

APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS NA PREDIÇÃO DE ACIDENTE VASCULAR CEREBRAL

Rafael José Camargo Bekhauser¹

¹Graduando Bacharelado em Ciência da Computação - IFC Rio do Sul

rafaelcamargo.inf@gmail.com

Abstract. *Artificial intelligence is the branch of computer science that seeks to understand how the mind works, allowing computers to reason in a similar way. One of its ramifications is related to a set of techniques known as Machine Learning, which allow the construction of models that act with great reliability in the prediction of data, one of its techniques the artificial neural networks enable the classification of sets. Therefore, through the application of such techniques we developed an algorithm capable of acting in the prediction of stroke, and compare its accuracy and performance.*

Key-words: *Artificial Neural Networks; Stroke Prediction; Artificial Intelligence.*

Resumo. *A inteligência artificial é o ramo da ciência da computação que busca entender como a mente funciona, permitindo que os computadores raciocinam de maneira semelhante. Uma de suas ramificações se diz respeito a um conjunto de técnicas conhecida como Machine Learning, tais quais possibilitam a construção de modelos que atuam com grande confiabilidade na predição de dados, uma de suas técnicas as redes neurais artificiais possibilitam a classificação de conjuntos. Logo, por meio da aplicação de tais técnicas desenvolvemos um algoritmo capaz de atuar na predição do Acidente Vascular Cerebral, e compararmos sua acurácia e performance.*

Palavras-chave: *Redes Neurais Artificiais; Predição de AVC; Inteligência Artificial.*

1. Introdução

O sistema nervoso é uma complexa rede de comunicação de um organismo em que diferentes órgãos desempenham diferentes funções, como capturar informações e estímulos do ambiente, interpretar esses estímulos e arquivar todos esses dados. Por sua vez, o organismo responde, seja por meio de movimento, sentimento ou abstração, logo, alterações súbitas no processo de funcionamento do sistema nervoso, podem acarretar graves problemas à saúde (FRIEDRICH, 2019).

Dores de cabeça repentinas, dificuldade para falar e dormência nas extremidades sem motivo aparente podem ser caracterizados como sintomas de uma enfermidade que assola o cérebro de milhões de indivíduos e os limita pelo resto de suas vidas. O Acidente Vascular Cerebral (AVC), considerada uma das doenças mais mortais, é a

principal causa de incapacidade em pessoas com mais de 50 anos (ABRAMCZUK; VILLELA, 2009).

De acordo com a Academia Brasileira de Neurologia, a principal causa de AVCs é uma obstrução no fornecimento de vasos sanguíneos ao cérebro. Isso pode ser causado por aterosclerose ou um coágulo de sangue no cérebro. Outras causas incluem artérias estreitadas ou bloqueadas, doenças cardíacas e pressão alta. Outros fatores que aumentam as chances de alguém ter um derrame incluem idade, sexo e raça. Pessoas com mais de 55 anos são mais propensas a sofrer um AVC do que as mais jovens.

Para combater este cenário, portanto, é fundamental atuar na prevenção dos fatores de risco desde cedo. Embora a doença ainda seja prevalente entre os idosos, a população está desenvolvendo fatores que levam a doenças cardiovasculares cada vez mais precocemente. Portanto se faz de grande importância a existência de um diagnóstico rápido e preciso, visando analisar os sintomas e histórico do paciente de forma precoce a fim de prevenir danos gravíssimos à vida do paciente, logo desta forma por meio da aplicação de técnicas de inteligência artificial, podemos atuar na predição do AVC.

2. Machine Learning

Machine Learning ou aprendizado de máquina é um ramo da inteligência artificial que estuda a construção de algoritmos computacionais a partir do aprendizado de dados. O principal objetivo dos modelos de machine learning é criar um sistema de computador que aprenda a partir de um banco de dados predefinido e, finalmente, produza um modelo de previsão, classificação ou reconhecimento. Logo desta forma, os métodos de ML se enquadram perfeitamente como uma ferramenta aplicada ao auxílio do diagnóstico médico, no que tange o processo de identificação da enfermidade, possibilitando uma predição do acometimento da doença ao indivíduo.

2.1 Redes Neurais Artificiais

Redes neurais artificiais (RNAs) são uma classe de algoritmos de aprendizado de máquina derivados dos princípios do sistema nervoso humano. As redes neurais são usadas em muitas aplicações, como programação de computadores e estratégia militar.

As RNAs são compostas de unidades de processamento interconectadas – ou neurônios – que atuam como os cérebros dos programas de computador. Cada unidade de processamento tem sua própria função e memória. Dessa forma, as RNAs imitam o cérebro humano.

Dentro do amplo universo de redes neurais artificiais, existem diversos tipos de modelagem e métodos, sendo um deles o modelo Multilayer Perceptron.

2.2 Multilayer Perceptron

O método Multilayer Perceptron é uma rede neural artificial que consiste em pelo menos 3 camadas de nós: uma camada de entrada, uma camada oculta e uma camada de saída. A camada oculta possui este nome pois não é possível prever a saída desejada nas camadas intermediárias.

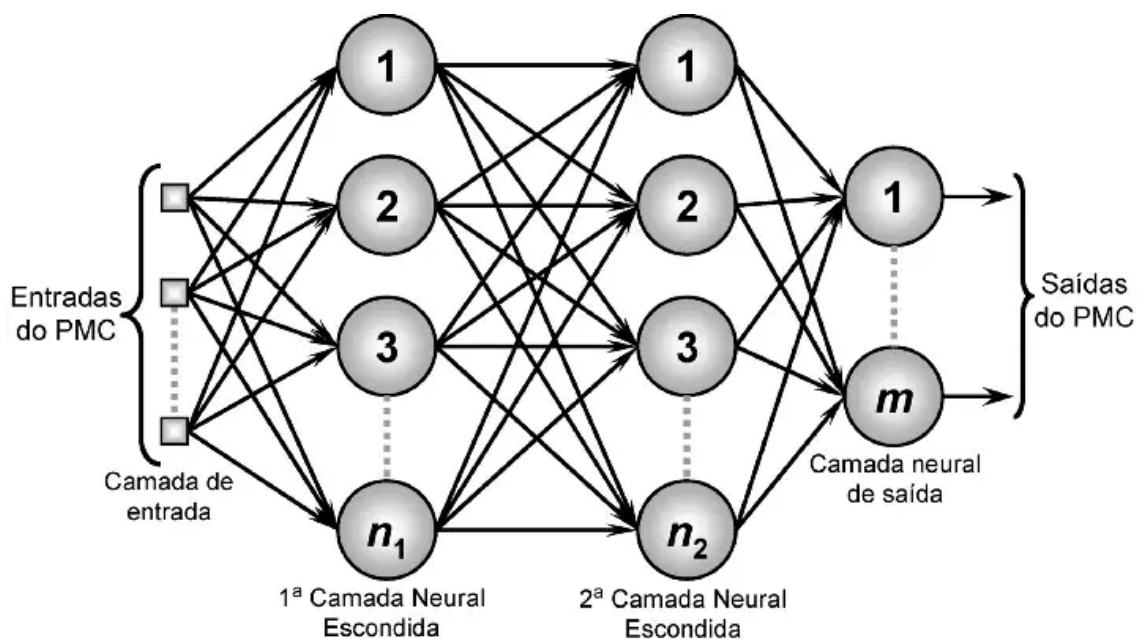


Figura 1 – Diagrama da rede multilayer perceptron.

3. Aplicação

A construção da aplicação consiste em realizar uma modelagem para a classificação se um paciente possui ou não AVC.

3.1 Base de dados

Para a construção do modelagem da RNA MLP, se faz necessário uma base de dados para que se possa realizar o devido treinamento da rede a fim de torná-la capaz de prever o diagnóstico de um AVC, por meio de variáveis de entrada. Para isto utilizamos da base de dados “Stroke Prediction Dataset” obtida através da plataforma on-line Kaggle (Criada em 2010, Kaggle é uma plataforma que possibilita a realização de uma ampla quantidade de atividades que envolvem as áreas de Data Science e Machine Learning).

3.2 Análise exploratória dos dados

Análise exploratória de dados é o processo de investigar e explorar um conjunto de dados para obter informações sobre ele. Este é um passo importante na análise de dados porque ajuda a entender melhor os dados e identificar padrões, tendências e relação entre variáveis. Usado para entender melhor os conjuntos de dados e preparar dados para análise.

O dataset de Stroke Prediction contém informações de dados de pacientes que tiveram ou não AVC, logo possui um total de 11 colunas contendo dados do histórico e índices do paciente e 1 coluna contendo o fator resultante, se possui ou não AVC, somando um total de 5110 registros.

Os dados fornecidos pelo dataset são:

1. id: identificador único;
2. gender: "Masculino", "Feminino" ou "Outro";
3. age: idade do paciente;
4. hypertension: 0 se o paciente não tem hipertensão, 1 se o paciente tem hipertensão
5. heart_disease: 0 se o paciente não tem nenhuma doença cardíaca, 1 se o paciente tem uma doença cardíaca;
6. ever_married: se o paciente já se casou (“Sim” ou “Não”);

7. work_type: tipo de trabalho do paciente (“Nunca trabalhou”, “Privado”, “Autônomo”, “Governo”, “Criança”);
8. Residência: "Rural" ou "Urbana";
9. avg_glucose_level: nível médio de glicose no sangue;
10. bmi: índice de massa corporal;
11. smoking_status: status de fumante do paciente (nunca fumou, ex-fumante, fumante atual ou desconhecido);
12. stroke: 1 se o paciente teve um acidente vascular cerebral ou 0 se não.

"Desconhecido" em smoking_status significa que a informação não está disponível para este paciente.

Analisando os dados apresentados, podemos notar a existência de variáveis categóricas (são variáveis categóricas, aquelas que não têm uma escala de medida, ou seja, são qualitativas) e numéricas.

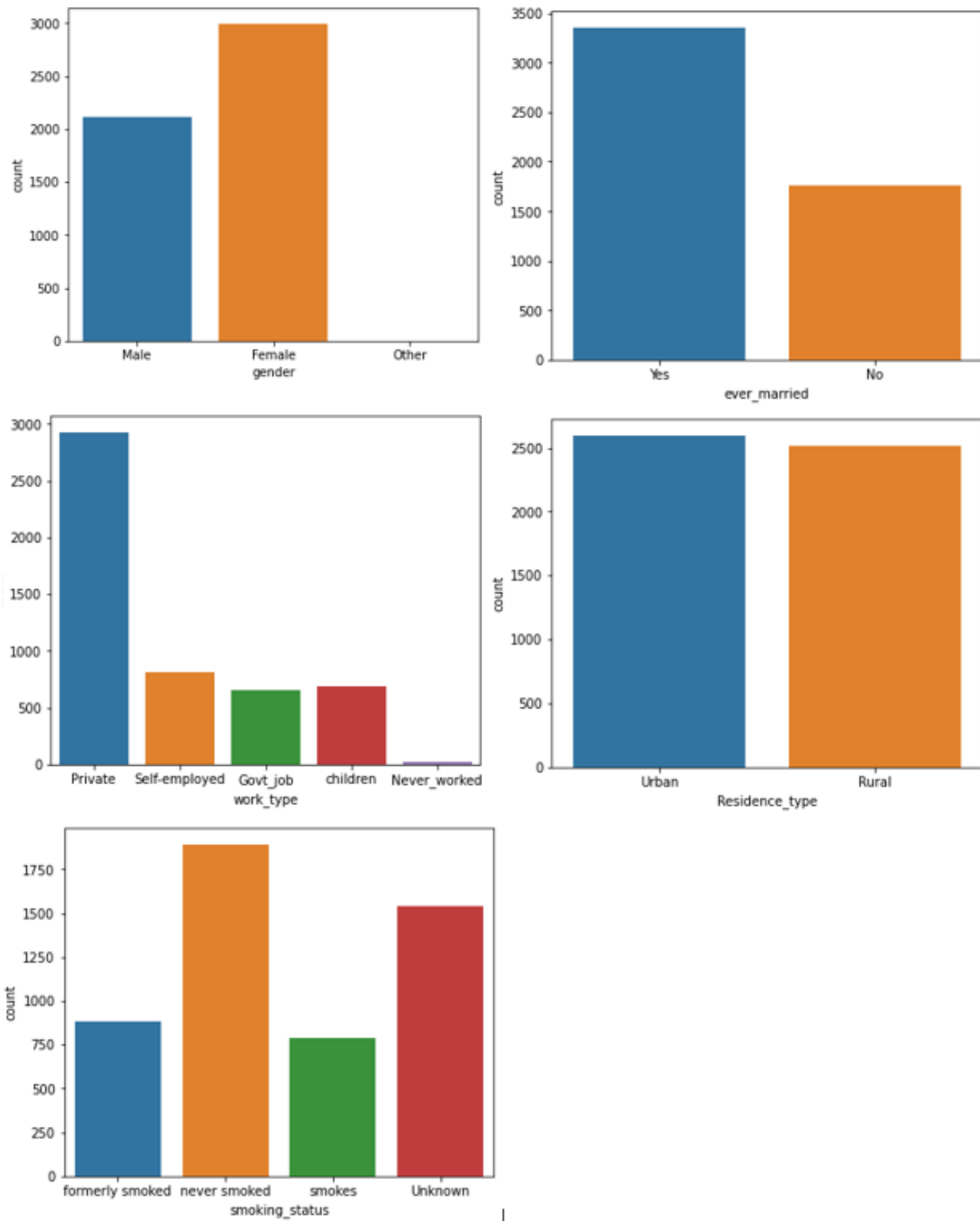


Figura 2 – Distribuição de variáveis categóricas.

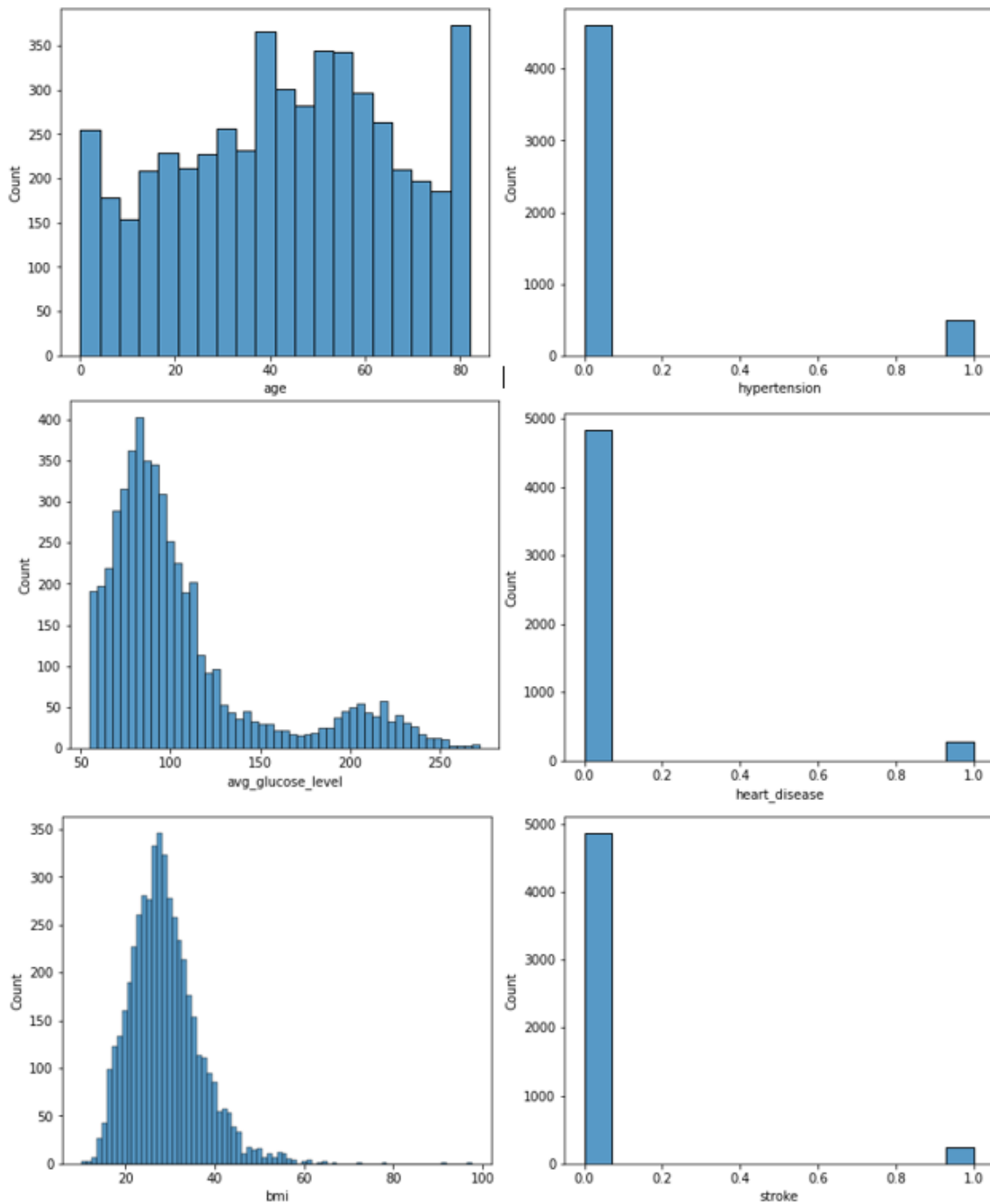


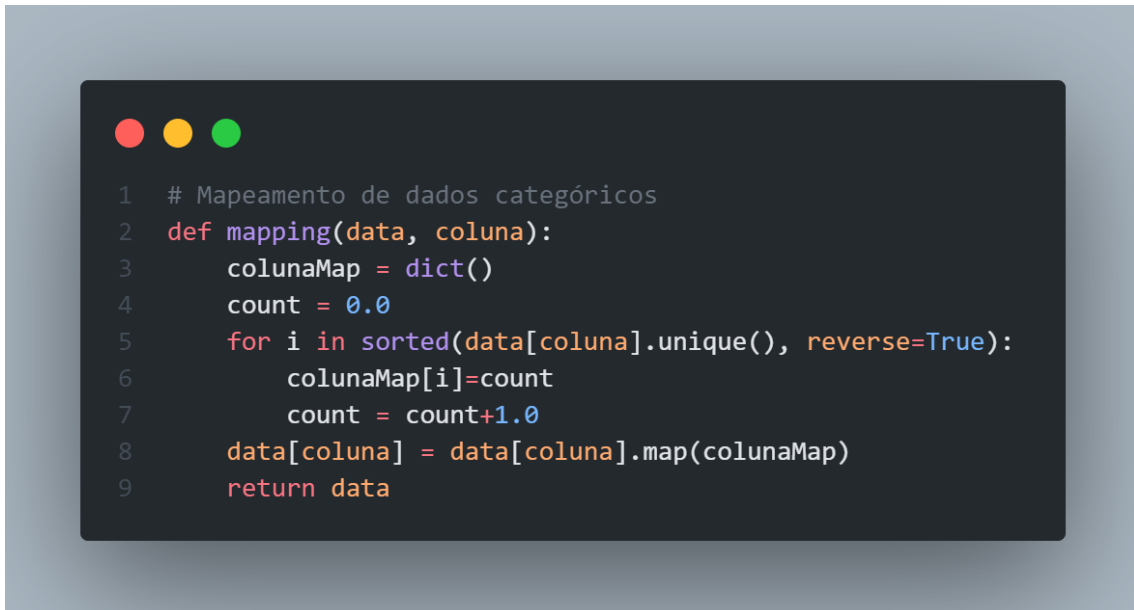
Figura 3 – Distribuição de variáveis numéricas.

3.3 Pré-processamento do dataset

O pré-processamento de conjuntos de dados é uma etapa importante na análise de dados. É responsável por preparar o conjunto de dados para processamento e análise para que as informações necessárias possam ser extraídas. O pré-processamento pode

ser usado para tarefas como lidar com dados ausentes, remover ruído, normalizar dados, converter variáveis categóricas em variáveis numéricas, etc.

Para que se possa utilizar as variáveis categóricas na modelagem se faz necessário fazer o mapeamento de suas classes bem como normalizar os dados a fim de estabilizar as informações. Logo, podemos definir duas funções para realizar tal operação.



```
1 # Mapeamento de dados categóricos
2 def mapping(data, coluna):
3     colunaMap = dict()
4     count = 0.0
5     for i in sorted(data[coluna].unique(), reverse=True):
6         colunaMap[i]=count
7         count = count+1.0
8     data[coluna] = data[coluna].map(colunaMap)
9     return data
```

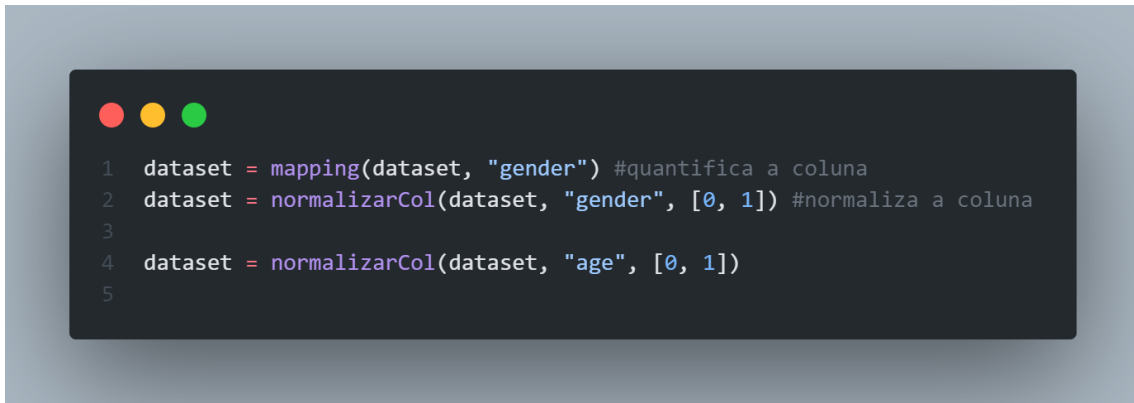
Figura 4 – Função de mapeamento.



```
1 # Normalização de dados numéricos
2 def normalizarCol(data, coluna, rangeN=[0,1]):
3     data[coluna] = (((data[coluna] - data[coluna].min()) / (data[coluna].max() - data[coluna].min())) * (rangeN[1] - rangeN[0])) - rangeN[0]
4     return data.convert_dtypes(infer_objects=False, convert_integer=False, convert_floating= True)
5
```

Figura 5 – Função de normalização.

Aplicamos o mapeamento em todas as variáveis categóricas e normalizamos todas as demais.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains five lines of Python code. Line 1: `dataset = mapping(dataset, "gender")` with a comment `#quantifica a coluna`. Line 2: `dataset = normalizarCol(dataset, "gender", [0, 1])` with a comment `#normaliza a coluna`. Line 3 is empty. Line 4: `dataset = normalizarCol(dataset, "age", [0, 1])`. Line 5 is empty.

```
1 dataset = mapping(dataset, "gender") #quantifica a coluna
2 dataset = normalizarCol(dataset, "gender", [0, 1]) #normaliza a coluna
3
4 dataset = normalizarCol(dataset, "age", [0, 1])
5
```

Figura 6 – Exemplo de mapeamento e normalização de dados.

Analisando as variáveis apresentadas, notamos a presença de um identificador único (id), tal qual se torna irrelevante na modelagem, visto que não é uma informação pertinente à predição, portanto foi necessário a remoção da mesma.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains two lines of Python code. Line 1: `# drop coluna id`. Line 2: `dataset.drop(['id'], axis=1, inplace=True)`.

```
1 # drop coluna id
2 dataset.drop(['id'], axis=1, inplace=True)
```

Figura 7 – Remoção da coluna id.

Verificando os valores nulos contidos no dataset podemos constatar que a coluna de BMI contém 4909 valores não nulos de 5110, ou seja, existem 201 dados faltantes. Como forma de preencher tais dados, inserimos no dataset nas linhas faltantes a média dos dados do BMI, conforme imagem abaixo.



```
1 dataset = dataset.fillna(dataset['bmi'].mean())
```

Figura 8 – Preenchimento dos dados nulos da coluna BMI.

3.4 Divisão dos conjuntos

A divisão do conjunto de dados para treinamento e teste é uma etapa necessária no processo de construção de um modelo de RNA. Por meio desta, é possível avaliar a performance e acurácia por meio do conjunto de testes, e realizar o treinamento da rede por meio do conjunto de treino. Para a construção deste modelo, utilizamos a proporção de 80% dos dados para treino e 20% para teste.



```
1 X, y = dataset.values[:, :-1], dataset.values[:, -1]
2 X= X.astype('float32')
3 y = LabelEncoder().fit_transform(y)
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
5 print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
6 n_features = X_train.shape[1]
```

Figura 9 – Divisão dos conjuntos de teste e treino.

3.5 Definição da rede

A construção da rede neural multilayer perceptron, se dá por meio da definição das camadas ocultas, quantidade de neurônios e função de ativação. Logo, podemos definir:

```

1 model = tf.keras.models.Sequential()
2 model.add(tf.keras.layers.Dense(n_features, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
3 model.add(tf.keras.layers.Dense(6, activation='relu', kernel_initializer='he_normal'))
4 model.add(tf.keras.layers.Dense(14, activation='relu', kernel_initializer='he_normal'))
5 model.add(tf.keras.layers.Dense(24, activation='relu', kernel_initializer='he_normal'))
6 model.add(tf.keras.layers.Dense(6, activation='relu', kernel_initializer='he_normal'))
7 model.add(tf.keras.layers.Dense(2, activation='softmax'))

```

Figura 10 – Modelagem da rede.

3.6 Avaliação da rede

A acurácia e a perda são duas métricas usadas para avaliar modelos de aprendizado de máquina. A acurácia refere-se à capacidade de um modelo de classificação de prever corretamente as classes de um conjunto de dados, ou seja, apresenta quão um modelo é preciso para classificar um conjunto de dados. Uma função de perda, também conhecida como função de custo, leva em consideração as probabilidades ou incertezas de uma previsão com base em quanto a previsão varia do valor real. Isso nos dá uma visão mais detalhada do desempenho do modelo.

Como resultado apresentado pelo modelo obtivemos uma perda média de 0.182 e uma acurácia média de 0.955.

```

loss, acc = model.evaluate(X_test, y_test)
✓ 0.1s

32/32 [=====] - 0s 1ms/step - loss: 0.1822 - accuracy: 0.9550

print('Perda: %.3f, \nAcuracia: %.3f' % (loss, acc))
✓ 0.4s

Perda: 0.182,
Acuracia: 0.955

```

Figura 11 – Resultados de avaliação.

Outro parâmetro de avaliação de uma rede de classificação se diz na aplicação de uma matriz de confusão, tal qual é uma tabela que permite a visualização do desempenho de um algoritmo de classificação.

Dado que os dados separados para teste em um modelo de aprendizado de máquina são também dados passados — sobre os quais sabemos o valor real da variável resposta —, é possível comparar a coluna de predições com a coluna de valores reais. Ou seja, a matriz de confusão apresenta uma tabela comparativa dos valores que um algoritmo trouxe como predição em relação aos valores reais ocorridos.

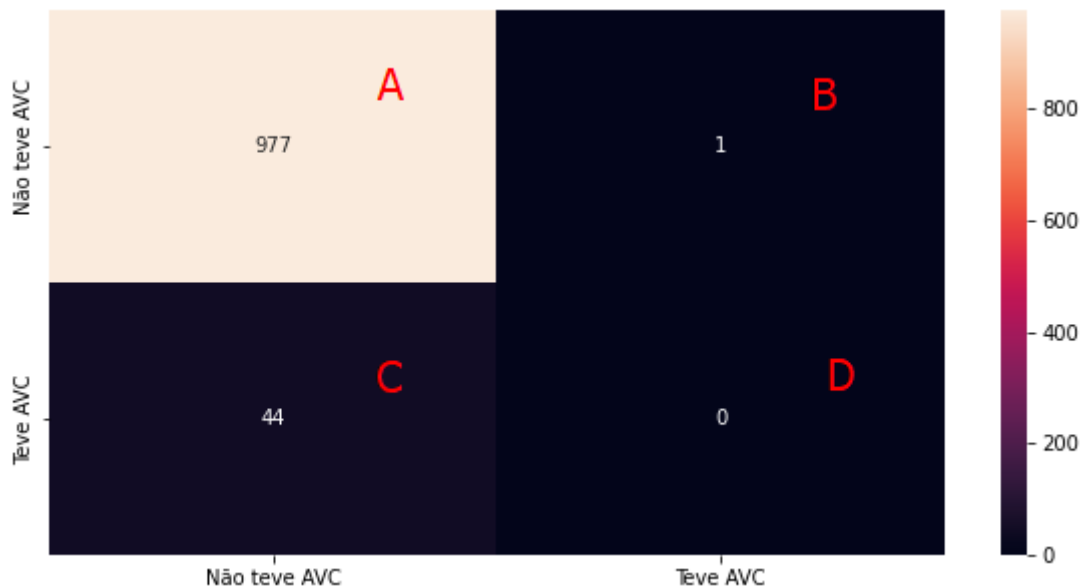


Figura 12 – Matriz de confusão.

Podemos definir os 4 quadrantes da matriz de confusão como sendo A = Positivo verdadeiro, B = Positivo falso, C = Negativo falso e D = Negativo verdadeiro. Analisando os resultados demonstrados pela matriz, podemos verificar que o modelo de predição ficou mais ajustado para prever se uma pessoa não tem AVC, apresentando uma maior concentração de valores.

4 Conclusões

É relevante ressaltar a importância da inteligência artificial em nosso cotidiano. Esta ferramenta tecnológica está se desenvolvendo muito rapidamente para ajudar vários setores industriais e afetar a vida das pessoas de diferentes maneiras, não apenas financeiramente, mas também em termos de saúde.

Por meio do uso da rede neural multilayer perceptron para a modelagem da predição do acidente vascular cerebral, obtivemos bons resultados no conjunto de testes, demonstrando que a rede consegue se ajustar e generalizar, outrossim apresentou um baixo tempo nos testes realizados. Resultando em uma acurácia de 95.5%, satisfatório para tal predição.

Referências

ABRAMCZUK, Beatriz; VILLELA, Edlaine. **A luta contra o AVC no Brasil.** ComCiência, n. 109, p. 0-0, 2009.

DE OLIVEIRA, Roberto de Magalhães Carneiro; DE ANDRADE, Luiz Augusto Franco. **Acidente vascular cerebral.** Rev Bras Hipertens, p. 8-3, 2001.

FRIEDRICH, Maurício. **O que são doenças neurológicas?** 2019.

MAINALI, Shraddha; DARSIE, Marin E.; SMETANA, Keaton S. **Machine learning in action: Stroke diagnosis and outcome prediction.** Frontiers in neurology, p. 2153, 2021.

SOUZA, Cristiano Gonçalves; TOSTES, Lelia de Mello. **Utilização de redes neurais artificiais no auxílio ao diagnóstico de doenças reumatológicas.** 2004.

Apêndice A - Código fonte do algoritmo de predição

```
# %% [markdown]
# # Predição de Acidente Vascular Cerebral (AVC) com RNA MLP

# %% [markdown]
# A base de dados utilizada neste projeto foi obtida no
Kaggle, e pode ser encontrada
\[aqui\]\(https://www.kaggle.com/fedesoriano/stroke-prediction-dataset\).

# %% [markdown]
# ### Importando as bibliotecas necessárias

# %%
import tensorflow as tf
import numpy as np
import pandas as pd
import seaborn as sns
import pickle
import matplotlib
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
import datetime

# %% [markdown]
# ### Carregando os dados

# %%
dataset =
pd.read_csv('data/healthcare-dataset-stroke-data.csv')
dataset.head(5)

# %%
dataset.info()

# %%
"""
    Dados de entrada:
    gender:          sexo do paciente (masculino, feminino,
    outro)
```

```

    Age:                idade do paciente (em anos)
    hypertension:       se o paciente tem hipertensão (0, 1)
    heart_disease:      se o paciente tem doença cardíaca (0,
1)
    ever_married:       se o paciente já se casou (sim, não)
    work_type:          tipo de trabalho do paciente (nunca
trabalhou, privado, autonomo, governo, criança)
    Residence_type:     tipo de residência do paciente (urbana,
rural)
    avg_glucose_level:  nível médio de glicose no sangue do
paciente (em mg/dL)
    bmi:                índice de massa corporal do paciente
(em kg/m^2)
    smoking_status:     status de fumante do paciente (nunca
fumou, ex-fumante, fumante atual ou desconhecido)
    stroke:            se o paciente teve um AVC (0, 1)
"""

# %%
# drop coluna id
dataset.drop(['id'], axis=1, inplace=True)

# %% [markdown]
# ### Análise Exploratória de Dados (EDA)

# %%
import matplotlib.pyplot as plt

# %% [markdown]
# #### Colunas categóricas e numéricas

# %%
colCategoricas = dataset[['gender', 'ever_married',
'work_type', 'Residence_type', 'smoking_status']]

for col in colCategoricas:
    plt.figure(figsize=(6,5))
    sns.countplot(x=col, data=dataset)
    plt.show()

# %%

```



```

colNumericas = dataset[['age', 'hypertension',
'heart_disease', 'avg_glucose_level', 'bmi', 'stroke']]

for col in colNumericas:
    plt.figure(figsize=(6,5))
    sns.histplot(x=col, data=dataset)
    plt.show()

# %% [markdown]
# ### Pré-processamento dos dados

# %%
# Mapeamento de dados categóricos
def mapping(data, coluna):
    colunaMap = dict()
    count = 0.0
    for i in sorted(data[coluna].unique(), reverse=True):
        colunaMap[i]=count
        count = count+1.0
    data[coluna] = data[coluna].map(colunaMap)
    return data

# Normalização de dados numéricos
def normalizarCol(data, coluna, rangeN=[0,1]):
    data[coluna] = (((data[coluna] - data[coluna].min()) /
(data[coluna].max() - data[coluna].min())) * (rangeN[1] -
rangeN[0])) -rangeN[0]
    return data.convert_dtypes(infer_objects=False,
convert_integer=False, convert_floating= True)

# %%
print(dataset.gender.unique())

dataset['gender'] = dataset['gender'].apply(lambda x: 1 if x
== 'Male' else 0)

print(dataset.gender.unique())

# %%
dataset.age.info()
dataset = normalizarCol(dataset, "age", [0, 1])

```

```
# %%
print(dataset.ever_married.unique())

dataset = mapping(dataset, "ever_married")
dataset = normalizarCol(dataset, "ever_married", [0, 1])

print(dataset.ever_married.unique())

# %%
print(dataset.work_type.unique())

dataset = mapping(dataset, "work_type")
dataset = normalizarCol(dataset, "work_type", [0, 1])

print(dataset.work_type.unique())

# %%
print(dataset.Residence_type.unique())

dataset = mapping(dataset, "Residence_type")
dataset = normalizarCol(dataset, "Residence_type", [0, 1])

print(dataset.Residence_type.unique())

# %%
dataset.avg_glucose_level.info()
dataset = normalizarCol(dataset, "avg_glucose_level", [0, 1])

# %%
dataset.bmi.info()
dataset = dataset.fillna(dataset['bmi'].mean())
dataset = mapping(dataset, "bmi")
dataset = normalizarCol(dataset, "bmi", [0, 1])

# %%
print(dataset.smoking_status.unique())

dataset = mapping(dataset, "smoking_status")
dataset = normalizarCol(dataset, "smoking_status", [0, 1])

print(dataset.smoking_status.unique())
```

```

# %%
dataset.stroke.info()
dataset = normalizarCol(dataset, "stroke", [0, 1])

# %% [markdown]
# ### Divisão dos dados em treino e teste

# %%
X, y = dataset.values[:, :-1], dataset.values[:, -1]
X= X.astype('float32')
y = LabelEncoder().fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20)
print(X_train.shape, X_test.shape, y_train.shape,
y_test.shape)
n_features = X_train.shape[1] #feature selection

# %% [markdown]
# ### Construção do modelo de predição

# %%
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(n_features, activation='relu',
kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(tf.keras.layers.Dense(6, activation='relu',
kernel_initializer='he_normal'))
model.add(tf.keras.layers.Dense(14, activation='relu',
kernel_initializer='he_normal'))
model.add(tf.keras.layers.Dense(24, activation='relu',
kernel_initializer='he_normal'))
model.add(tf.keras.layers.Dense(14, activation='relu',
kernel_initializer='he_normal'))
model.add(tf.keras.layers.Dense(2, activation='softmax'))

# %%
model.summary()

# %%

model.compile(optimizer='adam',

```

```

loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=64, batch_size=64,
validation_data=(X_test, y_test))

# %%
loss, acc = model.evaluate(X_test, y_test)

# %%
print('Perda: %.3f, \nAcuracia: %.3f' % (loss, acc))

# %%
from sklearn.metrics import confusion_matrix

predictions = model.predict(X_test)

plt.figure(figsize=(10,5))
cm = confusion_matrix(y_test, predictions.argmax(axis=1))
confusion_matrix = pd.DataFrame(cm, index = ['Não teve AVC',
'Teve AVC'], columns = ['Não teve AVC', 'Teve AVC'])
sns.heatmap(confusion_matrix, annot=True, fmt='g')

# %% [markdown]
# A = Positivo verdadeiro
# B = Positivo falso
# C = Negativo falso
# D = Negativo verdadeiro
#

# %%

```