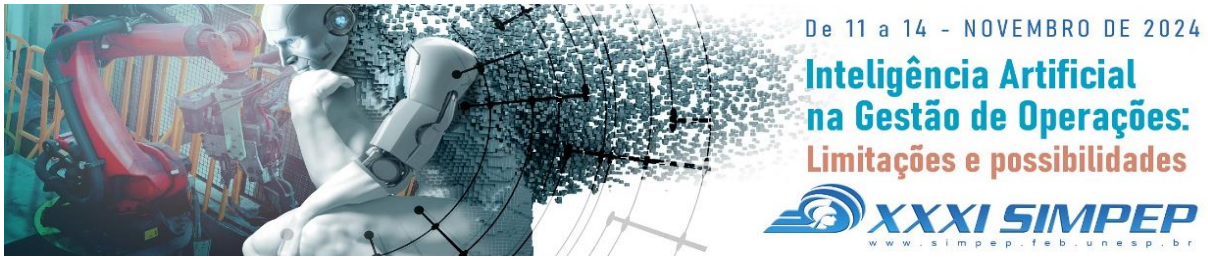


INTELIGÊNCIA ARTIFICIAL E MACHINE LEARNING: UM ESTUDO COMPARATIVO DOS MODELOS DE APRENDIZAGEM DE MÁQUINA FLORESTA ALEATÓRIA E REDES NEURAIS

ALFREDO NAZARENO PEREIRA BOENTE - professor@boente.eti.br
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - HCTE-UFRJ

MARIANNE LUANA BUENO - mariannebueno@live.com
UNIVERSIDADE VEIGA E ALMEIDA - UVA

ÁREA: *03. PESQUISA OPERACIONAL*
SUBÁREA: *3.7 - INTELIGÊNCIA COMPUTACIONAL*

RESUMO: *ESTE ESTUDO UTILIZOU TÉCNICAS DE APRENDIZADO DE MÁQUINA, SUBCAMPO DA INTELIGÊNCIA ARTIFICIAL, PARA ANALISAR A INFLUÊNCIA QUE O CONTEXTO SOCIAL DE UM CANDIDATO DO EXAME NACIONAL DO ENSINO MÉDIO (ENEM) PODE EXERCER EM SUA PONTUAÇÃO NA DISCIPLINA DE MATEMÁTICA E SUAS TECNOLOGIAS. MICRODADOS ABERTOS DO ENEM DO ANO DE 2022, QUE ENGLOBAM AS VARIÁVEIS SOCIAIS, EDUCACIONAIS E ECONÔMICAS DOS CANDIDATOS, FORAM COLETADOS E TRATADOS NESTE ESTUDO, POR MEIO DA APLICAÇÃO DE MODELOS DE APRENDIZADO DE MÁQUINA, FLORESTA ALEATÓRIA E REDES NEURAIS, A FIM DE AVALIAR E COMPARAR AS TÉCNICAS SEGUINDO CRITÉRIOS DE PRECISÃO, ROBUSTEZ E INTERPRETABILIDADE DOS DADOS. POR FIM, FOI APRESENTADO O MODELO QUE DEMONSTROU O MELHOR RESULTADO NA COMPREENSÃO DA RELAÇÃO ENTRE OS ELEMENTOS ESTUDADOS, A FIM DE IDENTIFICAR AS VARIÁVEIS DE INFLUÊNCIA E PROMOVER UMA REFLEXÃO SOBRE A EFETIVIDADE DA ATUAÇÃO DO GOVERNO NA IMPLEMENTAÇÃO DE MELHORIAS EDUCACIONAIS NO BRASIL.*

PALAVRAS-CHAVES: *RANDOM FLOREST; ARTIFICIAL INTELLIGENCE; NEURAL NETWORKS; MACHINE LEARNING; DEEP LEARNING.*

1. INTRODUÇÃO

O Exame Nacional do Ensino Médio (ENEM) se tornou a principal porta de entrada para a população ao ensino superior no Brasil. As ações afirmativas implementadas pelo governo em programas que aderem as notas do exame para contemplação das vagas ofertadas, têm favorecido a diversidade social dentro das instituições públicas e privadas do país. Nesse cenário, apesar de tais ações contribuírem para a democratização do acesso ao ensino superior, a pluralização dos contextos sociais aos quais os estudantes estão contidos permanece dispare em termos de desempenho nas notas obtidas no exame, sobretudo na disciplina de matemática e suas tecnologias.

Durante a última década, segundo o Censo da Educação Superior de 2022 realizado pelo Instituto Nacional de Estudos e Pesquisas Educacionais (Inep), o número de ingressos na educação superior federal por meio de ações afirmativas aumentou em 167%. Nesse mesmo ano, dados coletados pelo PISA (Programa Internacional de Avaliação de Alunos) indicam que o Brasil está entre os 20 países com pior desempenho em matemática, alcançando a pontuação de 379, abaixo da média de 479 registrada pelos países da OCDE[1].

Os resultados demonstram que, embora o acesso da população às vagas de graduação tenha sido flexibilizado, a educação básica no país ainda enfrenta inúmeros desafios para proporcionar um ensino de qualidade aos estudantes. Tais desafios podem se estender e englobar outros elementos sociais. Segundo o levantamento do Todos Pela Educação feito com base em dados de 2012 a 2022 do IBGE, o percentual de jovens negros com acesso à educação na idade certa foi equiparado aos alunos brancos que possuíam dez anos antes.

Com base nos dados apresentados, entende-se que diversos aspectos podem impactar diretamente a jornada de preparação dos estudantes até a prestação do vestibular, considerando a distinção do ambiente ao qual estes estiveram ou estão inseridos unido as oportunidades e experiências de contextos sociais ao longo de suas vidas. Desse modo, o reflexo dessa desigualdade é evidenciado, principalmente, no desempenho aplicado no exame

[1] A Organização para a Cooperação e o Desenvolvimento Econômico (OCDE), segundo o MEC, é composto por 35 países que cooperam entre si na busca de promover meios que melhorem as políticas públicas globais, como questões sociais, econômicas, educacionais e ambientais.

do ENEM. Faz-se necessário, portanto, analisar por meio científico quais elementos interferem na proficiência dos estudantes aplicada na prova do ENEM.

2. REFERENCIAL TEÓRICO

2.1 Inteligência Artificial e Machine Learning

Na década de 40, o neurofisiologista Warren McCulloch e o matemático Walter Pitts estudavam sobre como o neurônio biológico funcionava no cérebro humano. A fim de reproduzir o mecanismo em modelos computacionais, a dupla propôs seu primeiro modelo artificial de neurônios no artigo "*A logical calculus of the ideas immanent in nervous activity*" (MCCULLOCH e PITTS, 1943). Embora o termo ainda não existisse, o modelo de McCulloch e Pitts foi considerado a primeira inteligência artificial, estabelecendo bases para pesquisas em redes neurais e pavimentando caminho para a revolução tecnológica.

A inteligência artificial, conforme afirma Taulli (2020), é um campo da ciência da computação que se dedica ao estudo e ao desenvolvimento de máquinas e programas computacionais capazes de reproduzir o comportamento humano na tomada de decisões e na realização de tarefas, desde as mais simples até as mais complexas.

Machine Learning pode ser definido como modelo matemático aplicados em dados brutos para a realização de predições. Esse tipo de modelo pode ser dividido em duas grandes categorias: modelo supervisionado e modelo não-supervisionado (BREIMAN, 2001).

2.2 Modelos Supervisionados X Modelos Não-Supervisionados

Em modelos supervisionados, o algoritmo é treinado a partir de um conjunto de dados composto por pares constituídos de uma ou mais variáveis de entrada (*inputs*) e uma variável de saída (*output* ou *target*). Chamamos estes de dados rotulados, pois é sabido de antemão as saídas esperadas para as entradas.

Assim, ao treinarmos um modelo, os dados de saída servem como "supervisores" para comparar os dados previstos e orientar o modelo através de medidas de erro entre as previsões e os rótulos verdadeiros, como a função de perda.

Em modelos não supervisionados, o algoritmo é treinado com dados não rotulados, ou seja, em conjuntos onde não há uma variável de saída como referência. Nesse cenário, o modelo estrutura os dados e aplica previsões descobrindo padrões entre as variáveis de entrada utilizando técnicas que não dependam de orientação, como a clusterização ou redução de dimensionalidade (BREIMAN, 2001).

2.3 Algoritmos de Classificação x Algoritmos de Regressão

Os algoritmos de classificação possuem variáveis de destino categóricas, por meio de cálculos matemáticos de probabilidade para cada classe. Nesse formato, os modelos aprendem com os dados a classificar uma observação com base nas características (variáveis de entrada) apresentadas. Por exemplo, um banco pode analisar através do histórico do cliente, se este está apto ou não a realizar um empréstimo.

Por outro lado, os algoritmos que possuem variáveis de destino contínuas (valores previstos ao qual não se tem controle), como o valor do empréstimo disponibilizado pelo banco para o cliente, são chamados de algoritmos de regressão (LIAW e WIENER, 2002).

2.4 Dados Abertos do ENEM e Definição dos Modelos

O Inep (Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira) começou a disponibilizar dados abertos do Exame Nacional do Ensino Médio (ENEM) a partir do ano de 2009. A iniciativa de disponibilizar esses dados visa promover a transparência e o acesso à informação. Desa forma, foram utilizados os dados do ENEM referente ao ano de 2022 em modelos de *machine learning* para observarmos o comportamento dos dados e suas relações (INEP, 2024).

O conjunto de dados possui 76 (setenta e seis) variáveis, divididas por dados do participante, dados da escola, dados do local de aplicação da prova, dados da prova objetiva, dados da redação e dados do questionário socioeconômico. Das variáveis disponíveis, 17 (dezessete) foram escolhidas como variáveis de entrada para os modelos de Floresta Aleatória e Redes Neurais[2], aplicados por meio da linguagem de programação Python.

2.5 Árvores de Decisão e Floresta Aleatória

A Árvore de Decisão é uma técnica que delineia hierarquicamente no formato de árvore uma representação visual e lógica das trajetórias de decisões tomadas e seus resultados.

[2] Apesar dos modelos escolhidos serem efetivos tanto em problemas de classificação como em regressão, este trabalho se restringirá em abordar somente conceitos relacionados à regressão, como meio de explorar apenas os aspectos específicos do que será aplicado.

Sua estrutura inicia-se com um único nó chamado de "nó-raiz", que se ramifica, através de ações ou respostas ao nó anterior, em possíveis resultados (NAPOLEÃO, 2024). Esses resultados podem adicionar outros nós à estrutura, chamados de "nós de decisão", que crescem a árvore dependendo do seu nível de complexidade. Ao final, tem-se os nós resultantes do conjunto de decisões tomadas, conhecidos como "nós-folhas".

No contexto de *machine learning*, a Árvore de Decisão é um modelo preditivo supervisionado proposto por Breiman (1996) conhecido como CART (*Classification and Regression Tree).* O algoritmo mapeia um conjunto de entradas para uma saída através de uma série de decisões baseadas nas características dessas entradas.

A Floresta Aleatória (*Random Forest*, em inglês) é um algoritmo desenvolvido por Leo Breiman como meio de solucionar a limitação de *overfitting* que a Árvore de Decisão traz em seu modelo. Para isso, o algoritmo utiliza o método *ensemble*, onde diversos modelos operam em conjunto para obtenção de um resultado final único. No caso da Floresta Aleatória, os modelos aplicados são as Árvores de Decisão (BREIMAN, 2001).

O modelo de Floresta Aleatória apresenta uma previsão com base na média de diversos resultados preditivos gerados por árvores individuais, numa saída confiável e precisa.

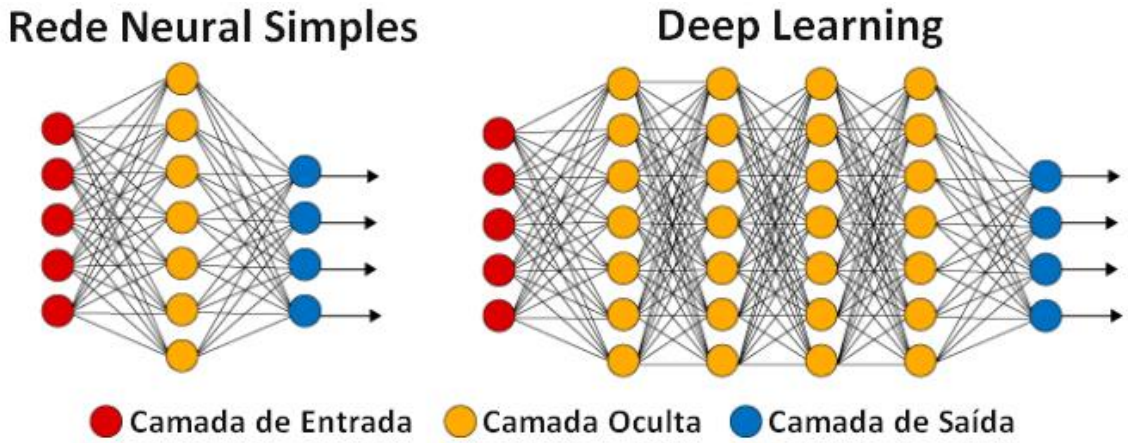
2.6 Redes Neurais e Deep Learning

Em 1958, Frank Rosenblattem propôs um algoritmo de rede neural inspirado no funcionamento dos neurônios no cérebro humano. Originalmente voltado a classificação binária de saída única, o modelo matemático ficou reconhecido como pioneiro na aprendizagem de máquina, entitulado "*Perceptron*". Apesar do surgimento anterior de outros algoritmos e das limitações que o seu possuía envolvendo problemas complexos, o modelo exerceu grande influência histórica devido à teoria matemática sólida, ao potencial percebido e às aplicações práticas (ROSENBLATTEM, 1958).

O algoritmo é composto por elementos que fazem uma analogia à estrutura de um neurônio biológico. Na parte inicial do neurônio estão os dendritos, ramificações que recebem sinais elétricos de outros neurônios. Estes, são representados no modelo por dados de entradas (atributos) que o *perceptron* processa. As sinapses são as conexões entre neurônios, representadas por pesos que determinam a relevância de cada entrada. O potencial de repouso corresponde ao bias, valor constante adicionado à soma ponderada das entradas, que ajustam a fronteira de decisão. Por fim, temos o axônio, que transmite o sinal elétrico quando o neurônio atinge um determinado limiar de ativação.

A diferença entre um modelo de rede neural simples e um modelo de *deep learning* é a quantidade de camadas ocultas implementadas em seu algoritmo (GOODFELLOW, BENGIO e COURVILLE, 2017). Enquanto uma rede neural simples consiste em apenas uma camada de entrada, uma camada oculta e uma camada de saída, um modelo de *deep learning* pode utilizar duas ou mais camadas, como ilustrado na figura a seguir.

FIGURA 1 - Comparação entre o modelo de rede neural simples e o modelo de *deep learning*.



Fonte: DATA SCIENCE ACADEMY (2024).

3. PROCEDIMENTOS METODOLÓGICOS

Para contribuir com o levantamento de dados sociais que contextualizam e enriquecem o tema proposto, foram utilizados dados publicados por instituições oficiais. A fundamentação técnica do estudo foi embasada em artigos publicados por referências na área de Inteligência Artificial, além de livros relacionados a modelos de *machine learning* e suas aplicações em Python (BUENO e BOENTE, 2024).

O estudo aplica a análise exploratória de dados abertos do ENEM, referentes ao ano de 2022. A análise envolve uma investigação detalhada dos dados a fim de compreendê-los melhor, avaliando sua estrutura, os tipos de variáveis, a presença de valores nulos ou ausentes, a quantidade de observações coletadas e possíveis inconsistências.

Utilizou-se uma amostra com base de dados de todos os candidatos que realizaram o ENEM no ano de 2022, que foi representativa do universo estudado, visto que foram consideradas apenas observações onde nenhuma de suas variáveis fossem vazias ou nulas e candidatos com a situação de ensino médio como “concluído” ou “a concluir”.

O estudo apresentou uma abordagem quantitativa, visto que são analisadas variáveis numéricas e categóricas dos candidatos.

Os meios de investigação utilizados incluíram técnicas estatísticas e de *machine learning* para analisar os dados de forma abrangente e detalhada. Isso envolveu a aplicação de métodos estatísticos descritivos para compreender a distribuição e as características das variáveis, bem como a utilização de modelos de regressão e outras técnicas de análise quantitativa para explorar relações entre as variáveis e as notas dos candidatos.

4. RESULTADOS E DISCUSSÕES

4.1 Implementação do Modelo Floresta Aleatória

Para aplicação dos modelos apresentados neste estudo, foi utilizada a linguagem de programação Python, com o ambiente de desenvolvimento Google Colab, cuja implementação do modelo Floresta Aleatória, sua estruturação, manipulação e visualização de dados, bem como para aplicação do modelo de aprendizagem de máquina, foram utilizadas as seguintes bibliotecas: Pandas, NumPy, Seaborn, Matplotlib e Scikit-learn.

FIGURA 2 - Trecho do programa que realiza a ingestão e o tratamento de dados.

```
# Implementação do Modelo Floresta Aleatória

# > bibliotecas
import pandas as pd # estruturação e manipulação de dados
import numpy as np # processamento de dados
import matplotlib.pyplot as plt #  interpretação de modelos e visualização de dados
import seaborn as sns #  interpretação de modelos e visualização de dados
from sklearn.ensemble import RandomForestRegressor # ML de floresta aleatória
from sklearn.model_selection import train_test_split # treino e teste de dados
from sklearn.metrics import r2_score # R2
from sklearn.metrics import mean_squared_error # MSE
import shap #  interpretação de modelos e visualização de dados

# > ingestão de dados
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/microdados_enem_2022.csv', sep=';', encoding='latin-1',
     usecols=['TP_FAIXA_ETARIA', 'TP_SEXO', 'TP_ESTADO_CIVIL', 'TP_COR_RACA', 'TP_ST_CONCLUSAO', 'CO_UF_ESC',
              'TP_DEPENDENCIA_ADM_ESC', 'NU_NOTA_MT', 'Q001', 'Q002', 'Q003', 'Q004', 'Q005', 'Q006', 'Q010',
              'Q024', 'Q025']) # criação de dataframe

# > tratamento de dados
df = df[df['TP_ST_CONCLUSAO'].isin([1, 2])] # filtra campos preenchidos com 1 (concluído) e 2 (a concluir)

df = df.rename(columns = {'TP_FAIXA_ETARIA':'FAIXA_ETARIA', 'TP_SEXO':'SEXO', 'TP_ESTADO_CIVIL':'ESTADO_CIVIL',
                          'TP_COR_RACA':'COR_RACA', 'TP_ST_CONCLUSAO':'ST_CONCLUSAO', 'CO_UF_ESC':'COD_MUN',
                          'TP_DEPENDENCIA_ADM_ESC':'DEP_ESC', 'NU_NOTA_MT':'NOTA_MT', 'Q001':'ESC_PAI',
                          'Q002':'ESC_MAE', 'Q003':'PRF_PAI', 'Q004':'PRF_MAE', 'Q005':'QNT_RES',
                          'Q006':'RENDA_MENSAL', 'Q010':'POSSUI_CARRO', 'Q024':'POSSUI_COMPU',
                          'Q025':'POSSUI_INTER'})

```

Fonte: Elaboração própria.

A Figura 2, ilustra o processo de ingestão dos dados a partir da função pd.read_csv() da biblioteca Pandas. O primeiro argumento é dado pelo caminho do Google Drive para carregar os dados do arquivo 'microdados_enem_2022.csv'. Em seguida, definimos o

parâmetro sep como ';' para indicar que o arquivo está separado por ponto e vírgula. O parâmetro *encoding* é especificado como 'latin-1' para garantir a correta leitura de caracteres especiais. Por fim, utilizamos o parâmetro usecols para selecionar apenas as colunas relevantes para o nosso estudo. O resultado da função é atribuído à variável ‘df’.

Realizamos em seguida o tratamento dos dados para garantir a sua consistência e adequação ao modelo proposto. Nesta etapa, aplicamos filtros e renomeações de colunas para facilitar a compreensão e manipulação dos dados. Após a aplicação do filtro, renomeamos as colunas do *dataframe* utilizando o método rename(), atribuindo nomes mais descritivos e compreensíveis às variáveis.

Em seguida, aplicamos um tratamento para lidar com valores ausentes utilizando a função dropna(), removendo registros que possuem valores nulos em colunas específicas.

FIGURA 3 - Trecho do programa que realiza o tratamento de valores ausentes.

```
# info: tot = 3.476.105 linhas · 695.008 linhas
# remoção de valores nulos
df = df.dropna(subset = ['NOTA_MT'])
df = df.dropna(subset = ['FAIXA_ETARIA'])
df = df.dropna(subset = ['SEXO'])
df = df.dropna(subset = ['ESTADO_CIVIL'])
df = df.dropna(subset = ['COR_RACA'])
df = df.dropna(subset = ['ST_CONCLUSAO'])
df = df.dropna(subset = ['COD_MUN'])
df['REGIAO'] = df['COD_MUN'].apply(lambda x: int(str(x)[0])) # captura primeiro dígito do código
df = df.drop('COD_MUN', axis = 1) # descarta coluna que não será utilizada
df = df.dropna(subset = ['DEP_ESC'])
df = df.dropna(subset = ['ESC_PAI'])
df = df.dropna(subset = ['ESC_MAE'])
df = df.dropna(subset = ['PRF_PAI'])
df = df.dropna(subset = ['PRF_MAE'])
df = df.dropna(subset = ['QNT_RES'])
df = df.dropna(subset = ['RENDA_MENSAL'])
df = df.dropna(subset = ['POSSUI_CARRO'])
df = df.dropna(subset = ['POSSUI_COMPU'])
df = df.dropna(subset = ['POSSUI_INTER'])

```

Fonte: Elaboração própria.

Para facilitar a manipulação e análise dos dados, realizamos o mapeamento de algumas variáveis categóricas para valores numéricos. Criamos variáveis correspondentes às características e definimos dicionários de mapeamento para cada uma delas.

A partir de então, aplicamos esses mapeamentos às respectivas colunas do conjunto de dados utilizando o método map().

FIGURA 4 - Trecho do programa que realiza os mapeamentos definidos no dicionário.

```
# criação de dicionário
sexo = {'M': 1, 'F': 0}
escolaridade = {'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7, 'H': 0}
profissao = {'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 0}
regiao = {1: 'N', 2: 'NE', 3:'SE', 4:'S', 5:'CO'}
renda = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8, 'J': 9,
         'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O': 14, 'P': 15, 'Q': 16}
possui_carro_compu = {'A': 0, 'B': 1, 'C': 1, 'D': 1, 'E': 1}
possui_inter = {'A': 0, 'B': 1}

# aplicação do mapeamento definido no dicionário
df['SEXO'] = df['SEXO'].map(sexo)
df['ESC_PAI'] = df['ESC_PAI'].map(escolaridade)
df['ESC_MAE'] = df['ESC_MAE'].map(escolaridade)
df['PRF_PAI'] = df['PRF_PAI'].map(profissao)
df['PRF_MAE'] = df['PRF_MAE'].map(profissao)
df['REGIAO'] = df['REGIAO'].map(regiao)
df['RENDA_MENSAL'] = df['RENDA_MENSAL'].map(renda)
df['POSSUI_CARRO'] = df['POSSUI_CARRO'].map(possui_carro_compu)
df['POSSUI_COMPU'] = df['POSSUI_COMPU'].map(possui_carro_compu)
df['POSSUI_INTER'] = df['POSSUI_INTER'].map(possui_inter)

```

Fonte: Elaboração própria.

Em seguida, utilizamos a função pd.get_dummies() da biblioteca Pandas para realizar a codificação *One-Hot Encoding* das variáveis categóricas selecionadas. Essa abordagem nos permite preservar a informação contida nas variáveis categóricas sem impor uma ordem hierárquica entre as categorias.

FIGURA 5 - Trecho do programa que cria coluna dammy para valores binários.

```
# criação de colunas dummy com valores binários
df = pd.get_dummies(df, columns = ['SEXO'], prefix = ['SEXO'])
df = pd.get_dummies(df, columns = ['COR_RACA'], prefix = ['COR_RACA'])
df = pd.get_dummies(df, columns = ['ESTADO_CIVIL'], prefix = ['ESTADO_CIVIL'])
df = pd.get_dummies(df, columns = ['DEP_ESC'], prefix = ['DEP_ESC'])
df = pd.get_dummies(df, columns = ['REGIAO'], prefix = ['REGIAO'])

```

Fonte: Elaboração própria.

Por fim, renomeamos as colunas binárias criadas para refletir de forma mais clara e intuitiva as categorias originais das variáveis categóricas.

FIGURA 6 - Trecho do programa que renomeia as colunas dummy.

```
# renomeio das colunas dummy
df = df.rename(columns = {'SEXO_0':'SEXO_F', 'SEXO_1':'SEXO_M'})
df = df.rename(columns = {'COR_RACA_0':'COR_RACA_ND', 'COR_RACA_1':'COR_RACA_BR',
                          'COR_RACA_2':'COR_RACA_PR', 'COR_RACA_3':'COR_RACA_PD',
                          'COR_RACA_4':'COR_RACA_AM', 'COR_RACA_5':'COR_RACA_IN',
                          'COR_RACA_6':'COR_RACA_NDI'})
df = df.rename(columns = {'REGIAO_1':'REGIAO_N', 'REGIAO_2':'REGIAO_NE',
                          'REGIAO_3':'REGIAO_SE', 'REGIAO_4':'REGIAO_S',
                          'REGIAO_5':'REGIAO_CO'})
df = df.rename(columns = {'DEP_ESC_1.0':'DEP_FE', 'DEP_ESC_2.0':'DEP_ES',
                          'DEP_ESC_3.0':'DEP_MU', 'DEP_ESC_4.0':'DEP_PR'})
df = df.rename(columns = {'ESTADO_CIVIL_0':'ESTADO_CIVIL_NI', 'ESTADO_CIVIL_1':'ESTADO_CIVIL_SO',
                          'ESTADO_CIVIL_2':'ESTADO_CIVIL_CA', 'ESTADO_CIVIL_3':'ESTADO_CIVIL_DI',
                          'ESTADO_CIVIL_4':'ESTADO_CIVIL_VI'})

```

Fonte: Elaboração própria.

Dividimos o conjunto de dados em ‘X’ para as variáveis de entrada (características) e ‘y’ para a variável de saída (a que desejamos realizar as previsões). Também, utilizamos a função train_test_split() do Scikit-learn para dividir as variáveis de entrada ‘X’ e a variável de saída ‘y’ em conjuntos de treinamento (X_train e y_train) e teste (X_test e y_test). Definimos o parâmetro test_size como 0.25, indicando que 25% dos dados serão reservados para teste, enquanto 75% serão utilizados para treinamento do modelo.

Em seguida, instanciamos pela variável ‘rf’ a classe RandomForestRegressor do Scikit-learn para criar o modelo de regressão.

FIGURA 7 - Trecho do programa que faz a divisão dos dados para treinamento do modelo.

```
# > divisão das variáveis de entrada (explicativas/independentes) e alvo (resposta/dependente)
X = df[['FAIXA_ETARIA', 'SEXO_F', 'SEXO_M', 'COR_RACA_ND', 'COR_RACA_BR', 'COR_RACA_PR', 'COR_RACA_PD',
        'COR_RACA_AM', 'COR_RACA_IN', 'ESTADO_CIVIL_NI', 'ESTADO_CIVIL_SO', 'ESTADO_CIVIL_CA', 'ESTADO_CIVIL_DI',
        'ESTADO_CIVIL_VI', 'ST_CONCLUSAO', 'DEP_FE', 'DEP_ES', 'DEP_MU', 'DEP_PR', 'REGIAO_N', 'REGIAO_NE', 'REGIAO_SE',
        'REGIAO_S', 'REGIAO_CO', 'ESC_PAI', 'ESC_MAE', 'PRF_PAI', 'PRF_MAE', 'QNT_RES', 'RENDA_MENSAL', 'POSSUI_CARRO',
        'POSSUI_COMPU', 'POSSUI_INTER']] # variáveis de entrada (x)
y = df[['NOTA_MT']] # variável alvo (y)

# > divisão dos dados para treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
# variáveis explicativas de treino, variáveis explicativas de teste, variável resposta de treino, variável resposta de te
# 75% dos dados são alocados para treino
# 25% dos dados são alocados para teste

# > instanciamento e treinamento do modelo
# criação da instância do modelo rf e ajuste de hiperparâmetros
rf = RandomForestRegressor(n_estimators=1000, random_state=42, verbose=1, n_jobs=-1)
# treino do modelo com as variáveis de entrada e alvo alocadas para treinamento
rf.fit(X_train, y_train)

```

Fonte: Elaboração própria.

Após o treinamento do modelo de regressão, procedemos com a geração de previsões utilizando os dados de teste. Essa etapa nos permite avaliar o desempenho do modelo ao fazer previsões sobre dados não utilizados durante o treinamento.

Em seguida, é fundamental avaliar o desempenho do modelo por meio de métricas apropriadas. Duas métricas comuns para avaliar modelos de regressão são o coeficiente de determinação (R^2) e o erro médio quadrático (MSE). Utilizamos as funções r2_score e mean_squared_error do Scikit-learn para calcular essas métricas.

O coeficiente de determinação (R^2) é uma medida que indica a proporção da variância da variável dependente que é explicada pelas variáveis independentes no modelo. Ele varia de 0 a 1, onde 1 indica um ajuste perfeito do modelo aos dados.

O erro médio quadrático (MSE) é a média dos quadrados dos erros entre as previsões do modelo e os valores reais. Ele fornece uma medida da dispersão das previsões em relação aos valores reais, onde valores menores indicam melhor ajuste do modelo.

FIGURA 8 - Trecho do programa para a métrica de avaliação de desempenho do modelo.

```
116  # > predições do modelo com dados de teste
117  y_pred = rf.predict(X_test) # alocação dos resultados preditos na variável y_pred
118
119  # > métricas de avaliação de desempenho do modelo
120  print(r2_score(y_test, y_pred))           # R2
121  print(mean_squared_error(y_test, y_pred)) # MSE
122  ----------------------------------------------------------------------------------------------------------------------------
123  Output:
124  0.2152369100923196
125  10602.490903688475
126
```

Fonte: Elaboração própria.

4.2 Implementação do Modelo de Redes Neurais

Para a implementação do modelo Redes Neurais, sua estruturação, manipulação e visualização de dados, bem como para aplicação do modelo de aprendizagem de máquina, foram utilizadas as seguintes bibliotecas: Pandas, NumPy, Seaborn, Matplotlib, Torch, Lightning, TorchMetrics, Sharp e Tqdm.

Como meio de organizar e encapsular o código relacionado à arquitetura da rede neural, seus parâmetros, métodos de treinamento e inferência, criamos duas classes, estas sendo ‘Dataset’ e ‘Model’.

A classe Dataset será responsável por fornecer os dados para treinamento e validação do modelo de rede neural, herdando funcionalidades da classe torch.utils.data.Dataset, conforme ilustra a Figura 9.

FIGURA 9 - Trecho do programa que define a classe Dataset.

```python
# Implementação do Modelo Redes Neurais

# bibliotecas
import pandas as pd # estruturação e manipulação de dados
import numpy as np # processamento de dados
import matplotlib.pyplot as plt # visualização de dados
import seaborn as sns # visualização de dados
import torch # modelo de aprendizagem de máquina baseada em tensores
import torch.nn.functional as F # funções de operações em redes neurais
from torch.utils.data import DataLoader # facilitador de carregamento de dados
import pytorch lightning as L # simplificador e padronizador no processo de treinamento de modelos
import torchmetrics # métricas de avaliação do modelo
import shap #  interpretação de modelos e visualização de dados
from tqdm import tqdm # interface para criar barras de progresso no terminal

# > classes
class Dataset(torch.utils.data.Dataset): # classe para manipulação do conjuntos de dados

    def __init__(self, df): # método de inicialização da classe
        self.df = df # atribuição do conjunto de dados ao objeto da classe

    def __len__(self): # método para retornar o comprimento do conjunto de dados
        return len(self.df) # retorna o comprimento

    def __getitem__(self, idx): # método para obter um item específico do conjunto de dados
        row = self.df.iloc[idx] # obtém a linha específica do conjunto de dados
        X = row.drop('NOTA_MT').values.astype(np.float32) # aloca as variáveis de entrada no tensor 'x'
        y = row['NOTA_MT'].astype(np.float32) # aloca a variável alvo no tensor 'y'
        return X, y  # retorna os tensores

```

Fonte: Elaboração própria.

FIGURA 10 - Trecho do programa que define a class Model e realiza a ingestão de dados.

```python
class Model(L.LightningModule): # classe para construção do modelo

    def __init__(self, n_inputs): # método de inicialização
        super().__init__() # chama o método de inicialização da classe pai
        self.fc1 = torch.nn.Linear(n_inputs, 128) # definição da primeira camada totalmente conectada
        self.fc2 = torch.nn.Linear(128, 64) # definição da segunda camada totalmente conectada
        self.fc3 = torch.nn.Linear(64, 1) # definição da terceira camada totalmente conectada
        self.r2 = torchmetrics.R2Score() # cálculo do coeficiente de determinação R²

    def forward(self, x): # método para frente (forward pass) da rede neural
        x = F.relu(self.fc1(x)) # aplicação da função de ativação ReLU à saída da primeira camada
        x = F.relu(self.fc2(x)) # aplicação da função de ativação ReLU à saída da segunda camada
        x = self.fc3(x) # retorno da saída da terceira camada
        return x # retorna a saída da rede neural

    def training_step(self, batch, batch_idx): # método para uma etapa de treinamento
        x, y = batch # aloca as variáveis de entrada e alvo do lote
        y_hat = self(x) # cálculo de previsões do modelo
        loss = F.mse_loss(y_hat, y.view(-1, 1)) # cálculo de função de perda (MSE)
        self.log('train_loss', loss, on_epoch=True, prog_bar=True, on_step=False)  # registro de perda do treinamento
        return loss # retorna a perda do treinamento
    def validation_step(self, batch, batch_idx): # método para uma etapa de validação
        x, y = batch # aloca os dados de entrada e target do lote
        y_hat = self(x) # cálculo de previsões do modelo
        loss = F.mse_loss(y_hat, y.view(-1, 1))  # cálculo de função de perda (MSE)
        self.log('val_loss', loss, on_epoch=True, prog_bar=True, on_step=False) # registro de perda da validação
        self.log('r2', self.r2(y_hat, y.view(-1, 1)), on_epoch=True, prog_bar=True, on_step=False) # registro do R²
        return loss # retorna a perda da validação
    def configure_optimizers(self): # método para configurar otimizadores
        optimizer = torch.optim.Adam(self.parameters(), lr=0.01) # definição do otimizador Adam
        scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=10,
                                                               verbose=True) # definição do agendador de taxa de aprendizado
        return {'optimizer': optimizer, 'lr_scheduler': scheduler, 'monitor': 'val_loss'} # retorna os otimizadores e o monitoramento

```

Fonte: Elaboração própria.

A Figura 10, ilustra a classe Model é responsável pela arquitetura da rede neural e definição dos métodos para treinamento, validação e otimização do modelo, com funcionalidades herdadas da classe L.LightningModule.

A seguir, declaramos a condição if __name__ == '__main__':, que verifica se o script está sendo executado como o programa principal, se a condição for satisfeita, então realizamos a ingestão dos dados. A ingestão é realizada da mesma forma que foi aplicada ao utilizarmos o modelo de Floresta Aleatória, reproduzindo o mesmo tratamento nos dados.

Definimos a variável n_inputs como o número de colunas do dataframe (df) menos 1, pois a última coluna é considerada como a variável alvo (target). Por fim, utilizamos a função torch.utils.data.random_split() para dividir o dataset em conjuntos de treinamento e teste onde, o primeiro argumento da função, o objeto Dataset(df), contém todos os dados, e o segundo argumento indica a proporção de dados que devem ser atribuídos ao conjunto de treinamento e ao conjunto de teste (75% destinados ao treinamento e 25%, para o teste.

Criamos dataloaders de treino (train_loader) para carregar os dados de treinamento (train_dataset) e de teste (test_loader) para carregar os dados de teste (test_dataset). Definimos uma variável que possui o tipo de dispositivo utilizado para o processamento do modelo e criamos uma instância do modelo '*Model*', passando os parâmetros necessários.

FIGURA 11 - Trecho do programa que faz a ingestão de dados e o treinamento do modelo.

```
# > ingestão de dados
if __name__ == '__main__':
    df = pd.read_csv('microdados_enem_2022.csv', sep=';', encoding='latin-1',
                     usecols=['TP_FAIXA_ETARIA', 'TP_SEXO', 'TP_ESTADO_CIVIL', 'TP_COR_RACA', 'TP_ST_CONCLUSAO',
                              'CO_UF_ESC', 'TP_DEPENDENCIA_ADM_ESC', 'NU_NOTA_MT', 'Q001', 'Q002', 'Q003', 'Q004',
                              'Q005', 'Q006', 'Q010', 'Q024', 'Q025'])  # criação de dataframe

n_inputs = df.shape[1]-1 # definição do número de inputs

# > divisão dos dados para treino e teste
train_dataset, test_dataset = torch.utils.data.random_split(Dataset(df), [0.75, 0.25]) # variáveis de treino, variáveis de teste

# > criação de dataloaders
# carrega os dados de treino em lotes
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=False, num_workers=7, persistent_workers=True)
# carrega os dados de teste em lotes
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False, num_workers=7, persistent_workers=True)

device = 'cpu' # dispositivo de processamento
# > instanciamento, treinador e treinamento de modelo
model = Model(n_inputs=n_inputs) # criação da instância do modelo rn e ajuste de hiperparâmetros
trainer = L.Trainer(max_epochs=100, enable_progress_bar=True, enable_model_summary=True) # criação de treinador
trainer.fit(model, train_dataloaders=train_loader, val_dataloaders=test_loader) # treino do modelo
```

Fonte: Elaboração própria.

Assim como apresentado no modelo de Floresta Aleatória, aplicaremos as métricas de R^2 e MSE para que se possa verificar o comportamento do ajuste dos dados dos dados no modelo e sua precisão na predição de resultados:

FIGURA 12 - Trecho do programa para métrica de avaliação de desempenho do modelo.

```
# > métricas de avaliação de desempenho do modelo
print('R2:', trainer.logged_metrics['r2'].item()) # R2
print('MSE:', trainer.logged_metrics['val_loss'].item()) # MSE
----------------------------------------------------------------------------------------------------------------------
Output:
0.3154
9160.1631
```

Fonte: Elaboração própria.

Consideramos que quanto mais próximo de 1 for o valor de R^2, melhor o modelo se ajusta aos dados e que, quanto menor o valor do MSE, melhor o modelo está em fazer previsões precisas. Vejamos o comparativo entre os modelos, apresentado na Tabela 1:

TABELA 1 - Comparação das métricas a partir dos modelos de aprendizagem de máquina.

Métrica	Algoritmo	Resultado Final
R2	Floresta Aleatória	0.2152369100923196
	Redes Neurais	0.3154
MSE	Floresta Aleatória	10602.490903688475
	Redes Neurais	9160.1631

Fonte: Elaboração própria.

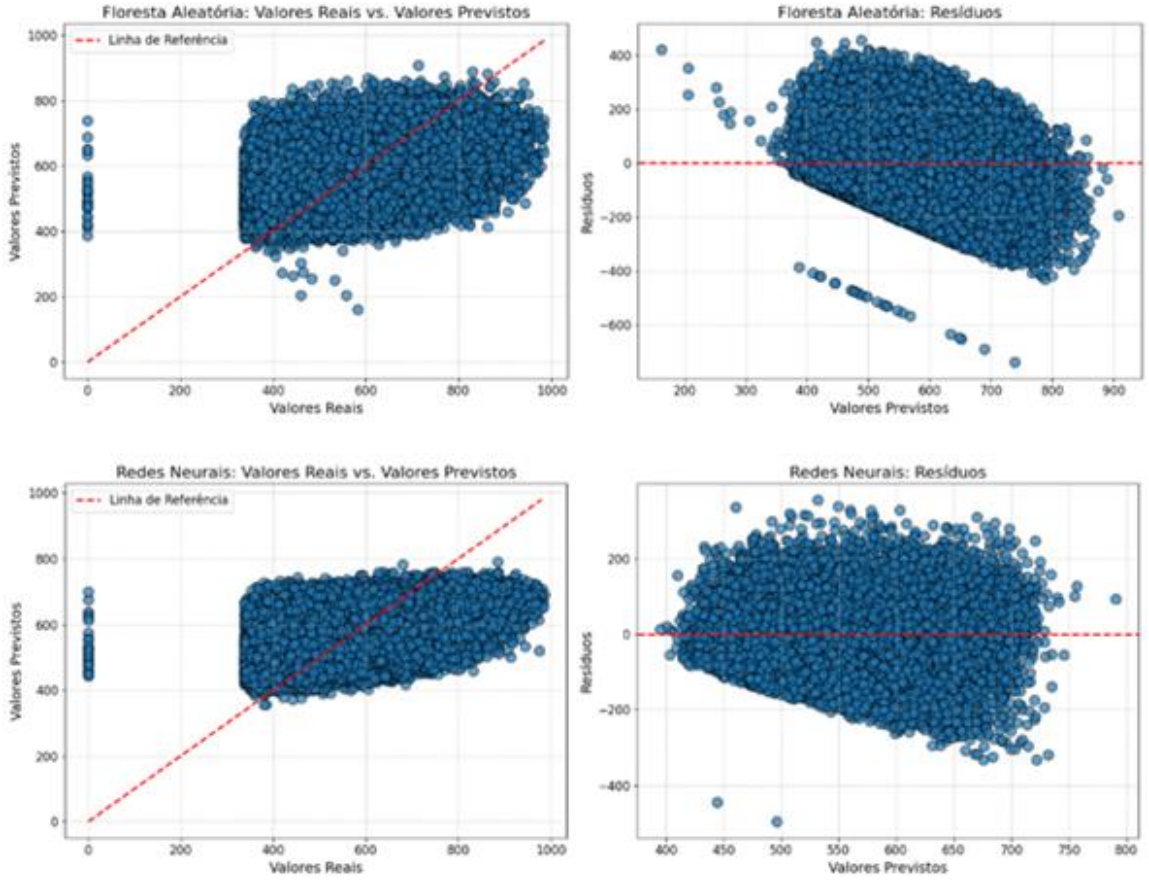
Ao compararmos as informações, observa-se que o modelo de Floresta Aleatória resultou em um R2 de aproximadamente 0.2152, enquanto o modelo de Redes Neurais apresentou um resultado de 0.3154. Isso significa que apenas 21.5% da variabilidade nas notas de matemática dos candidatos pode ser explicada pelas variáveis independentes utilizadas no modelo de Floresta Aleatória, indicando que este possui uma capacidade limitada de explicação. Já o modelo de Rede Neurais conseguiu explicar 31.54% dos dados, demonstrando uma performance melhor ao ajuste dos dados treinados.

Em relação ao MSE, a Floresta Aleatória apresentou um valor de aproximadamente 10602, indicando um valor significativamente alto. Significa que as diferenças entre as notas reais e as notas previstas pelo modelo são grandes e o modelo não está prevendo com precisão as notas de matemática, resultando em estimativas que estão, em média, bastante distantes dos valores reais. Em relação ao resultado de precisão do modelo anterior, o modelo de Redes Neurais foi superior, embora o valor de MSE de 9160, sugere uma precisão do modelo.

Na Figura 13, pode-se analisar o modelo de Floresta Aleatória, com uma dispersão dos pontos ao redor da linha de referência, indicando que há uma quantidade significativa de erros de previsão, ou seja, o modelo tende a subestimar ou superestimar os valores reais. Além

disso, existe uma concentração de pontos em um intervalo específico dos valores reais (entre 400 e 800), sugerindo que o modelo pode ter dificuldade em prever corretamente fora deste intervalo. Quanto ao gráfico de resíduos, a distribuição varia amplamente, com alguns pontos distantes, apresentando imprecisões e erros no modelo.

FIGURA 13 - Gráficos baseados nos modelos de Floresta Aleatória e Redes Neurais.



Fonte: Elaboração própria.

No modelo de Redes Neurais, em relação ao modelo anterior, a dispersão dos pontos ao redor da linha de referência é menor, indicando que as previsões das Redes Neurais são um pouco mais precisas. Contudo, ainda há uma variabilidade significativa nos valores previstos, especialmente para os valores reais mais baixos, indicando que o modelo não prevê bem esses valores. Quanto ao gráfico de resíduos, a distribuição é mais uniforme comparada ao modelo anterior, estando mais concentrados em torno da linha de referência, indicando menos erros.

Dessa forma, entende-se que apesar de ainda existirem erros significativos, especialmente para valores reais mais baixos, o modelo de Redes Neurais apresentou melhor performance de precisão e consistência das previsões do que o modelo de Floresta Aleatória.

6. CONSIDERAÇÕES FINAIS

O estudo teve como objetivo aplicar as técnicas de aprendizado de máquina Floresta Aleatória e Redes Neurais em microdados abertos do ENEM referentes ao ano de 2022, onde suas performances de resultado pudessem ser analisadas a fim de determinar qual dos modelos obteve melhor resultado na interpretabilidade da correlação entre as variáveis trabalhadas e em previsões geradas após seus treinamentos.

Conclui-se que, embora ambos os modelos tenham apresentado capacidade limitada de explicação das notas de matemática dos candidatos e uma precisão de previsões não ideal, o modelo de Redes Neurais mostrou-se mais eficaz na utilização das variáveis independentes para interpetabilidade dos dados e mais adequado para previsões de notas, impactando no resultado das notas de matemática e suas tecnologias pelos candidatos do ENEM de 2022.

REFERÊNCIAS

BREIMAN, L. **Random Forests**. *Machine Learning*, v. 45, p. 5-32, 2001.

_____. **Bagging Predictors**. *Machine Learning*, v. 24, p. 123-140, 1996.

BUENO, M.L.; BOENTE, A.N.P. **Determinantes de Desempenho no ENEM:** Uma abordagem em Machine Learning. Monografia (Graduação). Universidade Veiga de Almeida, Bacharel em Engenharia de Computação, Rio de Janeiro, RJ, 2024.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge: MIT Press, 2017.

INSTITUTO NACIONAL DE ESTUDOS E PESQUISAS EDUCACIONAIS ANÍSIO TEIXEIRA (INEP). **Microdados do ENEM**. Disponível em: <https://www.gov.br/inep/pt-br/acesso-a-informacao/dados-abertos/microdados/enem\>. Acesso em: 19 Março 2024.

LIAW, A.; WIENER, M. **Classification and Regression by randomForest**. *R News*, v. 2, n. 3, p. 18-22, 2002.

MCCULLOCH, W. S.; PITTS, W. **A logical calculus of the ideas immanent in nervous activity**. Bulletin of Mathematical Biophysics, v. 5, n. 4, p. 115-133, 1943.

NAPOLEÃO, B.M. **Árvores Decisórias**. Disponível em: https://ferramentasdaqualidade.org/arvores-decisorias/. Acesso em: 25 Abril 2024.

ROSENBLATT, F. **The Perceptron**: a probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386-408, 1958.

TAULLI, T. **Introdução à Inteligência Artificial:** Uma abordagem não técnica. São Paulo: Novatec, 2020.