre os dois grupos.

Gráfico de Dispersão (População e Densidade Demográfica): Relacionar a população e a densidade demográfica de maneira visual permite ao público entender como esses dois fatores interagem. Um gráfico de dispersão ajuda a identificar padrões ou correlações entre essas variáveis.

Heatmap (População nos Estados de São Paulo): O heatmap interativo gerado pelo Folium proporciona uma visualização de dados geoespaciais de forma intuitiva, permitindo que o público explore a distribuição populacional nos estados de São Paulo. O uso de cores para representar a intensidade de valores (população) facilita a identificação das regiões com maior ou menor densidade populacional.

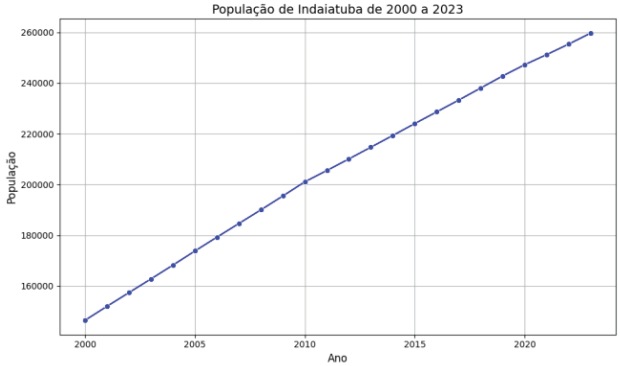
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import plotly.graph_objects as go
import geopandas as gpd
import folium
from folium.plugins import HeatMap

# Definir o caminho dos arquivos
caminho_arquivo = r"C:\Users\eduse\OneDrive\Documentos\UFPR - Inteligencia Artificial\IAA007 - Visualização de Dados e Storytelling\Graficos\estimativa_pop_indicadores_esp.csv"
shapefile_path = r"C:\Users\eduse\OneDrive\Documentos\UFPR - Inteligencia Artificial\IAA007 - Visualização de Dados e Storytelling\Graficos\DensidadeDemografica2023Polygon.shp"

# Ler os arquivos
df = pd.read_csv(caminho_arquivo, encoding='latin1', delimiter=';')
gdf_sp = gpd.read_file(shapefile_path)

# Mesclar a base de dados (df) com o shapefile (gdf_sp) usando a coluna 'nome_mun'
gdf_sp = gdf_sp.merge(df[['nome_mun', 'populacao']], left_on='Nome', right_on='nome_mun', how='left')

# Criar o mapa base
m = folium.Map(location=[-23.5505, -46.6333], zoom_start=7)  # Centralizado no estado de SP

# Adicionar os dados ao heatmap (usar a coluna 'populacao' como intensidade)
heat_data = [
```

```
    [point['geometry'].centroid.y,            point['geometry'].centroid.x,
point['populacao']]  # Usando a População
    for idx, point in gdf_sp.iterrows() if not pd.isna(point['populacao'])
]

# Adicionar o Heatmap ao mapa
HeatMap(heat_data).add_to(m)

# Salvar o mapa interativo em um arquivo HTML
m.save('heatmap_sp_populacao.html')

# Exibir a mensagem de sucesso
print("Mapa interativo com heatmap de população foi salvo como
'heatmap_sp_populacao.html'")

# Filtrar os dados para o município de Indaiatuba
df_indaiatuba = df[df['nome_mun'] == 'Indaiatuba']

# Filtrar dados para o estado de São Paulo
df_sp = df[df['cod_ibge'].astype(str).str[:2] == '35']  # '35' é o código de
São Paulo

# Calcular o crescimento populacional para cada município (usando .loc para
evitar o warning)
df_sp.loc[:,                        'crescimento_pop']                      =
df_sp.groupby('nome_mun')['populacao'].diff()

# Agora, ajustar a chave de merge para o nome correto da coluna
gdf_sp      =      gdf_sp.merge(df_sp[['nome_mun',      'crescimento_pop']],
left_on='Nome', right_on='nome_mun', how='left')

# Criar um mapa base do estado de São Paulo
m = folium.Map(location=[-23.5505, -46.6333], zoom_start=7)  # Localização
centralizada no estado de SP

# Adicionar o heatmap ao mapa
heat_data  =  [[point['geometry'].centroid.y,  point['geometry'].centroid.x,
point['crescimento_pop']]
             for    idx,    point    in    gdf_sp.iterrows()    if    not
pd.isna(point['crescimento_pop'])]
```

```
HeatMap(heat_data).add_to(m)

# Exibir o mapa
m.save('heatmap_sp.html')

# Gráfico de Linha: População de Indaiatuba de 2000 a 2023
plt.figure(figsize=(10, 6))
sns.lineplot(x='ano', y='populacao', data=df_indaiatuba, marker='o',
color='b')
plt.title('Evolução da População de Indaiatuba (2000-2023)', fontsize=14)
plt.xlabel('Ano', fontsize=12)
plt.ylabel('População', fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()

# Gráfico de Linha: Razão Sexo (Homens/Mulheres) ao Longo dos Anos
plt.figure(figsize=(10, 6))
sns.lineplot(x='ano', y='razao_sexo', data=df_indaiatuba, marker='o',
color='purple')
plt.title('Evolução da Razão Sexo em Indaiatuba (2000-2023)', fontsize=14)
plt.xlabel('Ano', fontsize=12)
plt.ylabel('Razão Sexo (Homens/Mulheres)', fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()

# Gráfico de Barras: Comparação de Homens e Mulheres em 2023
df_2023 = df_indaiatuba[df_indaiatuba['ano'] == 2023]
plt.figure(figsize=(10, 6))
bar_width = 0.35
x = np.arange(1)

# Plot para Homens
plt.bar(x - bar_width / 2, df_2023['homens'], width=bar_width,
label='Homens', color='skyblue')

# Plot para Mulheres
plt.bar(x + bar_width / 2, df_2023['mulheres'], width=bar_width,
label='Mulheres', color='salmon')
```

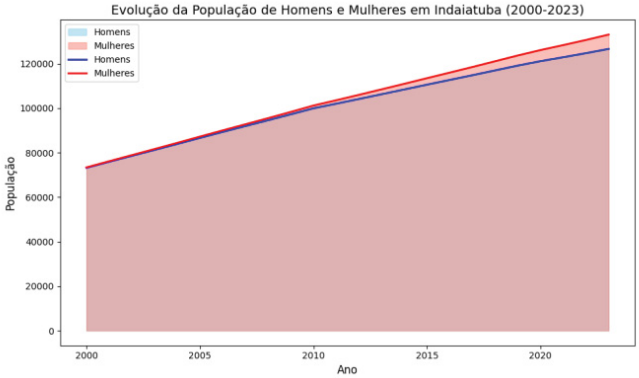
```
plt.title('Comparação de Homens e Mulheres em Indaiatuba (2023)', fontsize=14)
plt.xticks(x, ['2023'])
plt.xlabel('Ano', fontsize=12)
plt.ylabel('População', fontsize=12)
plt.legend()
plt.tight_layout()
plt.show()

# Gráfico de Dispersão: População vs Densidade Demográfica
plt.figure(figsize=(10, 6))
sns.scatterplot(x='populacao', y='dens_demog', data=df_indaiatuba, hue='ano', palette='viridis', size='ano', sizes=(50, 200), legend=False)
plt.title('Relação entre População e Densidade Demográfica em Indaiatuba (2000-2023)', fontsize=14)
plt.xlabel('População', fontsize=12)
plt.ylabel('Densidade Demográfica (hab/km²)', fontsize=12)
plt.tight_layout()
plt.show()

# -------------------- Gráfico Heatmap de Crescimento Populacional por Estado --------------------

# Calcular o crescimento populacional entre anos para todos os estados (usando .loc para evitar o warning)
df.loc[:, 'crescimento_pop'] = df.groupby('nome_mun')['populacao'].diff()

# Pivotar os dados para um formato adequado ao heatmap
df_pivot = df.pivot_table(index='nome_mun', columns='ano', values='crescimento_pop')

# Plotando o Heatmap
plt.figure(figsize=(14, 8))
sns.heatmap(df_pivot, cmap='YlGnBu', annot=True, fmt=".0f", linewidths=0.5)
plt.title('Crescimento Populacional por Estado (2000-2023)', fontsize=16)
plt.xlabel('Ano', fontsize=12)
plt.ylabel('Município', fontsize=12)
plt.tight_layout()
plt.show()
```

FIGURA 27 – POPULAÇÃO DE INDAIATUBA DE 2000 A 2023



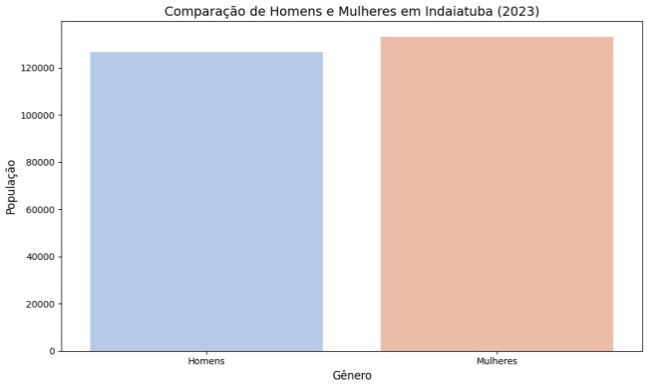
FONTE: De autoria própria (2025)

FIGURA 28 – EVOLUÇÃO DA POPULAÇÃO DE HOMENS E MULHERES EM INDAIATUBA DE 2000 A 2023



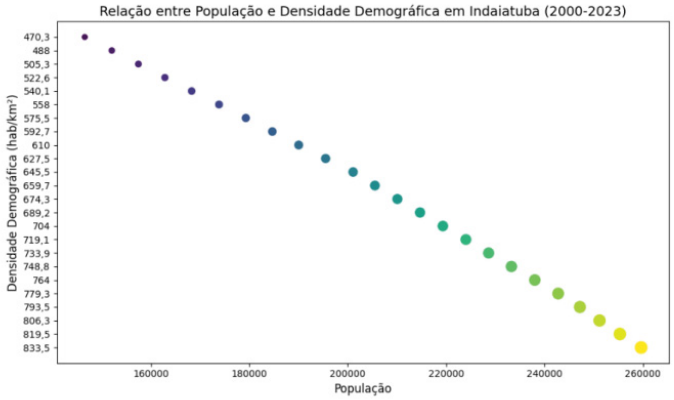
FONTE: De autoria própria (2025)

FIGURA 29 – COMPARAÇÃO DE HOMENS E MULHERES EM INDAIATUBA 2023



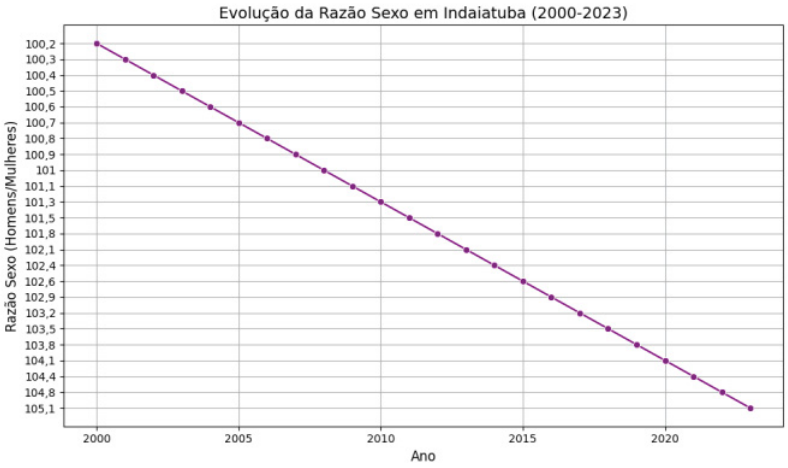
FONTE: De autoria própria (2025)

FIGURA 30 – RELAÇÃO ENTRE POPULAÇÃO E DENSIDADE DEMOGRÁFICA EM INDAIATUBA 2000 A 2023



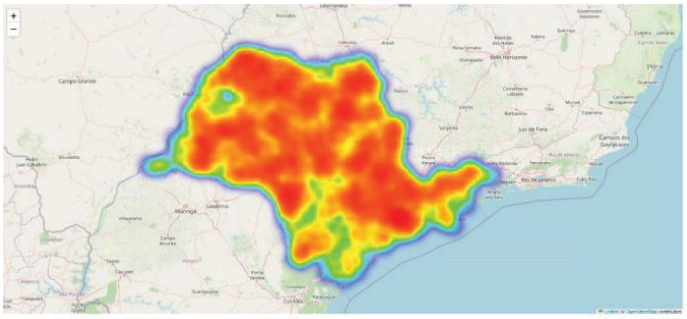
FONTE: De autoria própria (2025)

FIGURA 31 – EVOLUÇÃO DA RAZÃO SEXO EM INDAIATUBA



FONTE: De autoria própria (2025)

FIGURA 32 – MAPA DE CALOR DE SÃO PAULO



FONTE: De autoria própria (2025)

Esta visualização de dados tem como objetivo apresentar padrões de crescimento populacional urbano, permitindo uma melhor compreensão dos impactos desse crescimento e auxiliando na tomada de decisões.

Os dados mostram tendências de crescimento da população de Indaiatuba ao longo dos anos, evidenciando:

O crescimento contínuo da população na cidade.

A evolução das diferenças entre os gêneros.

Como a densidade populacional tem mudado ao longo dos anos.

A história contada pelos dados começa em 2000, quando a população de Indaiatuba começa a crescer de forma acelerada. A cada ano, observa-se um aumento constante, com pequenas oscilações que podem estar relacionadas a fatores econômicos, migração e desenvolvimento urbano. Nos gráficos comparativos de gênero, é possível notar como a proporção entre homens e mulheres se manteve estável ao longo dos anos, mas com pequenas variações que podem indicar mudanças no perfil demográfico. Por fim, o mapa de densidade demográfica ilustra como diferentes regiões da cidade têm se tornado mais populosas ao longo do tempo, influenciando a necessidade de investimentos em infraestrutura e serviços urbanos.

Possíveis Ações Baseadas nos Dados
Com base na análise dos dados, algumas ações podem ser consideradas:

- Desenvolvimento de planos estratégicos para infraestruturas urbanas.
- Criação de políticas públicas voltadas para mobilidade e habitação.
- Monitoramento do crescimento urbano para evitar impactos ambientais negativos.

APÊNDICE O - TÓPICOS EM INTELIGÊNCIA ARTIFICIAL

A – ENUNCIADO

1) Algoritmo Genético

Problema do Caixeiro Viajante

A Solução poderá ser apresentada em: Python (preferencialmente), ou em R, ou em Matlab, ou em C ou em Java.

Considere o seguinte problema de otimização (a escolha do número de 100 cidades foi feita simplesmente para tornar o problema intratável. A solução ótima para este problema não é conhecida).

Suponha que um caixeiro deva partir de sua cidade, visitar clientes em outras 99 cidades diferentes, e então retornar à sua cidade. Dadas as coordenadas das 100 cidades, descubra o percurso de menor distância que passe uma única vez por todas as cidades e retorne à cidade de origem.

Para tornar a coisa mais interessante, as coordenadas das cidades deverão ser sorteadas (aleatórias), considere que cada cidade possui um par de coordenadas (x e y) em um espaço limitado de 100 por 100 pixels.

O relatório deverá conter no mínimo a primeira melhor solução (obtida aleatoriamente na geração da população inicial) e a melhor solução obtida após um número mínimo de 1000 gerações. Gere as imagens em 2d dos pontos (cidades) e do caminho.

Sugestão:

(1) considere o cromossomo formado pelas cidades, onde a cidade de início (escolhida aleatoriamente) deverá estar na posição 0 e 100 e a ordem das cidades visitadas nas posições de 1 a 99 deverão ser definidas pelo algoritmo genético.
(2) A função de avaliação deverá minimizar a distância euclidiana entre as cidades (os pontos).
(3) Utilize no mínimo uma população com 100 indivíduos;
(4) Utilize no mínimo 1% de novos indivíduos obtidos pelo operador de mutação;
(5) Utilize no mínimo de 90% de novos indivíduos obtidos pelo método de cruzamento (crossover-ox);
(6) Preserve sempre a melhor solução de uma geração para outra.

Importante: A solução deverá implementar os operadores de “cruzamento” e “mutação”.

2) Compare a representação de dois modelos vetoriais

Pegue um texto relativamente pequeno, o objetivo será visualizar a representação vetorial, que poderá ser um vetor por palavra ou por sentença. Seja qual for a situação, considere a quantidade de palavras ou sentenças onde tenha no mínimo duas similares e no mínimo 6 textos, que deverão produzir no mínimo 6 vetores. Também limite o número máximo, para que a visualização fique clara e objetiva.

O trabalho consiste em pegar os fragmentos de texto e codificá-las na forma vetorial. Após obter os vetores, imprima-os em figuras (plot) que demonstrem a projeção desses vetores usando a PCA.

O PDF deverá conter o código-fonte e as imagens obtidas.

B – RESOLUÇÃO

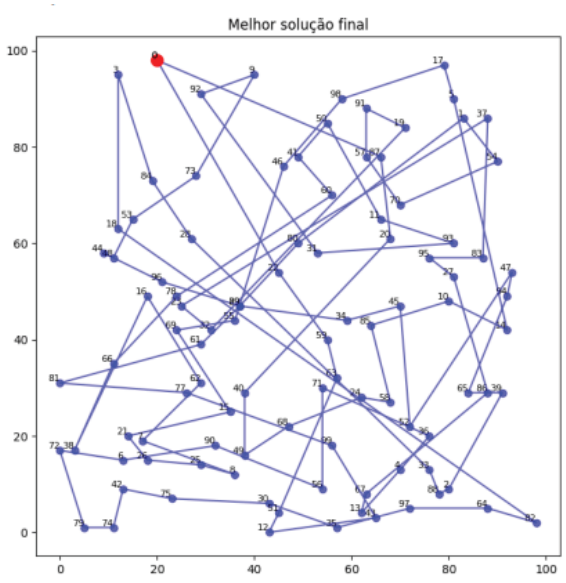
```
import random
import numpy as np
import matplotlib.pyplot as plt

NUM_CIDADES = 100  # 100 cidades
ESPACO = 100  # Espaço de 100x100 pixels
POPULACAO_SIZE = 100  # (3) População mínima de 100 indivíduos
GERACOES = 1000  # Número de gerações
TAXA_MUTACAO = 0.01  # (4) 1% de mutação
TAXA_CRUZAMENTO = 0.9  # (5) 90% de crossover

# Gerar cidades com coordenadas aleatórias dentro do espaço definido
cidades = np.random.randint(0, ESPACO, (NUM_CIDADES, 2))

# Imprimir as coordenadas das cidades geradas
print("Coordenadas das cidades:\n", cidades)

# Criar um indivíduo (permutação das cidades, sempre começando na cidade 0)
def criar_individuo():
    percurso = list(range(1, NUM_CIDADES))  # Lista de cidades sem a cidade
0
    random.shuffle(percurso)  # Embaralha as cidades
    return [0] + percurso + [0]  # Inclui a cidade 0 no início e no final

# Criar população inicial com indivíduos aleatórios
```

```
def populacao_inicial():
    return [criar_individuo() for _ in range(POPULACAO_SIZE)]

# Execução do Algoritmo Genético
# Contar o número de indivíduos na população
numero_individuos = len(populacao)

# Imprimir o número de indivíduos
print(f"Número de indivíduos na população: {numero_individuos}")

populacao = populacao_inicial()
print(populacao)

# Função para calcular a distância euclidiana entre duas cidades
def distancia(c1, c2):
    return np.linalg.norm(c1 - c2)

# Função de avaliação (fitness): distância total do percurso
def avaliacao(percurso):
    dist = 0  # Inicializa a variável dist para armazenar a soma das distâncias
    # Loop para percorrer todas as cidades no percurso (exceto a última)
    for i in range(len(percurso) - 1):
        # Calcula a distância entre a cidade atual (percurso[i]) e a próxima
cidade (percurso[i + 1])
        dist += distancia(cidades[percurso[i]], cidades[percurso[i + 1]])
    # Retorna o inverso da distância total, pois queremos um valor maior para
percursos mais curtos
    return 1 / dist  # Quanto menor a distância, melhor o fitness

# Percorrendo toda a população para encontrar o melhor percurso
for percurso in populacao:
    if avaliacao(percurso) > avaliacao(melhor_inicial):
        melhor_inicial = percurso
# melhor_inicial = max(populacao, key=avaliacao)
print(melhor_inicial)

# Função para plotar o percurso e numerar as cidades
def plotar(percurso, titulo):
    plt.figure(figsize=(8, 8))
    x, y = zip(*[cidades[i] for i in percurso])
    plt.plot(x, y, 'bo-', alpha=0.6)  # Traça o caminho
```

```
    plt.plot([x[0]], [y[0]], 'ro', markersize=10)  # Destaca a cidade inicial
    # Adiciona os números das cidades
    for i, (x_coord, y_coord) in enumerate(zip(x, y)):
        plt.text(x_coord, y_coord, str(percurso[i]), fontsize=8, ha='right',
va='bottom', color='black')
    plt.title(titulo)
    plt.show()

plotar(melhor_inicial, "Melhor solução inicial")

# Seleção dos pais por torneio (escolhe o melhor entre 5 indivíduos
aleatórios)
def selecao(populacao):
    return sorted(random.sample(populacao, 5), key=avaliacao,
reverse=True)[0]
# Cruzamento OX (Order Crossover) entre dois pais
def cruzamento(pai1, pai2):
    tamanho = len(pai1)
    inicio, fim = sorted(random.sample(range(1, tamanho - 1), 2))  # Ponto
de corte
    filho = [-1] * tamanho  # Inicializa filho com valores inválidos
    filho[inicio:fim] = pai1[inicio:fim]  # Copia segmento do primeiro pai
    p2_idx = fim  # Índice do segundo pai
    for i in range(fim, tamanho - 1):
        while pai2[p2_idx] in filho:
            p2_idx = (p2_idx + 1) % (tamanho - 1)
        filho[i] = pai2[p2_idx]
    for i in range(1, inicio):
        while pai2[p2_idx] in filho:
            p2_idx = (p2_idx + 1) % (tamanho - 1)
        filho[i] = pai2[p2_idx]
    filho[0] = filho[-1] = 0  # Mantém a cidade inicial no começo e no fim
    return filho

# Mutação (troca de duas cidades aleatórias no percurso)
def mutacao(individuo):
    if random.random() < TAXA_MUTACAO:
        i, j = random.sample(range(1, NUM_CIDADES - 1), 2)
        individuo[i], individuo[j] = individuo[j], individuo[i]

# Evolução da população através de seleção, cruzamento e mutação
```

```
def evoluir(populacao):
    nova_populacao = [max(populacao, key=avaliacao)]  # Preserva o melhor
indivíduo
    num_filhos = int(TAXA_CRUZAMENTO * POPULACAO_SIZE)
    while len(nova_populacao) < num_filhos:
        pai1, pai2 = selecao(populacao), selecao(populacao)
        filho = cruzamento(pai1, pai2)
        mutacao(filho)
        nova_populacao.append(filho)
    while len(nova_populacao) < POPULACAO_SIZE:
        nova_populacao.append(criar_individuo())       #    Adiciona    novos
indivíduos aleatórios
    return nova_populacao
# Loop de gerações para evoluir a população
for i in range(GERACOES):
    populacao = evoluir(populacao)
    if i % 100 == 0:  # Exibe progresso a cada 100 gerações
        melhor = max(populacao, key=avaliacao)
        print(f'Geração {i}: Melhor distância = {1 / avaliacao(melhor):.2f}')
# Exibir a melhor solução final
top_individuo = max(populacao, key=avaliacao)
plotar(top_individuo, "Melhor solução final")
```

FIGURA 33 – MELHOR SOLUÇÃO PARA O PROBLEMA DO CAIXEIRO VIAJANTE



FONTE: De autoria própria (2025)

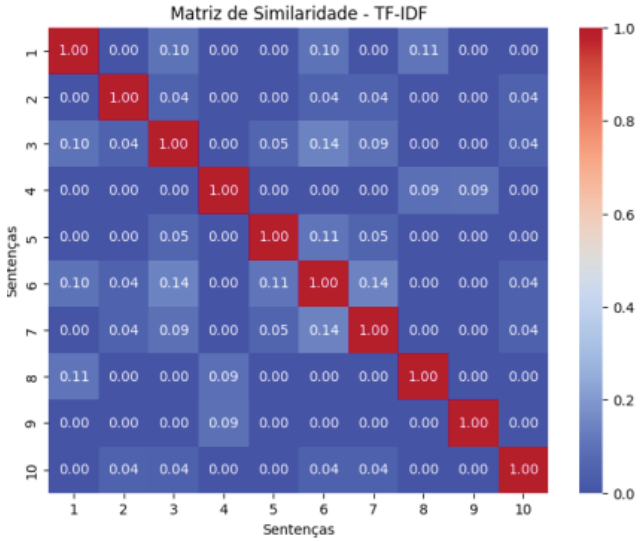
```
# ============================
# Instalação de bibliotecas (Colab)
# ============================
!pip install -q sentence-transformers scikit-learn

# ============================
# Imports
# ============================
from sklearn.feature_extraction.text import TfidfVectorizer
from sentence_transformers import SentenceTransformer
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# ============================
# Entrada do usuário
# ============================
def verficar_qtde_sentencas_validas():
    while True:
        print("Digite um texto com até 1000 caracteres (mínimo 6 frases com
algumas parecidas):\n")
        texto = input()[:1000]
        # Fragmentar em sentenças
        import re
        sentencas = re.split(r'(?<=[.!?])\s+', texto.strip())
        # Filtro: entre 6 e 10 sentenças
        sentencas = [s for s in sentencas if s]  # Remover sentenças vazias
        if len(sentencas) < 6:
            print("  Você  precisa  digitar  pelo  menos  6  sentenças  para
continuar.\n")
            continue
        sentencas = sentencas[:10]  # limite para clareza do gráfico
        print("\n Sentenças selecionadas:")
        for i, s in enumerate(sentencas):
            print(f"{i+1}. {s}")
        return sentencas

sentencas = verficar_qtde_sentencas_validas()

# ============================
```

```
# Modelo 1: TF-IDF
# ============================
vectorizer = TfidfVectorizer()
tfidf_vectors = vectorizer.fit_transform(sentencas).toarray()

# ============================
# Modelo 2: Sentence Embedding (BERT)
# ============================
model = SentenceTransformer('all-MiniLM-L6-v2')
bert_vectors = model.encode(sentencas)

# ============================
# Calcular Similaridades
# ============================
def calc_similaridade_tfidf(vetores, limiar=0.6):
    matriz_similaridade_tdidf = cosine_similarity(vetores)
    np.fill_diagonal(matriz_similaridade_tdidf, 0)  # Zerar diagonal para não
contar similaridade
    return (matriz_similaridade_tdidf >= limiar).sum() >= 2  # Pelo menos 2
sentenças com similaridade

def calc_similaridade_bert(vetores, limiar=0.7):
    matriz_similaridade_bert = cosine_similarity(vetores)
    np.fill_diagonal(matriz_similaridade_bert, 0)  # Zerar diagonal para não
contar similaridade
    return (matriz_similaridade_bert >= limiar).sum() >= 2  # Pelo menos 2
sentenças com similaridade

# Função para exibir mapa de calor
import seaborn as sns
def plot_heatmap(sim_matrix, titulo):
    plt.figure(figsize=(8, 6))
    sns.heatmap(
        sim_matrix,
        annot=True,
        cmap="coolwarm",
        fmt=".2f",
        xticklabels=range(1, len(sim_matrix) + 1),
        yticklabels=range(1, len(sim_matrix) + 1),
    )
    plt.title(f"Matriz de Similaridade - {titulo}")
```

```
    plt.xlabel("Sentenças")
    plt.ylabel("Sentenças")
    plt.show()

# Verificar se pelo menos 2 sentenças têm similaridade
while True:
    tfidf_sim_matrix = cosine_similarity(tfidf_vectors)
    bert_sim_matrix = cosine_similarity(bert_vectors)
    plot_heatmap(tfidf_sim_matrix, "TF-IDF")
    plot_heatmap(bert_sim_matrix, "BERT")
    if               calc_similaridade_tfidf(tfidf_vectors)               or
calc_similaridade_bert(bert_vectors):
        break  # Se pelo menos um critério for atendido, continue
    print("\n O texto não tem pelo menos 2 sentenças similares (≥ 0.6 para
TF-IDF ou ≥ 0.7 para BERT). Tente novamente.\n")
    # Solicitar novas sentenças e recalcular vetores
    sentencas = verficar_qtde_sentencas_validas()
    tfidf_vectors = vectorizer.fit_transform(sentencas).toarray()
    bert_vectors = model.encode(sentencas)

# ============================
# Redução PCA
# ============================
def reduzir_pca(vetores):
    pca = PCA(n_components=2)
    return pca.fit_transform(vetores)

tfidf_2d = reduzir_pca(tfidf_vectors)
bert_2d = reduzir_pca(bert_vectors)

# ============================
# Plot dos Vetores (TF-IDF vs BERT)
# ============================
def plot_vetores(vetores_2d, sentencas, titulo):
    plt.figure(figsize=(8, 6))
    for i, coord in enumerate(vetores_2d):
        plt.scatter(coord[0], coord[1], label=f"{i+1}")
        plt.text(coord[0] + 0.02, coord[1] + 0.02, f"{i+1}", fontsize=9)
    plt.title(f'Projeção PCA - {titulo}')
    plt.xlabel('Componente Principal 1')
    plt.ylabel('Componente Principal 2')
```

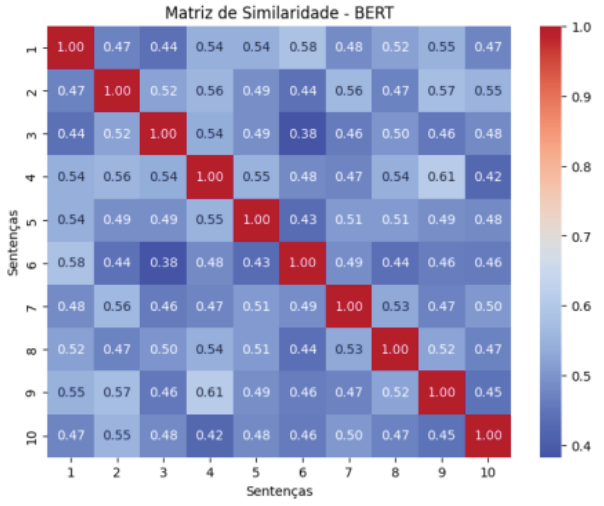
```
    plt.grid(True)
    plt.legend()
    plt.show()

plot_vetores(tfidf_2d, sentencas, "TF-IDF")
plot_vetores(bert_2d, sentencas, "BERT Embedding")

Digite um texto com até 1000 caracteres (mínimo 6 frases com algumas parecidas): O pôr do sol na praia era deslumbrante. As ondas quebravam suavemente, criando um som relaxante. Um grupo de amigos ria enquanto jogava bola na areia. Crianças corriam perto da água, desenhando formas com os pés. O céu exibia tons vibrantes de laranja, rosa e roxo. Era o tipo de cenário que parecia saído de um filme. Mais adiante, um casal caminhava de mãos dadas, trocando sorrisos. O vento trazia o cheiro do mar e da brisa salgada. Os pássaros voavam em formação, cruzando o horizonte. Um senhor tocava violão, cantando baixinho uma melodia nostálgica.

Sentenças selecionadas:
1. O pôr do sol na praia era deslumbrante.
2. As ondas quebravam suavemente, criando um som relaxante.
3. Um grupo de amigos ria enquanto jogava bola na areia.
4. Crianças corriam perto da água, desenhando formas com os pés.
5. O céu exibia tons vibrantes de laranja, rosa e roxo.
6. Era o tipo de cenário que parecia saído de um filme.
7. Mais adiante, um casal caminhava de mãos dadas, trocando sorrisos.
8. O vento trazia o cheiro do mar e da brisa salgada.
9. Os pássaros voavam em formação, cruzando o horizonte
10. Um senhor tocava violão, cantando baixinho uma melodia nostálgica.
```

FIGURA 34 – MATRIZ DE SIMILARIDADE TF IDF DA PRIMEIRA ENTRADA



FONTE: De autoria própria (2025)

FIGURA 35 – MATRIZ DE SIMILARIDADE BERT DA PRIMEIRA ENTRADA

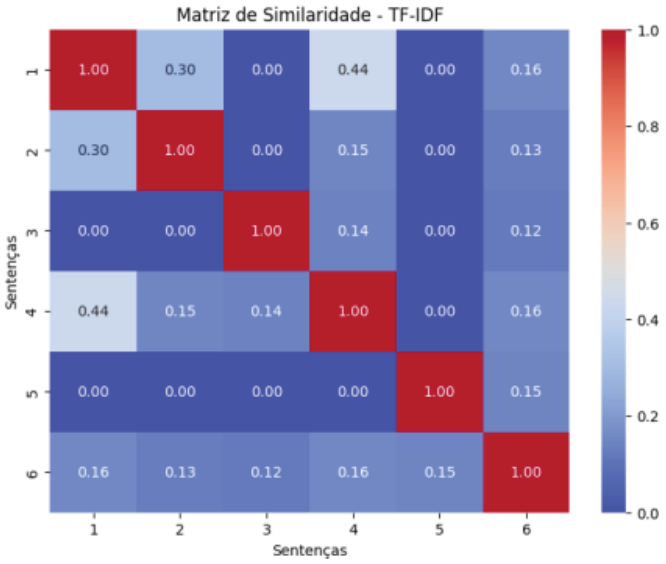


FONTE: De autoria própria (2025)

```
O texto não tem pelo menos 2 sentenças similares (≥ 0.6 para TF-IDF ou ≥ 0.7 para BERT). Tente novamente.
Digite um texto com até 1000 caracteres (mínimo 6 frases com algumas parecidas):
O gato preto pulou o muro. O gato preto saltou sobre a cerca. O cachorro marrom correu pelo jardim. O cachorro preto pulou o portão. O pássaro azul voou alto no céu. O gato e o cachorro brincaram no quintal.
```

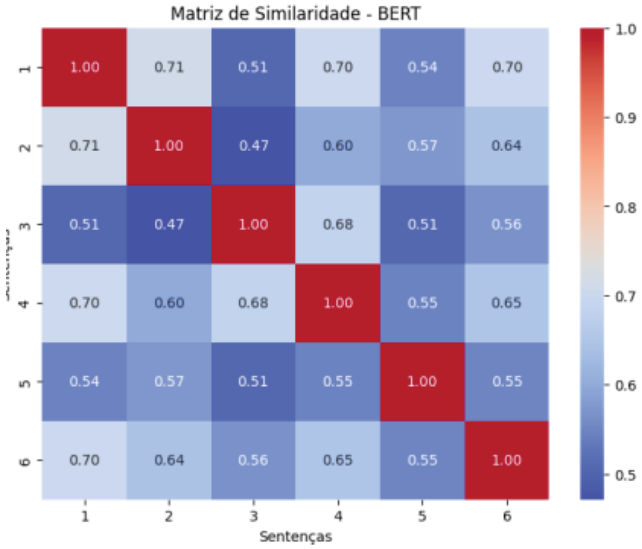
```
Sentenças selecionadas:
1. O gato preto pulou o muro.
2. O gato preto saltou sobre a cerca.
3. O cachorro marrom correu pelo jardim.
4. O cachorro preto pulou o portão.
5. O pássaro azul voou alto no céu.
6. O gato e o cachorro brincaram no quintal.
```

FIGURA 36 – MATRIZ DE SIMILARIDADE TF IDF DA SEGUNDA ENTRADA



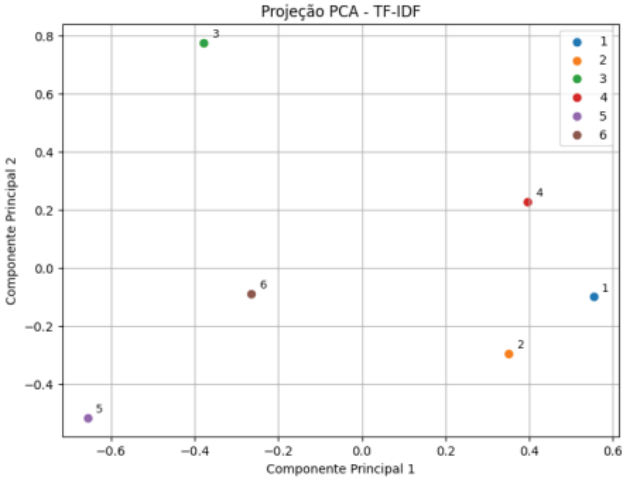
FONTE: De autoria própria (2025)

FIGURA 37 – MATRIZ DE SIMILARIDADE BERT DA SEGUNDA ENTRADA



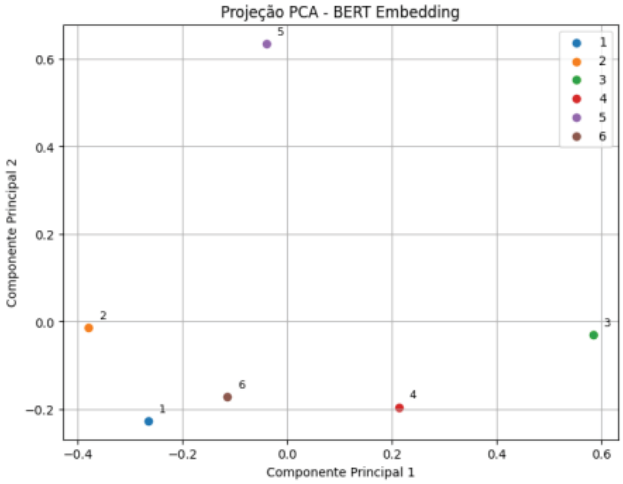
FONTE: De autoria própria (2025)

FIGURA 38 – PROJEÇÃO PCA TF-IDF



FONTE: De autoria própria (2025)

FIGURA 39 – PROJEÇÃO PCA BERT



FONTE: De autoria própria (2025)