Desempenho do método de Regressão Logística na classificação de exames por espectrometria de massa quanto à presença de câncer do ovário – 2021/1*

*Nota: Projeto da discplina Aprendizado de Máquina do 1o. Semestre de 2021

Aluno: Carlos Fonseca
Programa de Pós Graduação em Eng. Elétrica
PPGEE/UFAM
Manaus/AM, Brasil
henrique_fonseca@hotmail.com

Aluno: Willian Guerreiro
Programa de Pós Graduação em Eng. Elétrica
PPGEE/UFAM
Manaus/AM, Brasil
wguerreiro31@gmail.com

***Resumo*—Este trabalho tem como objetivo implementar o método de regressão logística, uma rede neural simples de 01 única camada, para classificar os resultados dos exames de espectometria de massa em 02 classes de diagnóstico: com ou sem câncer de ovário. Foi utilizada uma base de dados "ovarianInputs"com os dados de 216 pacientes examinados com intensidades de íons correspondentes a 100 valores específicos de carga-massa, assim como a base de dados "ovarianTargets"com os resultados do diagnóstico para fins de treinamento da rede neural (aprendizado supervisionado). Foi utilizado o método k-fold em 5 pastas randomizadas para a avaliação da acurácia média do modelo. Utilizou-se a matriz de confusão obtida a partir da classificação dos elementos do conjunto de teste de cada pasta. O algoritmo responsável por essa implementação foi desenvolvido utilizando as bibliotecas da linguagem Python e os resultados foram comparados com aqueles obtidos a partir da formulação matemática do modelo no software MATLAB, alcançando uma acurácia média de 93,03% em ambas as implementações.**

***Index Terms*—regressor logístico, gradiente descendente, k-fold, matriz de confusão.**

I. Introdução

A regressão logística é um método multivariável de classificação usado para atribuir observações a um conjunto binário (0 ou 1) de classes. Alguns dos exemplos deste tipo de classificação são: e-mail spam (sim ou não), transações de pagamento online (fraude ou não fraude), diagnósticos de câncer (tumor maligno ou benigno), dentre outros. Apesar do termo "regressão", trata-se de fato de uma técnica de classificação, uma vez que o método transforma o resultado da regressão logística em um valor de probabilidade (entre 0 e 1) aplicando a função sigmóide.

Os métodos multivariáveis visam encontrar a relação entre variáveis preditoras independentes x_i e a saída y_i, por meio da soma de produtos de coeficientes w_i (ou pesos) pelas variáveis independentes. Tais coeficientes são obtidos por meio do melhor ajuste matemático para o conjunto de dados e classes de saída especificadas.

Como exemplo, o trabalho de M. Hajmeer e I. Basheer[1] comparou os indicadores de performance da Regressão Logística com classificadores utilizando redes neurais artificiais de retropropagação e redes neurais probabilísticas (PNN), quando aplicados ao problema de classificação dos dados de crescimento ou ausência de crescimento de bactérias Escherichia coli R31 patogênica submetidas à variações de temperatura e atividade aquática.

Da mesma forma, no presente estudo, foi implementado o modelo de Regressão Logística de 01 camada e aplicado à base de dados 'OvarianInputs', com os resultados de exames de espectometria de 216 pacientes. Tais exames mediram as intensidades de íons correspondentes a 100 valores específicos de carga-massa a fim de determinar o diagnóstico dos pacientes com ou sem câncer de ovário (Fig. 1).

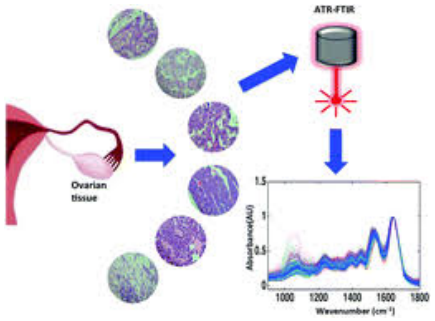

Figura 1. Espectometria para Diagnóstico do Câncer de Ovário

Esse é um caso de aprendizado supervisionado, uma vez que os resultados (0:sem câncer, 1: com câncer), já conhecidos para este conjunto de dados de exames, serão utilizados para a obtenção dos pesos ou coeficientes que descrevem a equação do modelo de classificação.

Na seção 2 será apresentada uma breve Fundamentação Teórica sobre Regressão Logística de 1 camada, na seção 3 a Metologia adotada para e implementação em Python e Matlab, na seção 4 a comparação dos Resultados encontrados e, por fim, na seção 5 as Conclusões sobre a aplicação do método no problema proposto.

II. Fundamentação Teórica

A regressão logística é uma técnica de Aprendizado de Máquina utilizada para aplicações de classificação binária em que a saída y_i assume apenas 02 valores de classe: 0 e 1. Apesar de sua arquitetura simples, com uma única camada ou neurônio, este tipo de classificador apresenta boa acurácia com baixo custo computacional para o treinamento dos parâmetros ou pesos w_i associados a cada variável preditora x_i, conforme ilustrado na figura 2.

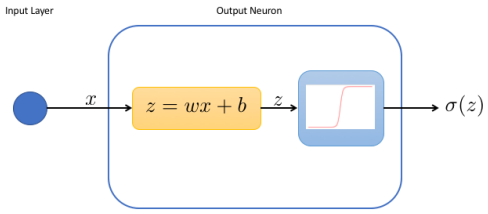

Figura 2. Representação do Modelo de um Regressor Logístico

Semelhante ao método de regressão linear, cujo objetivo é obter os valores w_i dos coeficientes que definem o peso sobre cada variável de entrada x_i, a regressão logística difere-se pela aplicação em sua saída de uma função não linear, chamada de sigmóide ou função logística que retorna valores na faixa de 0 a 1 conforme figura 3.

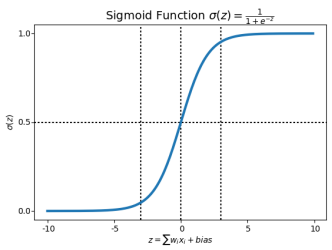

Figura 3. Função Logística ou Sigmóide

A vantagem de aplicar a função sigmóide na saída é que isso nos permite definir uma limite (*threshold*) para separação das classes. Neste trabalho, por exemplo, definimos que valores acima do limite de 0.5 seriam da classe 1 (pacientes com câncer) e valores menores ou igual a esse limite da classe 0 (pacientes sem câncer). De uma outra forma, podemos dizer que as predições feitas por regressão logística representam a probabilidade de uma determinada instância de dados pertencer à classe 0 ou classe 1.

A. Função de Custo

No método de regressão linear, normalmente, utiliza-se a função quadrática representada na equação 1, dada a sua convexidade e, portanto, a convergência do algoritmo para um ponto de mínimo único.

$$J(\theta) = \frac{1}{2}\sum_{n=1}^{m}(h_\theta(x_i) - y_i)^2 \tag{1}$$

Entretanto, para o problema de classificação binária, caso essa mesma função fosse aplicada a um algoritmo de Regressão Logística, a não linearidade produzida pela função sigmóide no cálculo das predições $h_\theta(x_i)$ comprometeria a convergência do algoritmo devido à introdução de vários mínimos locais (figura 4):

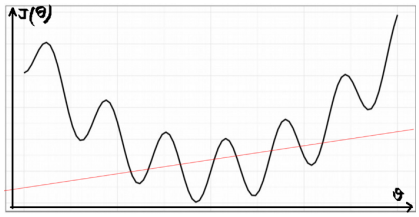

Figura 4. Função de Custo Quadrática com vários mínimos locais

Para evitar esse efeito, formulamos na equação 2 a hipótese $h_\theta(x)$ de um conjunto de parâmetros θ associados a um conjunto de variáves preditoras x_i, de tal forma que a saída esteja no intervalo de [0,1], em que g() representa a função sigmóide:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{(-\theta^T x)}} \tag{2}$$

Tal hipótese representa a probabilidade estimada de que a saída y_i seja igual a 1 quando submetida à entrada x_i parametrizada por θ_i (eq.3):

$$h_\theta(x) = P(y = 1|x;\theta) = 1 - P(y = 0|x;\theta) \tag{3}$$

Para tornar a função de custo seja convexa (e, portanto, garantir a convergência para o mínimo global), aplicamos uma transformação usando o logaritmo da função sigmóide conforme equação 4 a seguir:

$$Erro(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)), & p/ \quad y = 1 \\ -log(1 - h_\theta(x)), & p/ \quad y = 0 \end{cases} \tag{4}$$

Essa função de custo pode então ser representada por uma única equação dada por (eq 5):

$$Erro(h_\theta(x), y) = -ylog(h_\theta(x)) - (1 - y)(log(1 - h_\theta(x))) \tag{5}$$

Na figura 5 ilustramos o gráfico dessa função destacando sua convexidade para as 2 classes binárias y_i [0,1].

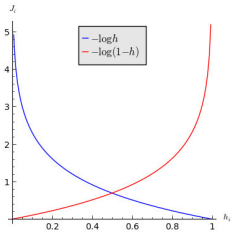

Figura 5. Função de Custo Logística

B. Gradiente Descendente

Definida a função de custo que representa o erro entre $h_\theta(x)$ e y_i, precisamos adotar um método de otimização para encontrar os valores dos coeficientes θ ou w_i que minimizem essa função de custo. Neste projeto, utilizou-se o método do gradiente descendente como otimizador, dada a sua simplicidade de cálculo representada pelas equações 6 e 7 a seguir:

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \tag{6}$$

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x_i) - y_i).x_j \tag{7}$$

C. Métricas de avaliação

De acordo com [2], uma matriz de confusão para um problema de n classes é uma matriz M de ordem nXn, no qual o elemento M_{ij} corresponde ao número de classificações corretas para a classe i, quando i = j. Para $i \neq j$, o elemento M_{ij} corresponde ao número de classificações da classe i que foram definidas incorretamente como classe j, e vice-versa.

Como estabelecido por [3], o número de acertos para cada classe se localiza na diagonal principal da matriz M.

Usualmente podem ser depreendidas algumas métricas de avaliação do modelo de classificação apenas com base na sua matriz de confusão. Neste trabalho, serão utilizadas as métricas de acurácia, sensibilidade e especificidade, definidas nas expressões 8, 9 e 10.

$$acurácia = \frac{acertos}{número\ de\ classificações} \tag{8}$$

$$sensibilidade = \frac{acertos\ de\ casos\ positivos}{número\ de\ casos\ positivos} \tag{9}$$

$$especificidade = \frac{acertos\ de\ casos\ negativos}{número\ de\ casos\ negativos} \tag{10}$$

III. Metodologia

Nesta seção serão descritos os procedimentos metodológicos necessários à solução do problema proposto. Serão apresentadas as ferramentas de software, estratégia de partição do conjunto de treinamento do modelo de classificação e por fim a métrica de avaliação utilizada.

A. Ferramentas computacionais

1) Biblioteca Numpy: A biblioteca NumPy é fundamental para a computação científica em Python. Através desta biblioteca é possível manipular objetos do tipo array, com múltiplas dimensões, incluindo suas variações como é o caso das matrizes. Além disso, são fornecidas operações de álgebra linear, estatística e muitas outras. [4]

2) Biblioteca Pandas: Pandas é uma biblioteca de código livre que fornece métodos práticos de se operar com estruturas de dados [5]. Para o estudo de ciência de dados, é especialmente útil na manipulação de datasets. Dentre os recursos oferecidos estão: redimensionamento, agrupamento, substituições, entre outros. As extensões de arquivos permitidas incluem os formatos ".xlsx", ".xls", ".csv', por exemplo.

3) Biblioteca Matplotlib: Matplotlib é uma biblioteca para visualização de dados estatísticos e criação de gráficos [6]. Pyplot é um módulo do matplotlib que fornece uma interface semelhante ao Matlab, com a vantagem de ser de código aberto e totalmente gratuito. Esta biblioteca pode ser integrada a scripts desenvolvidos na linguagem Python.

4) Biblioteca Tensorflow: TensorFlow é uma biblioteca de código aberto para aprendizado de máquina aplicável a uma ampla variedade de tarefas. É um sistema para criação e treinamento de redes neurais para detectar e decifrar padrões e correlações.[7]

B. Descrição do problema proposto

A base de dados "ovarianInputs" é constituída por 216 observações, das quais 121 representam pacientes diagnosticados com câncer e seus respectivos resultados no exame de espectrometria. Os 100 atributos de cada observação são valores reais compreendidos na faixa -0,0776 a 6,57081.

Neste trabalho, os 100 atributos de cada observação serão utilizados como a primeira camada de uma Rede Neural Artificial, modelada por um Regressor Logístico. Com este modelo objetiva-se diagnosticar determinado paciente como portador de câncer baseando-se nos dados obtidos no seu exame de espectrometria. Este modelo deve ser treinado e testado de acordo com o método de validação cruzada, ou seja, o dataset será dividido em 5 pastas distintas, estando estes conjuntos particionados em subconjunto de treino e subconjunto de teste. Como métrica de avaliação, para cada conjunto de teste, será elaborada uma matriz de confusão, visando extrair a acurácia local. Ao final do experimento, devem ser apresentados os valores de acurácia, especificidade e sensibilidade.

C. Implementação do modelo

A implementação deste trabalho foi realizada através do desenvolvimento de um script na linguagem Python, tendo o suporte dos pacotes Numpy, Pandas e Tensorflow. Para a edição e execução do código foi utilizado o ambiente de desenvolvimento Visual Studio Code. Para fins de comparação dos resultados, o mesmo modelo foi implementado no software Matlab com o suporte das definições matemáticas apresentadas na seção II.

Inicialmente foi realizada a leitura do dataset através do método *read_excel* do pacote Pandas, seguida da inserção de uma coluna adicional com o valor 1 para fins de polarização. O treinamento ocorreu em 5 pastas, sendo utilizado o método de validação cruzada como o intuito de variar o conjunto de treinamento ao longo de todo o dataset, e avaliar qual possui melhor desempenho. Estando o conjunto de entrada com quantidades desiguais de classes, sendo a classe 1 majoritária, com 121 observações, adotou-se o procedimento de shuffle, isto é,

dentro de uma pasta os dados forem coletados aleatoriamente, de forma a não ocorrer predominância de uma classe ou outra. A figura 6 ilustra essa metodologia de segmentação de dados.

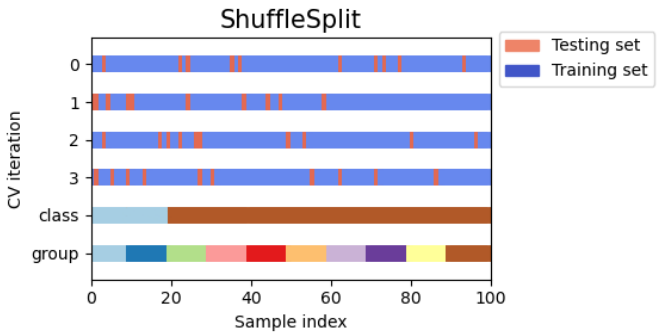

Figura 6. Método Shuffle Split - Fonte: [8]

Ao utilizar 5 pastas, o conjunto de treinamento constituiu-se de 80% dos dados, o que corresponde a 173 observações, restando 46 observações para o conjunto de teste.

Para a realização do fluxo de aprendizado de máquina, foram utilizadas algumas etapas fundamentais, a citar:

- Definição de cada Neurônio: Na ferramenta Tensorflow esta funcionalidade foi implementada pela função **tf.keras.Sequential**. Para esta aplicação definiu-se o neurônio com 1 camada, sendo a função de ativação a sigmóide.
- Definição da função de otimização: Na ferramenta Tensorflow esta funcionalidade foi implementada através do método **model.compile**, com o parâmetro 'SGD', indicando que se trata do gradiente descendente.
- Definição da função de perda: Ainda no método **model.compile** utilizou-se o parâmetro **binary_crossentropy**, equivalente ao logaritmo da perda, de acordo com [9].
- Compilação do modelo
- Treinamento do modelo: Realizado pela função **model.fit**, parametrizada com o conjunto de treino e número de épocas.
- Teste do modelo e obtenção de métricas

O treinamento do modelo, tanto na ferramenta Tensorflow quanto no Matlab, foi realizado com uma taxa de aprendizado adaptativa, isto é, a partir de 600 épocas diminuiu-se por um fator de 10.

O fluxograma exibido na figura 7 apresenta o procedimento completo de treinamento da máquina, obtendo a cada pasta a acurácia de classificação para o conjunto de dados apresentados. Por fim, obteve-se a média das acurácias alcançadas em cada uma das pastas, bem como a sua respectiva matriz de confusão, a fim de analisar o desempenho médio do classificador.

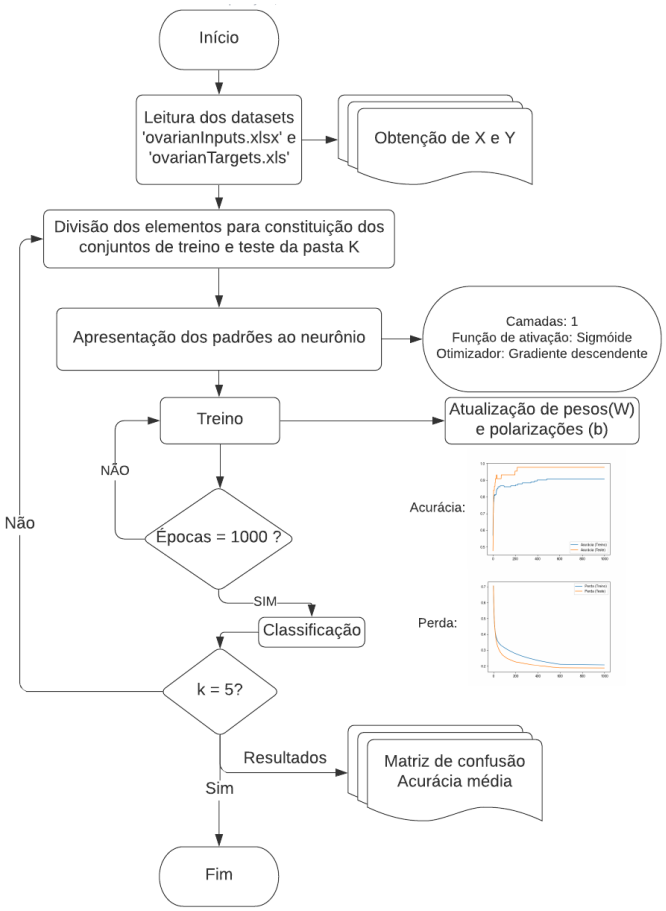

Figura 7. Fluxograma do algoritmo do regressor logístico

IV. Resultados

Dado que o processo de validação cruzada foi realizado de maneira randômica, existem diversas performances possíveis de serem obtidas com o modelo utilizado. Dentre os experimentos realizados, a tabela I exibe aqueles cujas pastas obtiveram melhor êxito para a solução baseada na biblioteca Tensorflow, considerando as métricas de acurácia, especificidade e sensibilidade.

Tabela I
Desempenho obtido com a ferramenta Tensorflow

Métrica	Exper. 1	Exper. 2	Exper. 3	Exper. 4	Exper. 5
Acurácia	97,73%	93,02%	93,02%	97,67%	83,72%
Sensibilidade	95,83%	96,15%	82,35%	96%	75,86%
Especificidade	100%	88,24%	100%	100%	100%
Acurácia Média	93,03%				
Sensibilidade Média	89,24%				
Especificidade Média	97,65%				

*Para este experimento a acurácia máxima foi obtida na pasta 1

Tabela II
DESEMPENHO OBTIDO COM A FERRAMENTA MATLAB

Métrica	Exper. 1	Exper. 2	Exper. 3	Exper. 4	Exper. 5
Acurácia	95,34%	90,69%	93,02%	95,34%	93,02%
Sensibilidade	92%	84%	88%	92%	88%
Especificidade	100%	94,74%	94,74%	100%	94,74%
Acurácia Média	93,48%				
Sensibilidade Média	88,80%				
Especificidade Média	96,844%				

*Para este experimento a acurácia máxima foi obtida na pasta 1

Como maneira de expressar graficamente o resultado das classificações e facilitar o processo de análise dos resultados, foram geradas as matrizes de confusão para cada pasta através dos métodos da biblioteca matplotlib. A figura 8 ilustra a matriz de confusão da pasta 1, cujo valor de acurácia superou as demais.

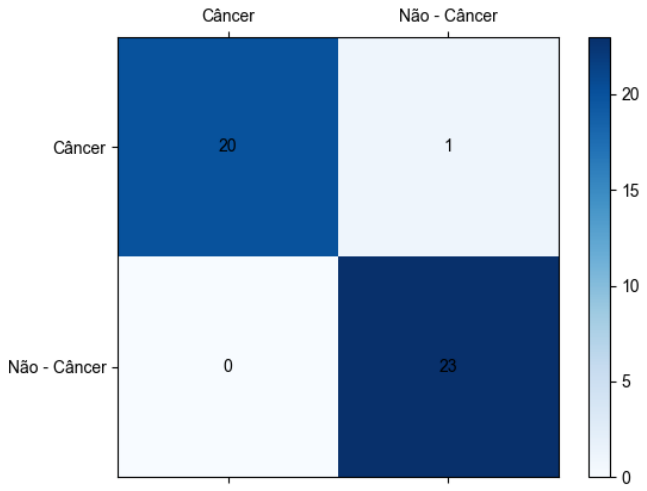

Figura 8. Matriz de confusão para a pasta 1 - acurácia = 97,67

V. CONCLUSÕES

A implementação do regressor logístico para a rede neural de 1 camada, teve como característica a simplicidade de implementação na ferramenta Tensorflow, possibilitando agilidade para a validação de modelos de redes neurais com diversas topologias e camadas. Da mesma maneira, com os conceitos vistos em sala de aula foi possível desenvolver um algoritmo baseado na rede de propagação direta com o apoio do software Matlab, uma vez que as funções de custo, de ativação e gradiente descente puderam ser modeladas matematicamente com os recursos da plataforma. O método de validação cruzada seguiu os mesmos princípios abordados em trabalhos anteriores, adicionando o fato de que os conjuntos de teste e treinamento foram tomados de maneira aleatória, dado que a quantidade de observações para cada classe não é mesma. Os cinco experimentos realizados obtiveram as métricas exibidas na tabela 1. As acurácias média de 93,03% e máxima de 97,73% indicam um nível de assertividade aceitável, visto que para a pasta 1 exibida na figura 8, apenas 1 de 44 itens foi classificado erroneamente. Foi possível observar que o modelo gerado possuiu alto índice de especificidade para todas as pastas, indicando que apresenta confiabilidade em classificar pacientes que não possuem câncer. Portanto, o regressor logístico mostra-se robusto e de boa performance para dados reais, com a vantagem de possuir baixo custo computacional. O modelo apresenta perspectiva de melhorias para um conjunto maior de dados e aumento no número de camadas.

REFERÊNCIAS

[1] M. Hajmeera and I. Basheerb, ”Comparison of logistic regression and neural network-based classifiers for bacterial growth,”in Food Microbiology, Elsevier, 2003.

[2] S. Godbole and S. Sarawagi, ”Discriminative methods for multi-labeled classification,”in Pacific-Asia conference on knowledge discovery and data mining. Springer, 2004.

[3] M. Douglas C, Applied statistics and probability for engineers. John Wiley Sons, Inc, 2002.

[4] “Numpy: About Us,” 2020. Accessed on: May. 05, 2021. [Online]. Available: https://numpy.org/about/

[5] “Pandas documentation,” 2021. Accessed on: May. 05, 2021. [Online]. Available: https://pandas.pydata.org/docs/

[6] “Matplotlib: Visualization with Python,” 2021. Accessed on: May. 05, 2021. [Online]. Available: https://matplotlib.org/stable/index.html

[7] ”Tensorflow: An end-to-end open source machine learning platform,”2021. Accessed on: May. 07, 2021. [Online]. Available: https://www.tensorflow.org/

[8] ”Visualizing cross-validation behavior in scikit-learn”2021.

[9] ”Tensorflow: tf.keras.losses.binary_crossentropy,”2021. Accessed on: May. 07, 2021. [Online]. Available: https://www.tensorflow.org/api

```
import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns


# learning rate scheduler
def schedule(epoch, lr, logs = {}):
    if epoch >= 600:
        return 0.0001
    return 0.001

#Leitura do banco de dados e vari veis alvo
data = pd.read_excel('ovarianInputs.xlsx',header = None)
target_data = pd.read_excel('ovarianTargets.xls',header = None)

#Defini  o de Labels do gr fico da matriz de confus o
graph_labels = ['C ncer','N o - C ncer']

#Data
x = np.array(data).reshape((-1,100))
ones_columm = np.ones(216,dtype=float).reshape(-1,1)
x = np.concatenate([x,ones_columm],axis = 1)

#Targets
y = np.array(target_data[0]).reshape((-1,1))

# Define the K-fold Cross Validator
kfold = KFold(n_splits=5, shuffle=True)
#skf = StratifiedKFold(n_splits=5)

# K-fold Cross Validation model evaluation
fold_no = 1

accuracies = list()
sensibilities = list()
specificities = list()

for train, test in kfold.split(x, y):
    X_train, X_test, y_train, y_test = x[train],x[test],y[train],y[test]

    N, D = X_train.shape
    #escalonamento [0,1]

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    print(X_train)
    print(X_test)
    #input('break')

    #Defini  o da primeira camada de neur nio
    model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(D,)),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    #utilizar a fun  o de perda log stica
    model.compile(optimizer='SGD',
              loss='binary_crossentropy',
```

```
            metrics=['accuracy'])

scheduler = tf.keras.callbacks.LearningRateScheduler(schedule)

# Train the model
r = model.fit(X_train, y_train, validation_data=(X_test, y_test)
,epochs=1000, callbacks = [scheduler])

aux = model.layers[0].get_weights()
#Pesos
print(aux[0])

# Evaluate the model - evaluate() returns loss and accuracy
print("Train score:", model.evaluate(X_train, y_train))
print("Test score:", model.evaluate(X_test, y_test))

print('Making predictions')
count = 0
sensibility = 0
specificity = 0
P = model.predict(X_test)
P = np.round(P).flatten()
for j,element in enumerate(P):
    if element == y_test[j]:
        count = count + 1
        if element == 0:
            specificity = specificity + 1
        else:
            sensibility = sensibility + 1

true_quantity = np.count_nonzero(y_test == 1)
false_quantity = np.count_nonzero(y_test == 0)

acc = round(count/len(P),4)
sens = round(sensibility/true_quantity,4)
spec = round(specificity/false_quantity,4)
# Calculate the accuracy, compare it to evaluate() output
print("Manually calculated accuracy:", acc)
print("Manually calculated sensibility:", sens)
print("Pacientes com cancer: (y_true) ",true_quantity)
print("Pacientes normais: (y_true) ",false_quantity)

#testing = np.concatenate([y_test,np.array(P).reshape((-1,1))],axis = 1)
#print(testing)

accuracies.append(acc)
sensibilities.append(sens)
specificities.append(spec)


#Bloco de codigo para estilizacao e visualizacao da matriz de confusao
window_name = 'Pasta ' + str(fold_no) + ' Matriz de confus o'
fig = plt.figure(window_name)
matrix = confusion_matrix(P,y_test)
#print(matrix)
ax = fig.add_subplot(111)
cax = ax.matshow(matrix,cmap = 'Blues')
for (m,n),label in np.ndenumerate(matrix):
    ax.text(n,m,int(label),ha='center',va='center')
ax.set_xticklabels(['']+graph_labels)
ax.set_yticklabels(['']+graph_labels)
fig.colorbar(cax)

# Plot what's returned by model.fit()
window_name = 'Pasta ' + str(fold_no) + ' Perdas'
fig = plt.figure(window_name)
```

```
    plt.plot(r.history['loss'], label='Perda (Treino)')
    plt.plot(r.history['val_loss'], label='Perda (Teste)')
    plt.legend()

    # Plot the accuracy
    window_name = 'Pasta ' + str(fold_no) + ' Acur cias'
    fig = plt.figure(window_name)
    plt.plot(r.history['accuracy'], label='Acur cia (Treino)')
    plt.plot(r.history['val_accuracy'], label='Acur cia (Teste)')
    plt.legend()

    #plt.show()

    fold_no = fold_no + 1

print('Acur cias: ',accuracies)
print('Sensibilidades: ',sensibilities)
print('ESpecificidades: ',specificities)
print('Media: ',np.mean(accuracies))
print('Maxima: ',np.max(accuracies))

accuracies.append(np.mean(accuracies))
sensibilities.append(np.mean(sensibilities))
specificities.append(np.mean(specificities))

metricas = [accuracies, sensibilities ,specificities]
print(metricas)

labels = ['Pasta 1', 'Pasta 2', 'Pasta 3', 'Pasta 4', 'Pasta 5', 'Mean']

sns.set_theme(style= 'whitegrid')

x = np.arange(len(labels))  # the label locations
width = 0.25# the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, accuracies, width, label='Acur cia')
rects2 = ax.bar(x + width/2, sensibilities, width, label='Sensibilidade')
rects3 = ax.bar(x + 3*width/2, specificities, width, label='Especificidade')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_title('Scores')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc = 'lower right')

ax.bar_label(rects1, padding=3)
ax.bar_label(rects2, padding=3)
ax.bar_label(rects3, padding=3)

fig.tight_layout()

plt.show()
```

```
close all;
clear all;

X = xlsread('ovarianInputs.xlsx');
R = xlsread('ovarianTargets.xls');
Y = R(:,1);
m = length(Y);
[r,c] = size(X);
idx = randperm(m);
%y = Y+1;
y = Y;
p = 0.8;
wMax = zeros(101);

xtrain = X(idx(1:round(p*m)),:);
ytrain = y(idx(1:round(p*m)),:);
xtest  = X(idx(round(p*m)+1:length(idx)),:);
ytest  = y(idx(round(p*m)+1:length(idx)),:);
Num_Pastas= 5;
Acuracia_Media = 0;

for Pasta = 1:Num_Pastas

% Atualiza w pelo gradiente descendente
alpha = 0.001;
w = double(randperm(c+1)/100);
w = -w'; % vetor de pesos iniciais (negativos)
%w = w'; % vetor de pesos iniciais (positivos)
mtrain  = length(ytrain);
xtrain2 = [ones(mtrain,1) xtrain];
Acuracia_TrainingSet = 0.0;

cont  = 1;
epoch = 1;
while (epoch <= 1200)

    if epoch >= 600
        alpha = 0.0001; % aumenta a precis o em 10x ap s 600  pocas
    end

    ztrain  = xtrain2*w;
    htrain = 1.0./(1.0 + exp(-ztrain));

    for j = 1:c+1
        somatoria = 0.0;
        for i = 1:mtrain
            somatoria = somatoria + alpha*double((htrain(i) - ytrain(i))*xtrain2(i,j));
        end
            w(j) = w(j) - somatoria;
    end
    Loss = 0.0;
    for i = 1:mtrain
        Loss = Loss + ytrain(i)*log(htrain(i)) + (1-ytrain(i))*log(1-htrain(i));
    end
    Loss = Loss * (-1.0/mtrain);
    perda(cont) = Loss;
    cont = cont + 1;

    ytrainpred = double(htrain > 0.5);
    %ytrainpred = ytrainpred+1;
    Acuracia_TrainingSet = mean(double(ytrainpred == ytrain))*100;
    epoch = epoch + 1;
end
fprintf('Pasta No: %d',Pasta);
```

```
%if Pasta == 1
%    max = Acuracia_TrainingSet
%    PastaMax = Pasta;
%else
%    if Acuracia_TrainingSet > max
%        max = Acuracia_TrainingSet;
%        wMax = w;
%        PastaMax = Pasta;
%    end
%end

Acuracia_TrainingSet
figure
plot(perda);
histogram(htrain,10);

mtest  = length(ytest);
xtest2 = [ones(mtest,1) xtest];
ztest  = xtest2*w;

% Evaluate the Model (Test DataSet)
htest = 1.0./(1.0 + exp(-ztest));
ytestpred = double(htest > 0.5);
ytestpred_sem = double(htest <= 0.5);

Acuracia_TestingSet = mean(double(ytestpred == ytest))*100
histogram(htest,10);

Acuracia_Media = Acuracia_Media + Acuracia_TestingSet;

if Pasta == 1
    max = Acuracia_TestingSet
    PastaMax = Pasta;
else
    if Acuracia_TestingSet > max
        max = Acuracia_TestingSet;
        wMax = w;
        PastaMax = Pasta;
    end
end


% Matriz de Confus o
true_cancer      = sum(double(ytest == 1))
true_no_cancer    = sum(double(ytest == 0))
predicted_cancer = sum(double(ytestpred == 1))
true_positive    = sum(double(ytest == 1) .* double(ytestpred == 1))
true_negative    = sum(double(ytest == 0) .* double(ytestpred_sem == 1))

false_cancer = sum(double(ytest == 0))

Precision = 100.0*true_positive/predicted_cancer
Recall = 100.0*true_positive/true_cancer

specificity = 100.0*true_negative/true_no_cancer
sensibility = 100.0*true_positive/true_cancer
%F1 Score

F1 = 2*Precision*Recall/(Precision + Recall)

perda = 1;
end

PastaMax
Acuracia_Media = Acuracia_Media/5
max
xlswrite('w-cf-5fold.xls',wMax);
```