MLP para classificação binária

Ionosphere binary classification dataset
https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.csv
(https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.csv)
https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.names
(https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.names))
Previsão da existencia de uma estrutura na atomosfera com dados de um radar.

Pipeline de um processo de Deep Learning implementado em PyTorch:

1. Preparar os Dados
2. Definir o Modelo
3. Treinar o Modelo
4. Avaliar o Modelo
5. Usar o Modelo

Melhoramento do Modelo

In [1]:
```
# Confirmar a instalação

import torch
print(torch.__version__)
```

```
1.8.1+cu111
```

Imports

In [2]:
```
# pytorch mlp for binary classification
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
from torch import Tensor
from torch.nn import Linear
from torch.nn import ReLU
from torch.nn import Sigmoid
from torch.nn import Module
from torch.optim import SGD, Adam
from torch.nn import BCELoss, BCEWithLogitsLoss
from torch.nn.init import kaiming_uniform_
from torch.nn.init import xavier_uniform_
```

In [3]:
```
#Constants

#path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master
/ionosphere.csv'
PATH = 'ionosphere.csv'

device = torch.device("cpu")

EPOCHS = 50
BATCH_SIZE = 64
LEARNING_RATE = 0.001
```

*1. Preparar os Dados

In [4]:
```
# definição classe para o dataset
class CSVDataset(Dataset):
    # ler o dataset
    def __init__(self, path):
        # ler o ficheiro csv para um dataframe
        df = pd.read_csv(path, header=None)
        # separar os inputs e os outputs
        self.X = df.values[:, :-1]
        self.y = df.values[:, -1]
        # garantir que os inputs sejam floats
        self.X = self.X.astype('float32')
        # fazer o encoding dos outputs (label) e garantir que sejam
floats
        self.y = LabelEncoder().fit_transform(self.y) #faz o fit e t
ransforma no self.y o 'g' e o 'b' em 0 e 1
        self.y = self.y.astype('float32')
        self.y = self.y.reshape((len(self.y), 1))

    # número de casos no dataset
    def __len__(self):
        return len(self.X)

    # retornar um caso
    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    # retornar índices para casos de treino e de teste
    def get_splits(self, n_test=0.33):
        # calcular o tamanho para o split
        test_size = round(n_test * len(self.X))
        train_size = len(self.X) - test_size
        # calcular o split do houldout
        return random_split(self, [train_size, test_size], generator
=torch.Generator().manual_seed(42))

# preparar o dataset
def prepare_data(path):
    # criar uma instância do dataset
    dataset = CSVDataset(path)
    # calcular o split
    train, test = dataset.get_splits()
    # preparar os data loaders
    train_dl = DataLoader(train, batch_size=len(train), shuffle=Tru
e) #32 len(train)
    test_dl = DataLoader(test, batch_size=1024, shuffle=False)
    train_dl_all = DataLoader(train, batch_size=len(train), shuffle=
False)
    test_dl_all = DataLoader(test, batch_size=len(test), shuffle=Fal
se)
    return train_dl, test_dl, train_dl_all, test_dl_all

# preparar os dados
train_dl, test_dl,  train_dl_all, test_dl_all = prepare_data(PATH)

# sanity check
x,y = next(iter(train_dl))
print(x.shape, y.shape)
x,y = next(iter(test_dl))
print(x.shape, y.shape)
```

```
torch.Size([235, 34]) torch.Size([235, 1])
torch.Size([116, 34]) torch.Size([116, 1])
```

*1.1 Visualizar os Dados

In [5]:
```
from IPython.display import display

def visualize_data(path):
    # criar uma instância do dataset
    df = pd.read_csv(path, header=None)
    display(df)

def visualize_dataset(train_dl, test_dl):
    print(f"Quantidade de casos de Treino:{len(train_dl.dataset)}")
    print(f"Quantidade de casos de Teste:{len(test_dl.dataset)}")
    x, y = next(iter(train_dl)) #fazer uma iteração nos loaders para ir buscar um batch de casos
    print(f"Shape tensor batch casos treino, input: {x.shape}, output: {y.shape}")
    x, y = next(iter(test_dl))
    print(f"Shape tensor batch casos teste, input: {x.shape}, output: {y.shape}")

visualize_data(PATH)
visualize_dataset(train_dl, test_dl)
```

	0	1	2	3	4	5	6	7	8	9	...
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...
...	...	...	...	...	...	...	...	...	...	...	...
346	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...
347	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...
348	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...
349	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...
350	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...

351 rows × 35 columns

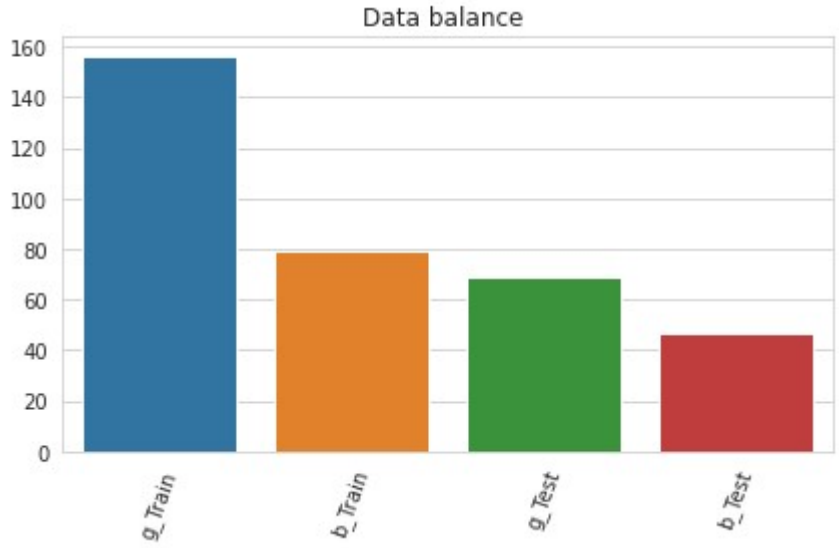
```
Quantidade de casos de Treino:235
Quantidade de casos de Teste:116
Shape tensor batch casos treino, input: torch.Size([235, 34]), output: torch.Size([235, 1])
Shape tensor batch casos teste, input: torch.Size([116, 34]), output: torch.Size([116, 1])
```

*1.2 Verificar balanceamento do dataset

In [6]:

```
import seaborn as sns
import matplotlib.pyplot as plt

def visualize_holdout_balance(y_train, y_test):
    _, y_train = next(iter(train_dl_all))
    _, y_test = next(iter(test_dl_all))

    sns.set_style('whitegrid')
    casos_treino=len(y_train) # calcular o nº de casos de treino
    casos_test=len(y_test) # calcular o nº de casos de teste
    b_Train=np.count_nonzero(y_train == 0) # calcular o nº de 0 nos
casos de treino
    g_Train = np.count_nonzero(y_train == 1) # calcular o nº de 1 no
s casos de treino
    b_Test=np.count_nonzero(y_test == 0) # calcular o nº de 0 nos ca
sos de teste
    g_Test = np.count_nonzero(y_test == 1) # calcular o nº de 1 nos
casos de teste
    print("casos_treino:",casos_treino)
    print("g_Train: ", g_Train)
    print("b_Train: ", b_Train)
    print("g_Train/b_Train: ", g_Train/b_Train) # rácio de g em b
    print("casos_test:",casos_test)
    print("g_Test: ", g_Test)
    print("b_Test: ", b_Test)
    print("g_Test/b_Test: ", g_Test/b_Test) # rácio de g em b

    grafico=sns.barplot(x=['g_Train','b_Train', 'g_Test', 'b_Test'],
                        y=[g_Train,b_Train, g_Test, b_Test])
    grafico.set_title('Data balance ')
    plt.xticks(rotation=70)
    plt.tight_layout()
    plt.show()

visualize_holdout_balance(train_dl_all, test_dl_all)
```

```
casos_treino: 235
g_Train:  156
b_Train:  79
g_Train/b_Train:  1.9746835443037976
casos_test: 116
g_Test:  69
b_Test:  47
g_Test/b_Test:  1.4680851063829787
```



*2. Definir o Modelo

In [7]:
```
# Instalar o torchinfo
# !pip install torchinfo
```

In [9]:

```
from torchinfo import summary

# Definição da classe para o modelo
class MLP(Module):
    # definir elementos do modelo
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        # input para a primeira camada - Linear - ReLU
        self.hidden1 = Linear(n_inputs, 10)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu') #
He initialization
        self.act1 = ReLU()
        # segunda camada - Linear - ReLU
        self.hidden2 = Linear(10, 8)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        # terceira camada e output Linear - Sigmoid
        self.hidden3 = Linear(8, 1)
        xavier_uniform_(self.hidden3.weight) # Glorot initialization
        self.act3 = Sigmoid()

    # sequência de propagação do input
    def forward(self, X):
        # input para a primeira camada
        X = self.hidden1(X)
        X = self.act1(X)
        # input para a segunda camada
        X = self.hidden2(X)
        X = self.act2(X)
        # input para a terceira camada e output
        X = self.hidden3(X)
        X = self.act3(X)
        return X

# definir a rede neuronal
model = MLP(34)
# visualizar a rede
batch_size = 32
# sumário da rede
print(summary(model, input_size=(batch_size, 34), verbose=0)) #verbo
se=2 Show weight and bias layers in full detail
model.to(device)
```

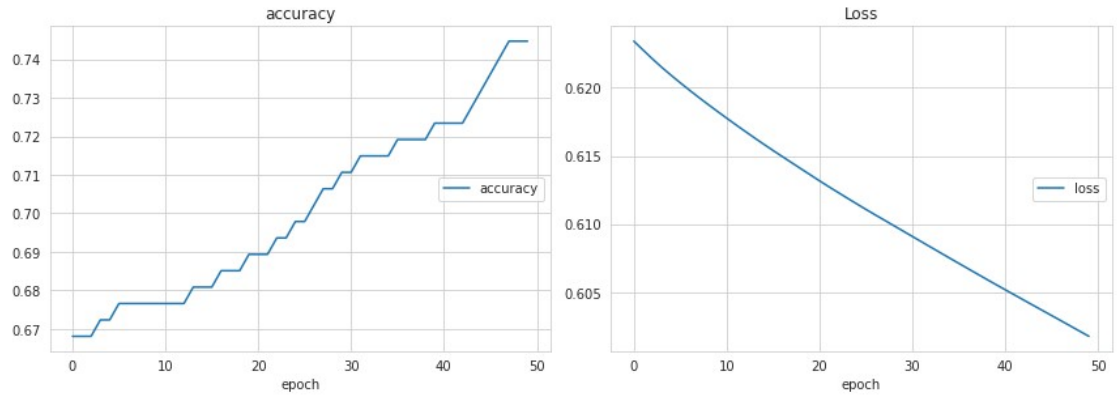
```
==========================================================================
========================
Layer (type:depth-idx)                   Output Shape
Param #
==========================================================================
========================
├─Linear: 1-1                            [32, 10]
350
├─ReLU: 1-2                              [32, 10]
--
├─Linear: 1-3                            [32, 8]
88
├─ReLU: 1-4                              [32, 8]
--
├─Linear: 1-5                            [32, 1]
9
├─Sigmoid: 1-6                           [32, 1]
--
==========================================================================
========================
Total params: 447
Trainable params: 447
Non-trainable params: 0
Total mult-adds (M): 0.01
==========================================================================
========================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.01
==========================================================================
========================
```

Out[9]:
```
MLP(
  (hidden1): Linear(in_features=34, out_features=10, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=10, out_features=8, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=8, out_features=1, bias=True)
  (act3): Sigmoid()
)
```

*3. Treinar o Modelo

In [10]:
```
# Instalar o livelossplot
# !pip install livelossplot
```

In [11]:
```
#versão com display de gráfico
from livelossplot import PlotLosses

# treino do modelo
def train_model(train_dl, model):
    liveloss = PlotLosses() # para visualizarmos o processo de trein
o
    # definir a função de loss e a função de otimização
    # criterion = BCELoss() # Binary Cross Entropy - precisa de sign
oid como função de ativação na saida
    criterion = BCEWithLogitsLoss()
    # optimizer = SGD(model.parameters(), lr=LEARNING_RATE, momentum
=0.9) # stochastic gradient descent
    optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
    # iterar as epochs
    for epoch in range(EPOCHS):
        logs = {} # para visualizarmos o processo de treino
        # iterar as batchs
        epoch_loss  = 0 # para visualizarmos o processo de treino
        epoch_acc  = 0 # para visualizarmos o processo de treino
        for i, (inputs, labels) in enumerate(train_dl): # backpropag
ation
            # inicializar os gradientes
            optimizer.zero_grad() # coloca os gradientes de todos os
parametros a zero
            # calcular o output do modelo - previsao/forward
            outputs = model(inputs)
            # calcular o loss
            loss = criterion(outputs, labels)
            # calcular a accuracy
            #acc = binary_acc(outputs, labels)
            acc = accuracy_score(outputs.detach().numpy().round(), l
abels.numpy())
            # atribuição alteraçoes "In the backward pass we receive
a Tensor containing the gradient of the loss
            # with respect to the output, and we need to compute the
gradient of the loss with respect to the input.
            loss.backward() #backpropagation
            # update pesos do modelo
            optimizer.step()
            # calcular epochs de loss e epochs de accuracy
            epoch_loss += loss.item()
            epoch_acc += acc.item()

        print(f'Epoch {epoch:03}: | Loss: {epoch_loss/len(train_d
l):.5f} | Acc: {epoch_acc/len(train_dl):.3f}')
        logs['loss'] = epoch_loss # para visualizarmos o processo de
treino
        logs['accuracy'] = epoch_acc/len(train_dl) # para visualizar
mos o processo de treino
        liveloss.update(logs) # para visualizarmos o processo de tre
ino
        liveloss.send() # para visualizarmos o processo de treino

# treinar o modelo
train_model(train_dl, model)
```




```
accuracy
        accuracy                  (min:    0.668, max:    0.745, cu
r:    0.745)
Loss
        loss                      (min:    0.602, max:    0.623, cu
r:    0.602)
```

*4. Avaliar o Modelo

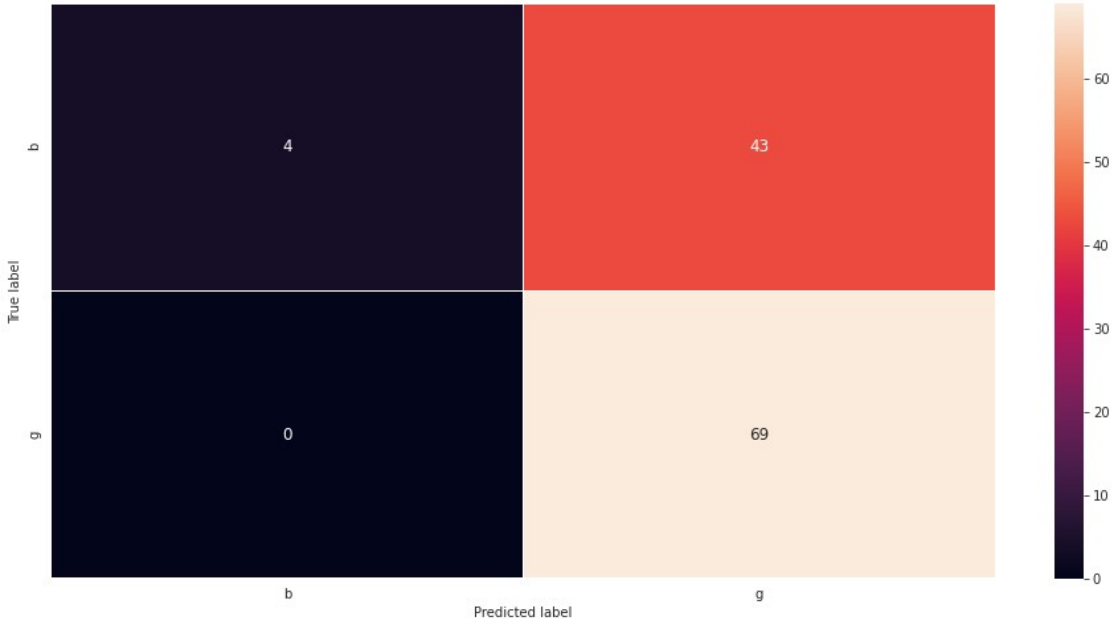
In [12]:

```
# Avaliar o modelo
def evaluate_model(test_dl, model):
    predictions = list()
    actual_values = list()
    for i, (inputs, labels) in enumerate(test_dl):
        # avaliar o modelo com os casos de teste
        yprev = model(inputs)
        # retirar o array numpy
        yprev = yprev.detach().numpy()
        actual = labels.numpy()
        # arredondar para obter a classe
        yprev = yprev.round()
        # guardar
        predictions.append(yprev)
        actual_values.append(actual)
    predictions, actual_values = np.vstack(predictions), np.vstack(a
ctual_values)
    return predictions, actual_values

def display_confusion_matrix(cm):
    plt.figure(figsize = (16,8))
    sns.heatmap(cm,annot=True,xticklabels=['b','g'],yticklabels=['b
','g'], annot_kws={"size": 12}, fmt='g', linewidths=.5)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

# avaliar o modelo
predictions, actual_values = evaluate_model(test_dl, model)
#actuals, predictions = evaluate_model(train_dl, model)
# calcular a accuracy
acc = accuracy_score(actual_values, predictions)
print(f'Accuracy: {acc:0.3f}\n')

acertou=0
falhou = 0
for r,p in zip(actual_values, predictions):
    print(f'real:{r} previsão:{p}')
    if r==p: acertou+=1
    else: falhou+=1
print(f'acertou:{acertou} falhou:{falhou}')

# relatório de classificação: precision, recall, f1-score, support v
s. 0,1, accuracy, macro avg, weighted avg
print(classification_report(actual_values, predictions))

# matriz confusão
cm = confusion_matrix(actual_values, predictions)
print (cm)
display_confusion_matrix(cm)
```

```
Accuracy: 0.629

real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
```

```
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[0.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
real:[0.] previsão:[1.]
real:[1.] previsão:[1.]
acertou:73 falhou:43
              precision    recall  f1-score   support
```

```
              0.0       1.00      0.09      0.16        47
              1.0       0.62      1.00      0.76        69

    accuracy                              0.63       116
   macro avg       0.81      0.54      0.46       116
weighted avg       0.77      0.63      0.52       116

[[ 4 43]
 [ 0 69]]
```



*5. Usar o Modelo

In [13]:
```
# fazer uma previsão utilizando um caso
def predict(row, model):
    # converter row para tensor
    row = Tensor([row])
    # fazer a previsão
    yprev = model(row)
    # retirar o array numpy
    yprev = yprev.detach().numpy()
    return yprev

# fazer uma única previsão (classe esperada = 1)
row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.037
60,0.85243,-0.17755,0.59755,-0.44945,0.60536,-0.38223,0.84356,-0.385
42,0.58212,-0.32192,0.56971,-0.29674,0.36946,-0.47357,0.56811,-0.511
71,0.41078,-0.46168,0.21266,-0.34090,0.42267,-0.54487,0.18641,-0.453
00]
yprev = predict(row, model)
print('Predicted: %.3f (class=%d)' % (yprev, yprev.round()))
```

```
Predicted: 0.851 (class=1)
```

In [ ]: