Perceptron Simples com Sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html)

```
from sklearn import preprocessing # biblioteca para suporte ao pré-processamento
from sklearn.model_selection import train_test_split # biblioteca para separação de amostras para treino e teste
from sklearn.linear_model import Perceptron # biblioteca com funções para a execução da RNA Perceptron
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import metrics # biblioteca para obtenção de métricas para avaliação dos modelos
import matplotlib.pyplot as plt # biblioteca para plotar gráfico
import numpy as np
import pandas as pd
import random # biblioteca aplicada na geração de números randômicos
from google.colab import drive
drive.mount('/content/drive') # Montando o Google Drive na mesma conta do Google Colab
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Caminho do dataset no Google Drive que será carregado em df
df = pd.read_csv("/content/drive/MyDrive/PosIA/RedesNeurais/Datasets/bancario.csv")

# Exibição dos dados
print(df)
```

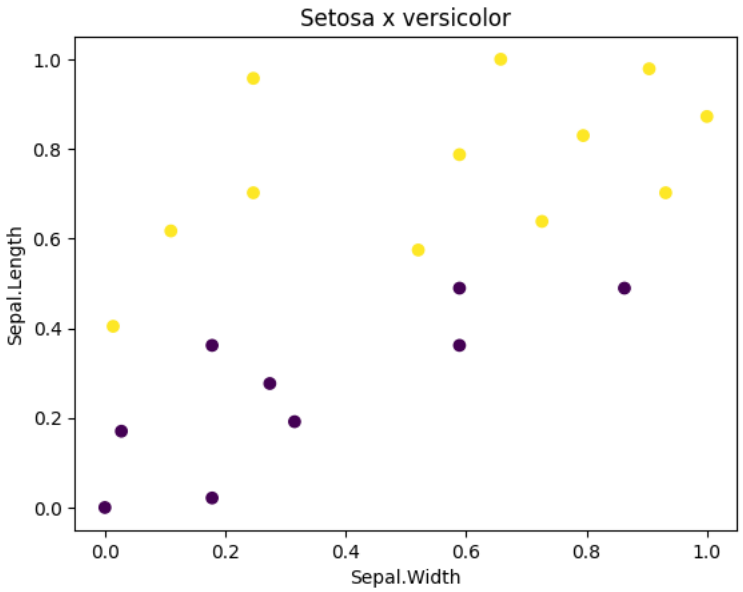
```
    Conta  Renda  Dívida Classe
0     101   2800     550    bom
1     102   1300     500    mau
2     103   1400      80    bom
3     104    500     200    mau
4     105   1100     270    mau
5     106   1800     450    bom
6     107   2400     650    bom
7     108   1950     600    bom
8     109    450      70    mau
9     110   2750     730    bom
10    111    850      90    mau
11    112   1300     200    mau
12    113   2100     750    bom
13    114    900     300    mau
14    115   2700     250    bom
15    116   1600     500    mau
16    117   1900     150    bom
17    118   2500     800    bom
18    119   1600     700    mau
19    120   2300     500    bom
20    121   2100     250    bom
```

```
# Pré-processamento de Dados
# Remoção da coluna 'Conta'
df = df.drop(columns=['Conta'])

# Convertendo a coluna 'Classe' para valores numéricos
df['Classe'] = df['Classe'].map({'bom': 1, 'mau': -1})

# Separando as características (features) e o alvo (target)
X_bancario = df.drop('Classe', axis=1).values
y_bancario = df['Classe'].values

print(y_bancario.shape)
print(X_bancario.shape)
```

```
(21,)
(21, 2)
```

```
# Normalizando os dados (features)
scaler = preprocessing.MinMaxScaler()
X_bancario = scaler.fit_transform(X_bancario)

# plotando o gráfico para verificação se as amostras são linearmente separáveis
plt.scatter(X_bancario[:,1],X_bancario[:,0],c=y_bancario)
plt.title("Setosa x versicolor" )
plt.xlabel('Sepal.Width')
plt.ylabel('Sepal.Length')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
# Dividindo os dados em conjunto de treino e teste
X_train_bancario, X_test_bancario, y_train_bancario, y_test_bancario = train_test_split(X_bancario, y_bancario, test_size=0.30, random_s


print(X_train_bancario.shape)
print(y_train_bancario.shape)
```

```
(14, 2)
(14,)
```

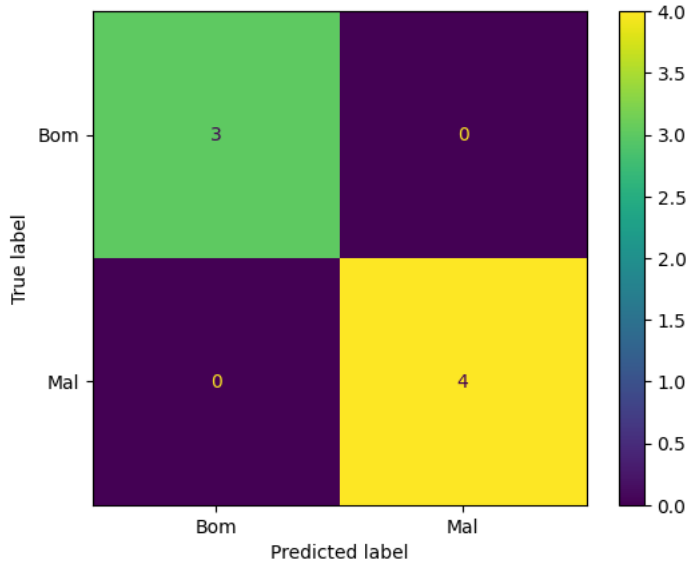
```
# Inicializando o modelo Perceptron com os mesmos parâmetros do programa original
p_bancario = Perceptron(random_state=42, eta0=0.0001, alpha=0.1)

# Treinando o modelo com os dados de treino
p_bancario.fit(X_train_bancario, y_train_bancario)
```

```
Perceptron
Perceptron(alpha=0.1, eta0=0.0001, random_state=42)
```

```
# Realizando previsões com os dados de treino e teste
predictions_train_bancario = p_bancario.predict(X_train_bancario) # validação do conjunto de amostras treinadas
predictions_test_bancario = p_bancario.predict(X_test_bancario) # validação do conjunto de amostras que não participaram do treinamento

# Avaliando o desempenho do modelo
train_score_bancario = accuracy_score(y_train_bancario, predictions_train_bancario) # avaliação de acurácia da classificação das amostra
test_score_bancario = accuracy_score(y_test_bancario, predictions_test_bancario) # avaliação de acurácia da classificação das amostras

# Mostrando as métricas de desempenho
print("Acurácia com dados de treinamento: ", train_score_bancario)
print("Acurácia com dados de teste: ", test_score_bancario)
print(classification_report(y_test_bancario, predictions_test_bancario))

print("Número de épocas no treinamento: ", p_bancario.n_iter_)
print("Lista de parâmetros configurados na Perceptron: ", p_bancario.get_params())
```

```
Acurácia com dados de treinamento:  0.7857142857142857
Acurácia com dados de teste:  1.0
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         4

    accuracy                           1.00         7
   macro avg       1.00      1.00      1.00         7
weighted avg       1.00      1.00      1.00         7

Número de épocas no treinamento:  6
Lista de parâmetros configurados na Perceptron:  {'alpha': 0.1, 'class_weight': None, 'early_stopping': False, 'eta0': 0.0001, 'fit_
```

```
# Apresentação gráfica da matriz de confusão dos testes classificados
conf_matrix = confusion_matrix(y_test_bancario, predictions_test_bancario)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = conf_matrix, display_labels = ['Bom', 'Mal'])
cm_display.plot()
plt.show()
```




```
# Testes com Novas Amostras
# Criando novas amostras para teste
new_A = np.array([[1500, 300]]) # Amostra aleatória 1
new_B = np.array([[2000, 800]]) # Amostra aleatória 2

# Normalizando as novas amostras
A = scaler.transform(new_A)
B = scaler.transform(new_B)

# Realizando previsões com as novas amostras
prediction_sample_1 = p_bancario.predict(A)
prediction_sample_2 = p_bancario.predict(B)

print("Previsão para nova amostra 1: ", prediction_sample_1)
print("Previsão para nova amostra 2: ", prediction_sample_2)
```

```
Previsão para nova amostra 1:  [1]
Previsão para nova amostra 2:  [1]
```