

Modelo para o Sensor CEI

Este dataset "**DataCEI.csv**" possui informações dispostas em colunas sobre as características dos objetos que passam pelo sensor:

- **Tamanho:** Segue a classificação do CEI2020 (Tamanho='0' - Grande 100%).
- **Referencia:** Referência dinâmica do *Threshold.
- **NumAmostra:** Número de amostras adquiridas.
- **Area:** Somatório das Amplitudes das amostras.
- **Delta:** Máxima Amplitude da amostra.
- **Output1:** Peça tipo 1.
- **Output2:** Peça tipo 2.

Bibliotecas

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

#Função do cálculo da sigmóide
def relu(x):
    return 1/(1+np.exp(-x))
```

Carregando os dados

Vamos começar lendo o arquivo DataCEI.csv em um dataframe do pandas.

In [2]:

```
DataSet=pd.read_csv('arruela_.csv')
```

In [3]:

```
DataSet.head()
```

Out[3]:

	Hora	Tamanho	Referencia	NumAmostra	Area	Delta	Output1	Output2
0	13:00:06	53	25	69	81	68	1	0
1	13:00:07	53	26	89	87	56	1	0
2	13:00:08	53	27	68	69	55	1	0
3	13:00:09	53	28	36	50	80	1	0
4	13:00:10	53	29	71	72	50	1	0

In [4]:

```
DataSet.drop(['Hora', 'Tamanho', 'Referencia'],axis=1,inplace=True)
```

In [5]:

```
DataSet.head()
```

Out[5]:

	NumAmostra	Area	Delta	Output1	Output2
0	69	81	68	1	0
1	89	87	56	1	0
2	68	69	55	1	0
3	36	50	80	1	0
4	71	72	50	1	0

In [6]:

```
DataSet.describe()
```

Out[6]:

	NumAmostra	Area	Delta	Output1	Output2
count	294.000000	294.000000	294.000000	294.000000	294.000000
mean	58.452381	62.238095	53.721088	0.333333	0.666667
std	17.327682	30.355535	34.816885	0.472208	0.472208
min	3.000000	6.000000	17.000000	0.000000	0.000000
25%	49.000000	44.000000	37.000000	0.000000	0.000000
50%	57.500000	55.000000	43.000000	0.000000	1.000000
75%	68.000000	67.000000	53.000000	1.000000	1.000000
max	120.000000	201.000000	251.000000	1.000000	1.000000

Váriaveis do Dataset

In [7]:

```
DataSet.columns
```

Out[7]:

```
Index(['NumAmostra', 'Area', 'Delta', 'Output1', 'Output2'], dtype='object')
```

Número de Peças

Vamos classificar os grupos pelo número de peças:

1. Grupo com uma peça
2. Grupo com duas peças

In [8]:

```
sns.set_style('whitegrid')
sns.countplot(x='Output2',data=DataSet,palette='RdBu_r')
plt.show()
```

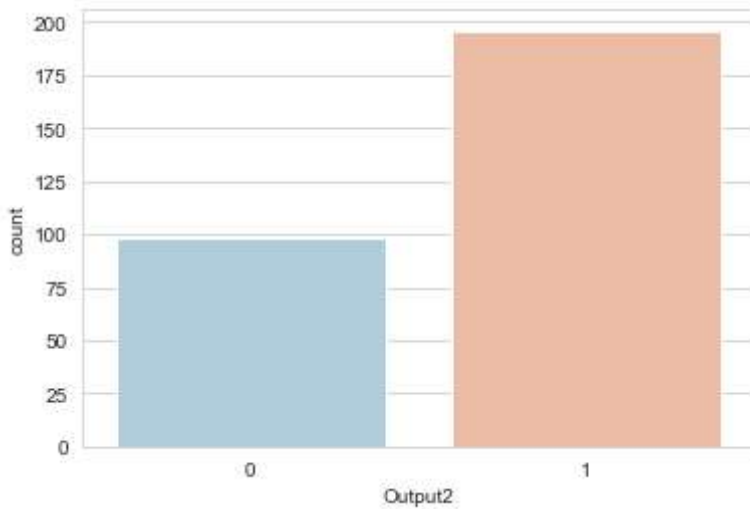


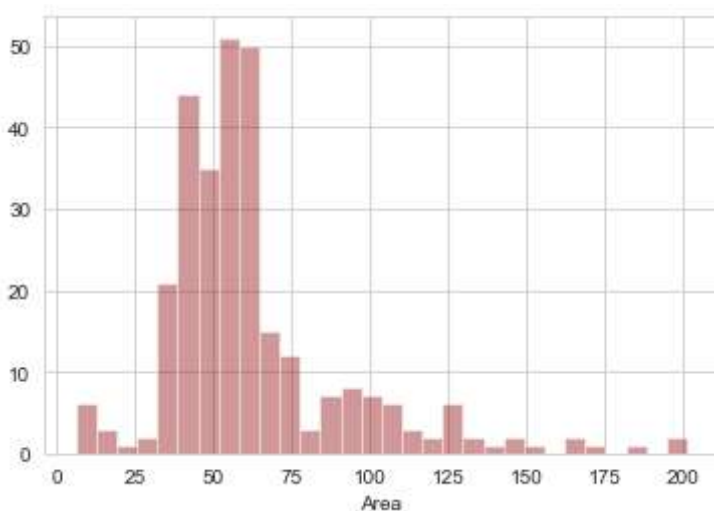
Gráfico da distribuição das áreas das peças

In [9]:

```
sns.distplot(DataSet['Area'].dropna(),kde=False,color='darkred',bins=30)
plt.show()
```

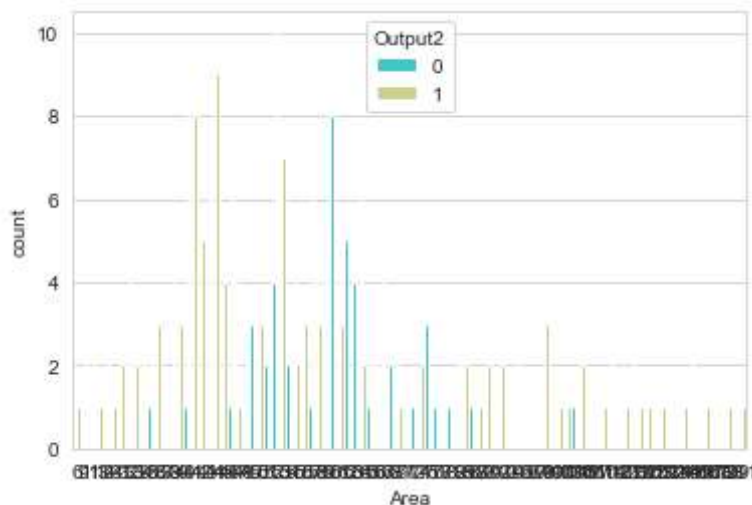
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

warnings.warn(msg, FutureWarning)



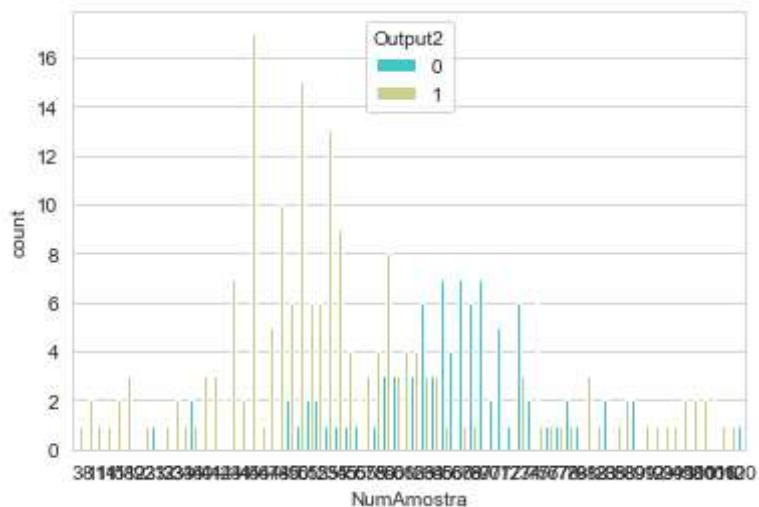
In [10]:

```
sns.set_style('whitegrid')
sns.countplot(x='Area',hue='Output2',data=DataSet,palette='rainbow')
plt.show()
```



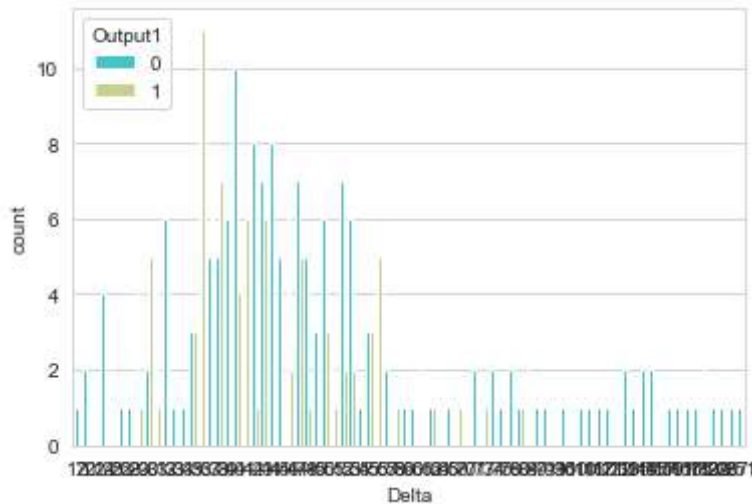
In [11]:

```
sns.set_style('whitegrid')
sns.countplot(x='NumAmostra',hue='Output2',data=DataSet,palette='rainbow')
plt.show()
```



In [12]:

```
sns.set_style('whitegrid')
sns.countplot(x='Delta', hue='Output1', data=DataSet, palette='rainbow')
plt.show()
```



As variáveis preditoras e a variável de resposta

Para treinar o modelo de regressão, primeiro precisaremos dividir nossos dados em uma matriz **X** que contenha os dados das variáveis preditoras e uma matriz **y** com os dados da variável de destino.

Matrizes X e y

In [13]:

```
#X = DataSet[['NumAmostra', 'Area', 'Delta']]
#y = DataSet[['Output1', 'Output2']]
```

Relação entre as variáveis preditoras

Algumas questões importantes

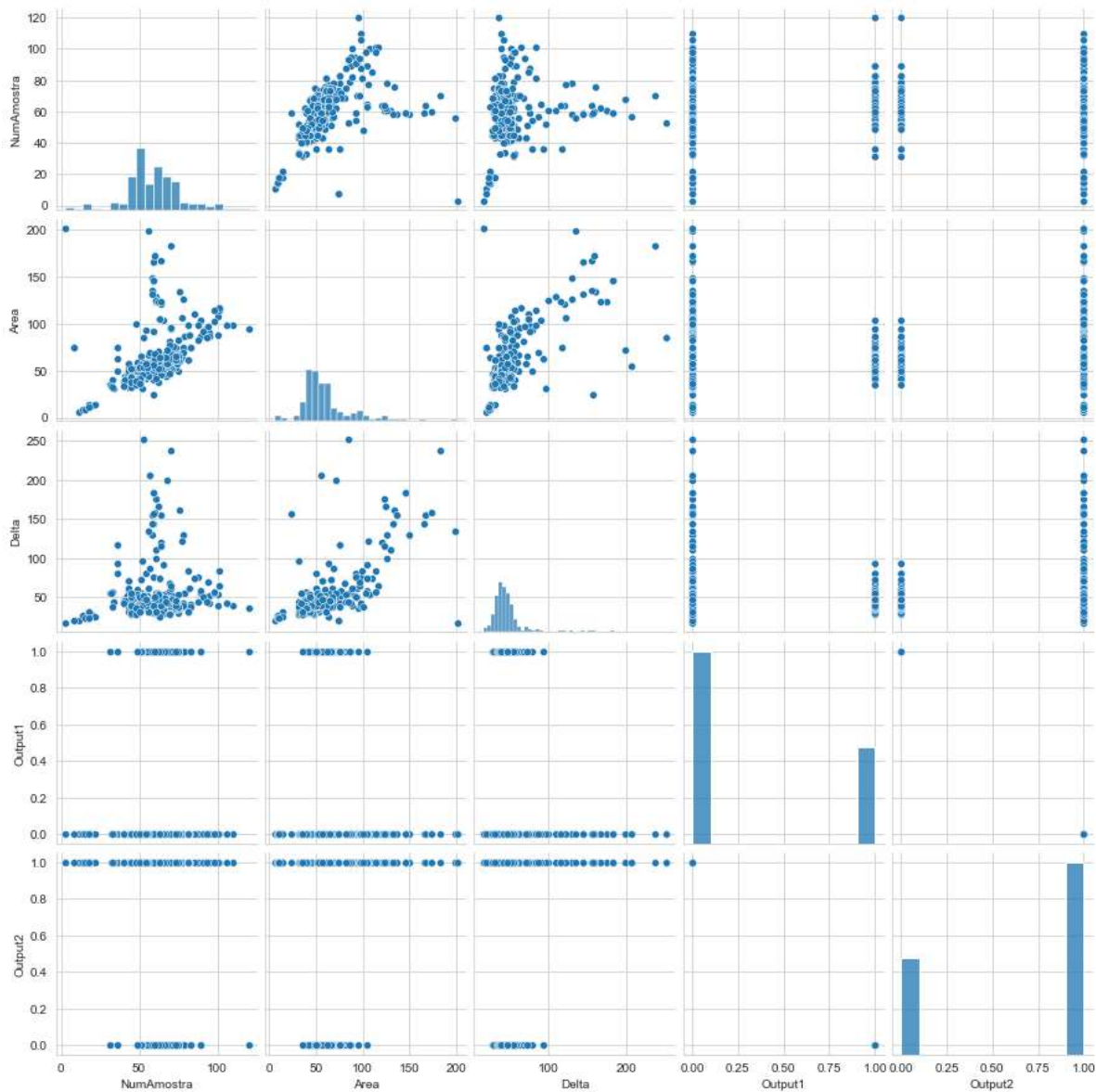
1. Pelo menos um dos preditores **x1, x2, ..., x5** é útil na previsão da resposta?
2. Todos os preditores ajudam a explicar **y**, ou apenas um subconjunto dos preditores?
3. Quão bem o modelo se ajusta aos dados?
4. Dado um conjunto de valores de previsão, quais valores de resposta devemos prever e quais as métricas indicam um bom modelo de previsão?

Gráficos simples de dispersão

Pelos gráficos abaixo percebemos ... nossa variável de resposta

In [14]:

```
sns.pairplot(DataSet)
plt.show()
```

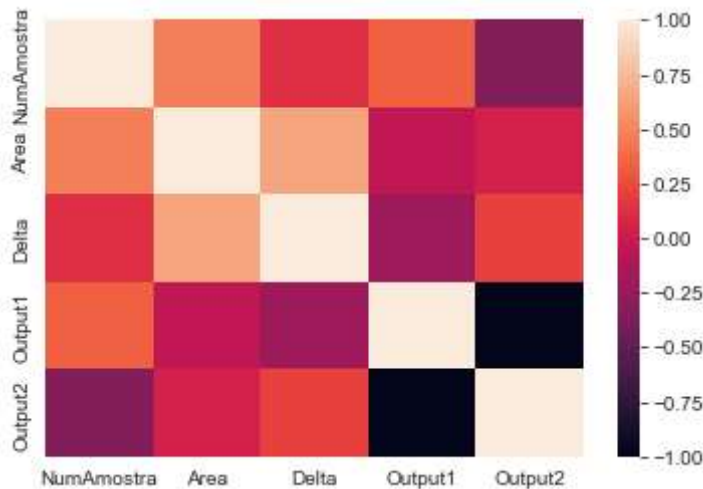


Mapa de Calor

O gráfico abaixo mostra através de uma escala de cores a correlação entre as variáveis do *Dataset*. Se observarmos as cores deste gráfico, a variável preditora '**Area**' possui maior correlação com a variável de resposta '**Output**' e a variável '**NumAmostra**' a menor.

In [15]:

```
sns.heatmap(DataSet.corr())
plt.show()
```



Normalização dos Dados

In [16]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
DataScaled=scaler.fit_transform(DataSet)
DataSetScaled=pd.DataFrame(np.array(DataScaled),columns = ['NumAmostra', 'Area', 'Delta', 'Output1', 'Output2'])
```

In [17]:

```
DataSetScaled.head()
```

Out[17]:

	NumAmostra	Area	Delta	Output1	Output2
0	0.609753	0.619126	0.410814	1.414214	-1.414214
1	1.765943	0.817120	0.065566	1.414214	-1.414214
2	0.551943	0.223137	0.036795	1.414214	-1.414214
3	-1.297961	-0.403846	0.756062	1.414214	-1.414214
4	0.725372	0.322134	-0.107058	1.414214	-1.414214

Conjunto de dados para o treinamento

In [18]:

```
X = DataSetScaled.drop(['Output1', 'Output2'],axis=1)
y = DataSet[['Output1', 'Output2']]
```

Separando os dados de treinamento e de validação

Agora vamos dividir os dados em um conjunto de treinamento e um conjunto de testes. Vamos treinar o modelo no conjunto de treinamento, em seguida, usar o conjunto de teste para validar o modelo.

Em nosso exemplo iremos separar de forma randômica 33% dos dados para validação. Estes dados não serão utilizados para determinação dos coeficientes preditores do modelo.

In [19]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)

print(y_test)
print(X_test)
```

	Output1	Output2
147	0	1
81	1	0
288	0	1
267	0	1
167	0	1
..
156	0	1
256	0	1
57	1	0
37	1	0
34	1	0

[89 rows x 2 columns]

	NumAmostra	Area	Delta
147	1.534705	1.576099	0.612209
81	-0.199580	-0.238851	-0.049517
288	-0.604247	-0.898832	-0.711242
267	-0.488628	-0.304849	-0.107058
167	-0.488628	-0.568841	-0.509848
..
156	0.147277	2.071086	1.331475
256	0.956610	0.652125	0.008024
57	0.609753	-0.106854	-0.423536
37	0.378515	-0.238851	-0.509848
34	0.205086	-0.304849	-0.394765

[89 rows x 3 columns]

Criando o Modelo de MPL

In [20]:

```
#Tamanho do DataSet de Treinamento
n_records, n_features = X_train.shape

#Arquitetura da MPL
N_input = 3
N_hidden = 8
N_output = 2
learnrate = 0.1
```

Inicialização dos pesos da MPL (Aleatório)

In [21]:

```
#Pesos da Camada Oculta (Inicialização Aleatória)
weights_input_hidden = np.random.normal(0, scale=0.1, size=(N_input, N_hidden))
print('Pesos da Camada Oculta:')
print(weights_input_hidden)

#Pesos da Camada de Saída (Inicialização Aleatória)
weights_hidden_output = np.random.normal(0, scale=0.1, size=(N_hidden, N_output))
print('Pesos da Camada de Saída:')
print(weights_hidden_output)
```

Pesos da Camada Oculta:

```
[[ -0.20841296 -0.0800278  -0.10659101  0.00695963 -0.074624    0.00980341
   0.11335053  0.10314534]
 [ -0.00524388 -0.09295535  0.17440083  0.09471292 -0.08611179  0.07055988
   0.10844329 -0.14234351]
 [ -0.22414069 -0.00361055 -0.23365955 -0.19340887 -0.04156754  0.04360085
   0.02618358 -0.08009075]]
```

Pesos da Camada de Saída:

```
[[ 0.11894226 -0.22344832]
 [-0.00535678  0.05537861]
 [ 0.02226043 -0.1833154 ]
 [-0.04831271 -0.0890571 ]
 [-0.05005091  0.14304487]
 [ 0.14019691  0.07762114]
 [-0.03047457 -0.09680561]
 [ 0.0194028  -0.11987589]]
```

Algoritmo Backpropagation

In [22]:

```

epochs = 200000
last_loss=None
EvolucaoError=[]
IndiceError=[]

for e in range(epochs):
    delta_w_i_h = np.zeros(weights_input_hidden.shape)
    delta_w_h_o = np.zeros(weights_hidden_output.shape)
    for xi, yi in zip(X_train.values, y_train.values):

# Forward Pass
        #Camada oculta
        #Calcule a combinação linear de entradas e pesos sinápticos
        hidden_layer_input = np.dot(xi, weights_input_hidden)
        #Aplicado a função de ativação
        hidden_layer_output = relu(hidden_layer_input)

        #Camada de Saída
        #Calcule a combinação linear de entradas e pesos sinápticos
        output_layer_in = np.dot(hidden_layer_output, weights_hidden_output)

        #Aplicado a função de ativação
        output = relu(output_layer_in)
        #print('As saídas da rede são',output)

#-----

# Backward Pass
        ## TODO: Cálculo do Erro
        error = yi - output

        # TODO: Calcule o termo de erro de saída (Gradiente da Camada de Saída)
        output_error_term = error * output * (1 - output)

        # TODO: Calcule a contribuição da camada oculta para o erro
        hidden_error = np.dot(weights_hidden_output,output_error_term)

        # TODO: Calcule o termo de erro da camada oculta (Gradiente da Camada Oculta)
        hidden_error_term = hidden_error * hidden_layer_output * (1 - hidden_layer_output)

        # TODO: Calcule a variação do peso da camada de saída
        delta_w_h_o += output_error_term*hidden_layer_output[:, None]

        # TODO: Calcule a variação do peso da camada oculta
        delta_w_i_h += hidden_error_term * xi[:, None]

#Atualização dos pesos na época em questão
weights_input_hidden += learnrate * delta_w_i_h / n_records
weights_hidden_output += learnrate * delta_w_h_o / n_records

# Imprimir o erro quadrático médio no conjunto de treinamento

    if e % (epochs / 20) == 0:
        hidden_output = relu(np.dot(xi, weights_input_hidden))
        out = relu(np.dot(hidden_output,
                           weights_hidden_output))
        loss = np.mean((out - yi) ** 2)

        if last_loss and last_loss < loss:

```

```
print("Erro quadrático no treinamento: ", loss, " Atenção: O erro está aumentando")
else:
    print("Erro quadrático no treinamento: ", loss)
last_loss = loss

EvolucaoError.append(loss)
IndiceError.append(e)
```

```
Erro quadrático no treinamento: 0.2160400628529528
Erro quadrático no treinamento: 0.042575727201295724
Erro quadrático no treinamento: 0.01444269642120324
Erro quadrático no treinamento: 0.009572146505768556
Erro quadrático no treinamento: 0.007236946149244231
Erro quadrático no treinamento: 0.005776398925434333
Erro quadrático no treinamento: 0.004733860765355718
Erro quadrático no treinamento: 0.0039496790940691355
Erro quadrático no treinamento: 0.0033552640845169795
Erro quadrático no treinamento: 0.0029002841158349978
Erro quadrático no treinamento: 0.0026748182560845032
Erro quadrático no treinamento: 0.0025541715712256476
Erro quadrático no treinamento: 0.0022197181531234053
Erro quadrático no treinamento: 0.0018503724519056072
Erro quadrático no treinamento: 0.001509350771618915
Erro quadrático no treinamento: 0.0012129078327091922
Erro quadrático no treinamento: 0.0009636078498533028
Erro quadrático no treinamento: 0.0007587111623780258
Erro quadrático no treinamento: 0.0005931612324752293
Erro quadrático no treinamento: 0.0004611399668610431
```

In [23]:

```
### Gráfico da Evolução do Erro
```

In [24]:

```
plt.plot(IndiceError, EvolucaoError, 'r') # 'r' is the color red
plt.xlabel('')
plt.ylabel('Erro Quadrático')
plt.title('Evolução do Erro no treinamento da MPL')
plt.show()
```



Validação do modelo

In [25]:

```

# Calcule a precisão dos dados de teste
n_records, n_features = X_test.shape
predictions=0

for xi, yi in zip(X_test.values, y_test.values):

# Forward Pass
    #Camada oculta
    #Calcule a combinação linear de entradas e pesos sinápticos
    hidden_layer_input = np.dot(xi, weights_input_hidden)
    #Aplicado a função de ativação
    hidden_layer_output = relu(hidden_layer_input)

    #Camada de Saída
    #Calcule a combinação linear de entradas e pesos sinápticos
    output_layer_in = np.dot(hidden_layer_output, weights_hidden_output)

    #Aplicado a função de ativação
    output = relu(output_layer_in)

#-----

#Cálculo do Erro da Predição
    ## TODO: Cálculo do Erro
    if (output[0]>output[1]):
        if (yi[0]>yi[1]):
            predictions+=1

    if (output[1]>=output[0]):
        if (yi[1]>yi[0]):
            predictions+=1

print("A Acurácia da Predição é de: {:.3f}".format(predictions/n_records))

```

A Acurácia da Predição é de: 0.899

In [26]:

```
print("A Acurácia da Predição é de: {:.3f}".format(predictions/n_records))
```

A Acurácia da Predição é de: 0.899

In []:

In []:

In []:

In []: