Atividade: *Perceptron* - Petróleo

INSTITUTO FEDERAL DE MINAS GERAIS

Departamento de Engenharia e Computação

Professor: Ciniro Nametala Leite

Aluno: Luan Carlos dos Santos

Pacotes necessários

In [227…
```
import numpy as np
import pandas as pd
import pygwalker as pyg
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import plotly.graph_objects as go
```

Funções

- Degrau Bipolar

In [227…
```
def degrau_bipolar(u):
   if u >= 0:
      y = 1 # óleo P1
   else:
      y = -1 # óleo P2
   return y
```

- Previsão

In [227…
```
def previsao(w, x):
   u = np.dot(w.T, x)
   yhat = degrau_bipolar(u)
   return yhat # esse y hat é a previsão da perceptron
```

Conjunto de dados

- Coletando os dados de treinamento;

In [227…
```
dados = pd.read_csv('../data-sets/tabela_treino.csv', sep=';')
dados.head()
```

Out[227…

	Amostra	x1	x2	x3	d
0	1	-0.6508	0.1097	40.009	-1.0
1	2	-14.4920	0.8896	44.005	-1.0
2	3	2.0850	0.6876	79.845	-1.0
3	4	0.2626	11.4760	12.071	-1.0
4	5	0.6418	10.2340	77.985	1.0

- Armazenando a quantidade de amostras;

In [227…

```
n_amostras = dados.shape[0]
```

- Armazenando a quantidade de variáveis e a matriz treinamento;

In [227…

```
n_variaveis = dados.shape[1] -1
x = dados.iloc[:, 1:n_variaveis].values
pd.DataFrame(x)[:5]
```

Out[227…

	0	1	2
0	-0.6508	0.1097	40.009
1	-14.4920	0.8896	44.005
2	2.0850	0.6876	79.845
3	0.2626	11.4760	12.071
4	0.6418	10.2340	77.985

- Inserindo o Bias na matriz de treinamento;

In [227…

```
# Criando um vetor preenchido com '1s' * -1
bias = np.ones((n_amostras, 1)) * -1
x = np.hstack((x, bias))
pd.DataFrame(x)[:5]
```

Out[227…

	0	1	2	3
0	-0.6508	0.1097	40.009	-1.0
1	-14.4920	0.8896	44.005	-1.0
2	2.0850	0.6876	79.845	-1.0
3	0.2626	11.4760	12.071	-1.0
4	0.6418	10.2340	77.985	-1.0

- Armazenando o vetor de saida;

In [227…
```
y = dados.iloc[:, n_variaveis].values
print(y)
print(f"Quantidade de amostras tipo P1(1): {(y == 1).sum()}")
print(f"Quantidade de amostras tipo P2(-1): {(y== -1).sum()}")
```

```
[-1. -1. -1. -1.  1.  1. -1.  1. -1.  1. -1. -1. -1. -1. -1. -1.  1.  1.
  1.  1. -1.  1.  1.  1.  1. -1. -1.  1. -1.  1.]
Quantidade de amostras tipo P1(1): 14
Quantidade de amostras tipo P2(-1): 16
```

- Inicializando o vetor de pesos w com valores aleatórios de 0 a 1;

In [228…
```
w = np.random.uniform(0, 1, n_variaveis)
print(w)
```

```
[0.85876398 0.31770244 0.00159796 0.65921372]
```

Pré configurações de Treinamento

- Definindo a taxa de aprendizagem η, a tolerância e o erro médio inicial;

In [228…
```
eta = 0.01
tolerancia = 0.01
erro_medio = tolerancia + 1
```

- Criando um vetor de épocas, um contador e o máximo de épocas;

In [228…
```
erros_epocas = []
epoca = 0
max_epocas = 800
```

- Inicializando os critérios de parada do treinamento;

In [228…
```
criterio_erro = True
criterio_epoca = True
```

Treinamento

"Equanto o erro for maior que a tolerância e o máximo de épocas não for atingido, o treinamento irá se repetir"

In [228…
```
n_treinamentos = 5
pesos_iniciais = []
pesos_finais = []

for treino in range(n_treinamentos):

    # reinicializando os pesos aleatórios
    w = np.random.uniform(0, 1, n_variaveis)
    pesos_iniciais.append(w.copy())

    # reinicia variáveis de controle
    criterio_erro = True
```

```
    criterio_epoca = True
    epoca = 0
    erros_epocas = []

    while criterio_erro and criterio_epoca:
        erro_atual = 0
        epoca += 1

        # embaralhando os indíces pra usar na rede
        indice_aleatorios = np.random.permutation(n_amostras)

        # para cada amostra no conjunto de amostras
        for i in range(n_amostras):
            # pegando indice_aleatorio pra amostra i
            i_amostra = indice_aleatorios[i]

            # capturando uma amostra da matriz de treinamento vinculada ao indice c
            amostra_atual = x[i_amostra,:]

            # calculando o potencial de ativação
            u = np.dot(w.T, amostra_atual)

            # obtendo previsão da rede pela Degrau Bipolar
            yhat = degrau_bipolar(u)

            # calculando o erro entre a previsão e o esperado
            e = y[i_amostra] - yhat

            # atualizando os pesos pela regra de HEBB
            w = w + ((eta * e) * amostra_atual)

            # acumulando os erros de de todas as amostras
            erro_atual += np.abs(e)

        # calcular o erro médio da época
        erro_medio = erro_atual/n_amostras

        # adicionando o erro medio ao vetor de erros do treinamento
        erros_epocas.append(erro_medio)
        # verificando e atualizando os critérios de parada
        criterio_erro = (erro_medio > tolerancia)
        criterio_epoca = (epoca < max_epocas)
    pesos_finais.append(w.copy())
    # print(f'Época: {epoca}')

# Cria nomes das colunas
colunas_iniciais = [f'w{i+1}' for i in range(n_variaveis)]
colunas_finais = [f'w{i+1}' for i in range(n_variaveis)]
colunas = colunas_iniciais + colunas_finais

# Junta pesos iniciais e finais
dados = np.hstack([pesos_iniciais, pesos_finais])

# Cria coluna de treinos
treinos = [f'Treino {i+1}' for i in range(n_treinamentos)]

# Cria DataFrame simples
df_pesos = pd.DataFrame(dados, columns=colunas)
df_pesos.insert(0, 'Treino', treinos)
```

```
df_pesos
```

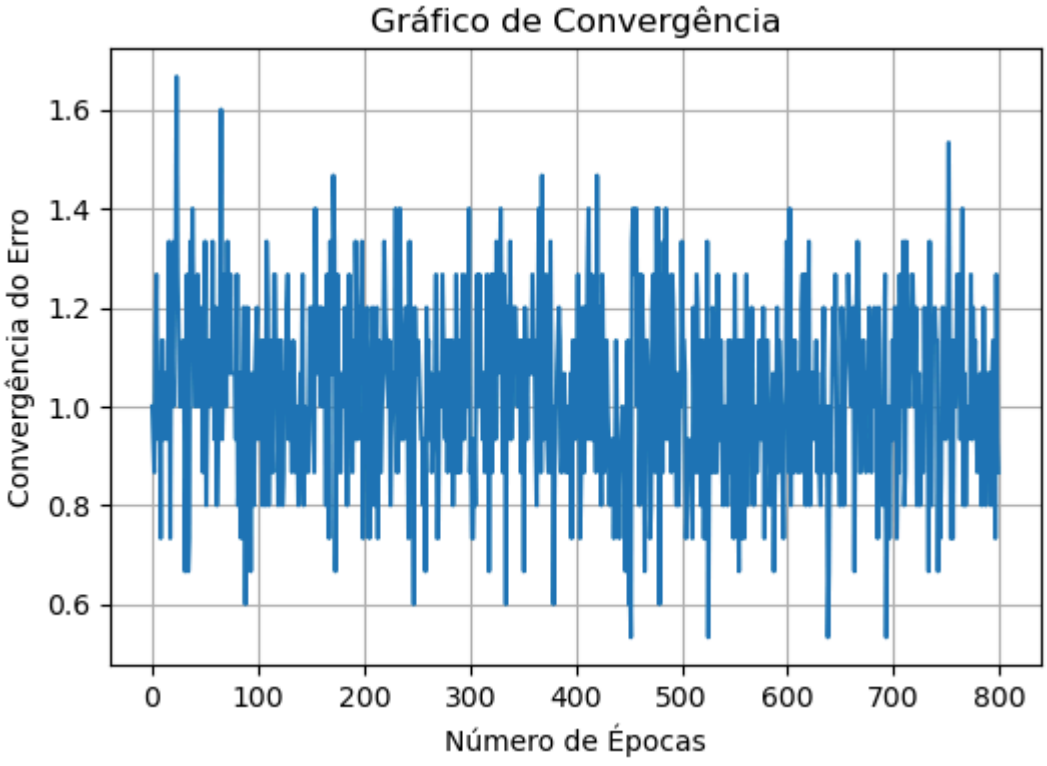
Out[228…

	Treino	w1	w2	w3	w4	w1	w2	w3	
0	Treino 1	0.917757	0.518431	0.985282	0.263550	-1.788005	1.719353	0.047842	10.08
1	Treino 2	0.599565	0.170372	0.306524	0.505823	-1.035439	1.107724	-0.841716	9.64
2	Treino 3	0.793813	0.914021	0.904940	0.779954	-1.508937	-0.042329	-0.109200	9.19
3	Treino 4	0.557972	0.393620	0.471455	0.930546	-1.377002	1.234236	-0.562385	9.65
4	Treino 5	0.967017	0.145762	0.155431	0.401817	-1.549567	0.970528	-0.166949	10.14

Análise de Convergência do Algorítmo com base no erro de cada época:

In [228…

```
plt.figure(figsize=(6,4))
plt.plot((erros_epocas[:max_epocas]))
plt.title('Gráfico de Convergência')
plt.xlabel('Número de Épocas')
plt.ylabel('Convergência do Erro')
plt.grid(True)
plt.show()
```



- Pesos Ideiais $w*$

In [228…
```
print(f'Quantidade de épocas até convergir "{epoca}"')
#exibindo os pesos calibrados pela regra Regra de Hebb
print('Conjunto ideal de pesos em cada treinamento')
pd.DataFrame(pesos_finais)
colunas_finais = [f'w{i+1}' for i in range(n_variaveis)]
colunas = colunas_finais

# Junta pesos iniciais e finais
dados = np.hstack([pesos_finais])

# Cria DataFrame simples
df_pesos = pd.DataFrame(dados, columns=colunas)

df_pesos
```

```
Quantidade de épocas até convergir "800"
Conjunto ideal de pesos em cada treinamento
```

Out[228…

	w1	w2	w3	w4
0	-1.788005	1.719353	0.047842	10.083550
1	-1.035439	1.107724	-0.841716	9.645823
2	-1.508937	-0.042329	-0.109200	9.199954
3	-1.377002	1.234236	-0.562385	9.650546
4	-1.549567	0.970528	-0.166949	10.141817

- Previsões dos Testes

In [228…
```
previsoes = []

for j in range(n_treinamentos):
   # vetor que recebe as previsões da rna para o treino j
   yhat = np.zeros(n_amostras)

   for i in range(n_amostras):
      yhat[i] = previsao(pesos_finais[j], x[i, :])

   # adiciona o vetor completo de previsões à lista
   previsoes.append(yhat)
pd.DataFrame(previsoes)
```

Out[228…

	0	1	2	3	4	5	6	7	8	9	…	20	21	22	23	24
0	-1.0	1.0	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	…	-1.0	-1.0	1.0	-1.0	-1.0
1	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	…	-1.0	-1.0	-1.0	-1.0	-1.0
2	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	…	-1.0	-1.0	1.0	-1.0	-1.0
3	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	…	-1.0	-1.0	-1.0	-1.0	-1.0
4	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	…	-1.0	-1.0	1.0	-1.0	-1.0

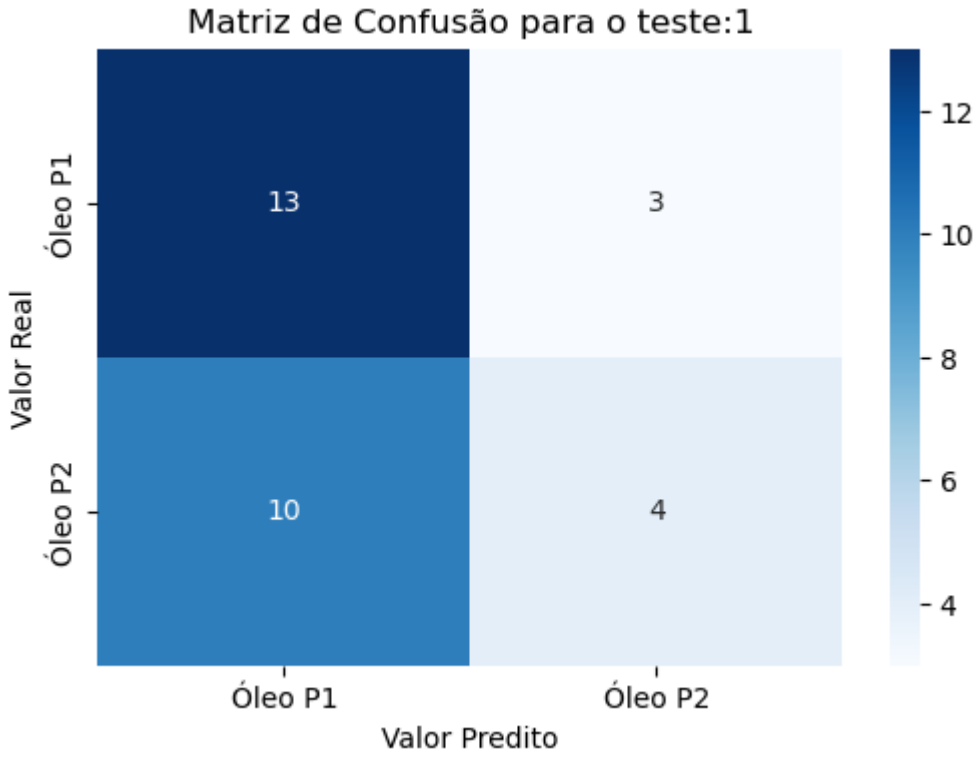
5 rows × 30 columns

Análise da Qualidade da Classificação

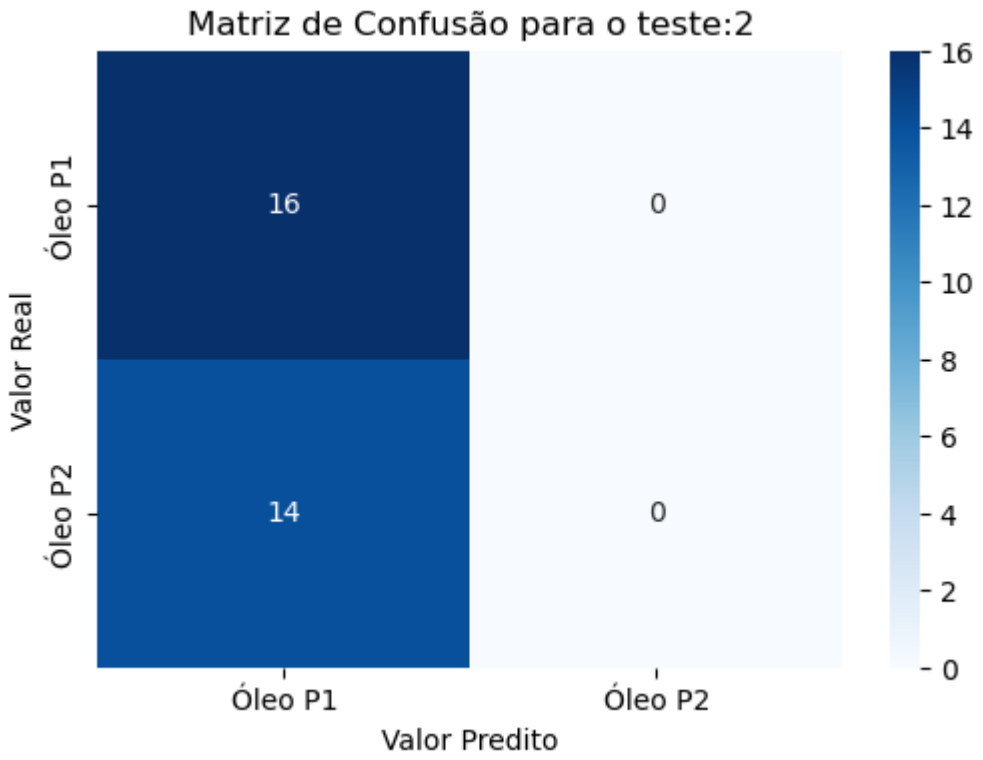
- Matriz de confusão

In [228…

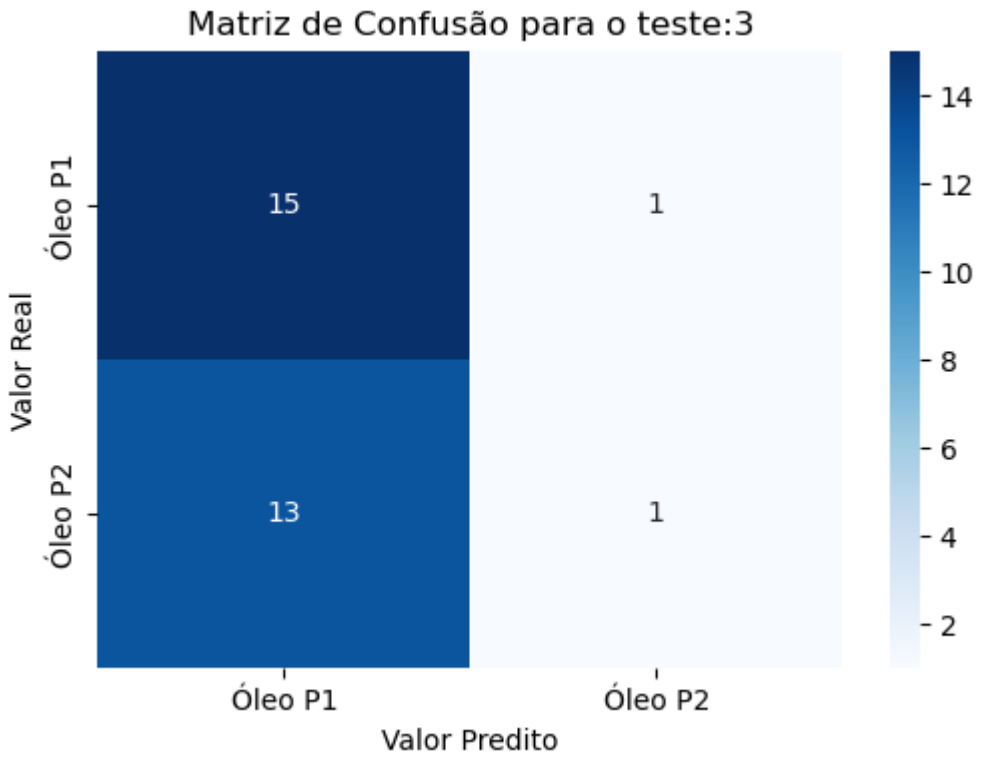
```
# imprimir todas as previsões
for j, yhat in enumerate(previsoes):
    #converter -1 em 0 para melhor visualizar a matriz de confusão
   yhat_display = np.where(yhat == -1, 0, yhat)
   y_display = np.where(y == -1, 0, y)

   #gera a matriz de confusão
   cm = confusion_matrix(y_display, yhat_display)

   #plotar matriz de confusão crocante
   plt.figure(figsize=(6, 4))
   sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
               xticklabels=['Óleo P1', 'Óleo P2'],
               yticklabels=['Óleo P1', 'Óleo P2'])
   plt.title(f'Matriz de Confusão para o teste:{j+1}')
   plt.xlabel('Valor Predito')
   plt.ylabel('Valor Real')
   plt.show()
   acuracia = (np.sum(yhat == y) / n_amostras) * 100
   print(f'Acerto percentual do treinamento: {acuracia}%')
```



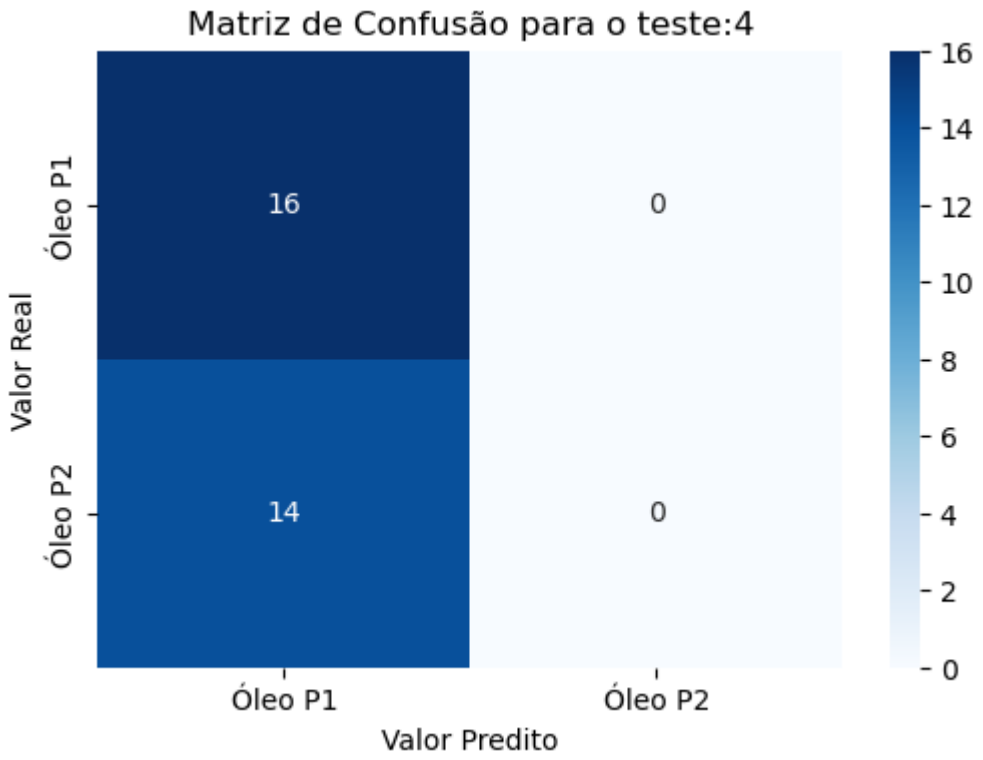

```
Acerto percentual do treinamento: 56.666666666666664%
```



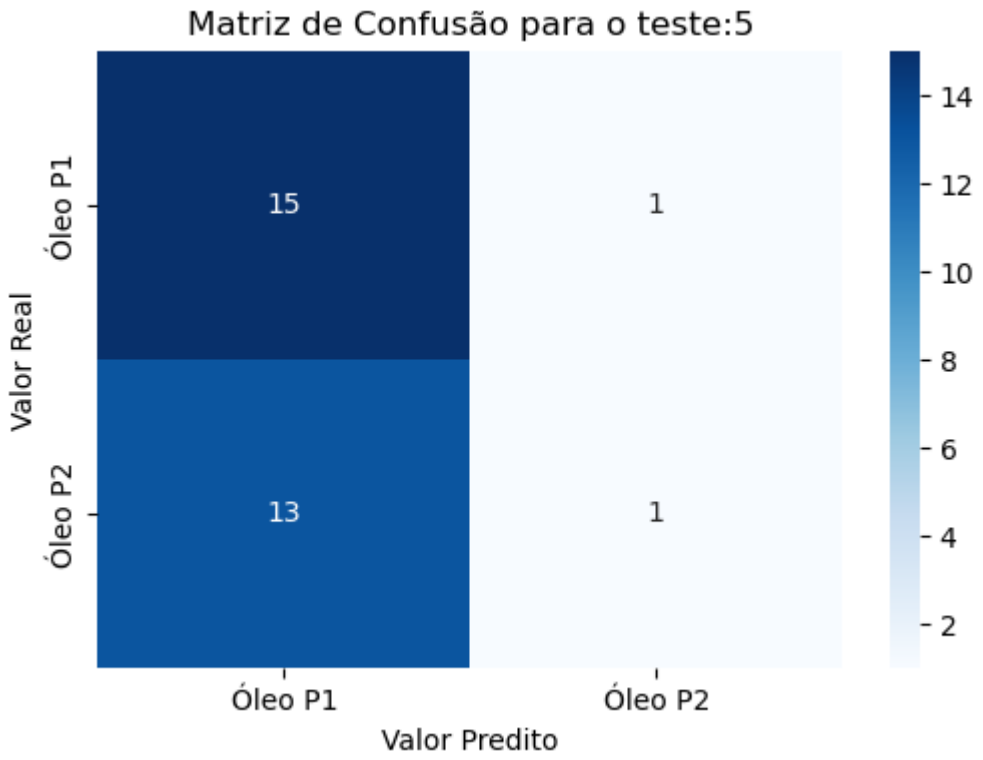

```
Acerto percentual do treinamento: 53.333333333333336%
```




```
Acerto percentual do treinamento: 53.333333333333336%
```




```
Acerto percentual do treinamento: 53.333333333333336%
```




```
Acerto percentual do treinamento: 53.333333333333336%
```

Implementação da rede *Perceptron*

In [229…

```
# lê os dados de teste
dados_classificacao = pd.read_csv('../data-sets/tabela_classificacao.csv', sep='
n_amostras = dados_classificacao.shape[0]

# adiciona bias
bias = np.ones((n_amostras, 1)) * -1  # usar 1 se os pesos foram treinados com b
x_novo = np.hstack((dados_classificacao.to_numpy(), bias))
```

```
# calcula previsões para cada treino
previsoes = []
for j in range(n_treinamentos):
    yhat = np.zeros(n_amostras)
    for i in range(n_amostras):
        yhat[i] = previsao(pesos_finais[j], x_novo[i, :])
    previsoes.append(yhat)

# transforma previsões em DataFrame (uma coluna por treino)
df_previsoes = pd.DataFrame(
    np.column_stack(previsoes),
    columns=[f'ŷ treino {j+1}' for j in range(n_treinamentos)]
)

# junta previsões com os dados originais do teste
df_final = pd.concat([dados_classificacao.reset_index(drop=True), df_previsoes],

# exibe a tabela final
df_final
```

Out[229…

	x1	x2	x3	ŷ treino 1	ŷ treino 2	ŷ treino 3	ŷ treino 4	ŷ treino 5
0	-0.3665	0.0620	5.9891	-1.0	-1.0	-1.0	-1.0	-1.0
1	-0.7842	1.1267	5.5912	-1.0	-1.0	-1.0	-1.0	-1.0
2	0.3012	0.5611	5.8234	-1.0	-1.0	-1.0	-1.0	-1.0
3	0.7757	1.0648	8.0677	-1.0	-1.0	-1.0	-1.0	-1.0
4	0.1570	0.8028	6.3040	-1.0	-1.0	-1.0	-1.0	-1.0
5	-0.7014	1.0316	3.6005	-1.0	-1.0	-1.0	-1.0	-1.0
6	0.3748	0.1536	6.1537	-1.0	-1.0	-1.0	-1.0	-1.0
7	-0.6920	0.9404	4.4058	-1.0	-1.0	-1.0	-1.0	-1.0
8	-1.3970	0.7141	4.9263	-1.0	-1.0	-1.0	-1.0	-1.0
9	-1.8842	0.2805	1.2548	-1.0	-1.0	-1.0	-1.0	-1.0

In [ ]: